VOL. 11, NO. 22, NOVEMBER 2016 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

COMPRESSION FUNCTION BASED ON PERMUTATIONS AND QUASIGROUPS

Zahraddeen A. Pindar, Sapiee Jamel, Abdulkadir H. Disina and Mustafa Mat Deris
Department of Information Security, Universiti Tun Hussein Onn Malaysia, Malaysia
E-Mail: deenpindar@gmail.com

ABSTRACT

Cryptographic hash functions are used to protect the integrity of information. Hash functions are implemented in applications such as; Message Authentication Codes, pseudo random number generators and key derivation functions. Thus, this arguably suggests the need for continuous development of hash functions. Traditionally, hash functions are designed based on existing block ciphers due to challenges and difficulties faced in constructing new hash functions from the scratch. However, the key generation for each encryption process results to huge computational cost. In order to reduce computational cost, only a limited instantiations of the block cipher such as the permutations and boolean operators are used as the underlying compression functions. Few works have been proposed in developing a less computational cost but secure and efficient compression function. This paper proposes a different approach (PQ and 3PQ) in constructing compression function based on permutations and non-associative quasigroup. Analysis of experimentation results have demonstrated that the proposed compression functions are suitable for operation in constraints environments (both memory and processing power) with very minimal computational cost. Similarly, the obtained results also shows the proposed compression functions have an effective one-way function, strong avalanche property and easy to implement.

Keywords: hash function, compression function, quasigroups, permutations.

1. INTRODUCTION

Cryptographic hash functions are used to protect the integrity of information. This is achieved by creating a unique relationship between the input (information) and the output (tag value), such that it is difficult to find same tag value for different inputs. Hash functions are implemented in many applications, including Message Authentication Codes (NIST, 2008), pseudo random number generators (Dimitrova and Markovski, 2004) and key derivation functions (Preneel, 1999). This arguably suggests the need for continuous development of hash functions.

The need for a secure cryptographic hash algorithm was further motivated by the weaknesses found in the widely used MD4, MD5, SHA-0 and SHA-1 hash algorithms (Dobbertin, 1998; Wang, Lai, Feng, Chen, and Yu, 2005; Wang, Yu, and Wang, 2009). In 2007, United States National Institute of Standards and Technology (NIST) called for a public competition to develop a publicly disclosed hash algorithm that is capable of protecting sensitive information for decades (NIST, 2007). One of the advantages of publicly disclosing the algorithms is to overcome the problem of obscurity through secrecy. Similarly, the cryptographic primitives of hash functions are required to have strong mathematical properties, hence, the primitives can be properly examined by cryptographers and thus the correctness of the algorithm will be analyzed (Naor and Yung, 1989).

Two main important criteria in the selection process of SHA3 hash function are: the possible reductions of the hash function security to the security of its compression function, and the resistance of the compression function against differential attacks (Andreeva *et al.*, 2012). This shows that the security of a hash function is determined by the security of its underlying compression function (F).

Traditionally, the underlying compression function of hash functions are constructed based existing block ciphers (Lai and Massey, 1993; Preneel, Govaerts and Vandewalle, 1994; Lin, Wu and Wu, 2009) due to difficulties in constructing new hash functions from the scratch. However, the key generation for each encryption process results to huge computational cost. Alternatively, in order to reduce the computational cost, few are keys generated from a strong key scheduling algorithm such that the hash function will only use these keys. However, according to Mennick and Preneel (Mennink and Preneel, 2012), this procedure also has computational cost. Hence, the solution is to consider a limited instantiations of the block cipher and not its entirety.

This new approach of constructing a compression function was first studied by Black, Cochran and Shrimpton (Black *et al.*, 2005). Their study showed that the simplest case of a compression function is *2n-to-n*-bit. The study also showed that one n-bit permutation (π) cannot be collision resistant. The result of their study has been generalized by Rogaway and Steinberger (2008), Stam (2008) and Steinberger (2010). Mennick and Preneel (2012) demonstrated the possibility of constructing a collision and pre-image resistant *2n-to-n*-bit compression function solely based on three distinct permutations (multi-permutation) and *XOR* operation. The security of the approach was analyzed based on defining equivalent class on the set of compression functions (F). Their work also suggests that compression functions based on single-permutation cannot be secure (collision and pre-image resistant).

Our work is aimed at constructing *2n-to-n*-bit compression function based on permutations (*single* and

multi-permutations) and non-associative quasigroups. Given that quasigroups are non-abelian structures, they have been shown to be more efficient than abelian structures such as *XOR* and addition operators in error detection codes (Schulz, 1991). The two proposed constructions (PQ and 3PQ) are designed to utilize the efficiency of permutations, quasigroups and string transformations in resisting attacks. However, this paper does not discuss a complete construction of a hash function but rather focuses on compression functions. The paper is organized as follows. Section II of this paper describes the basic concepts of permutation, quasigroup and quasigroup string transformation which are very relevant in the development of our scheme. Design of the algorithm is given in Section III. Experiments were carried out in Section IV inorder to analyse the strength of our proposed construction. Section V gives the concluding remarks and scope for future work in this direction.

2. PERMUTATION AND QUASIGROUPS

This section discusses mathematical properties that are relevant in the design of our proposed scheme.

Permutation

Permutations have various applications especially in error detection system. Using permutations, particularly non-commutative permutations, to complicate check digit equations increases the efficiency of the system in error detection (Belyavskaya, Izbash, and Shcherbacov, 2003; Schulz, 1991). This subsection will describe some basic properties of permutation.

Definition *1* (Ecker and Poch, 1986). A Permutation is a bijection π of a finite group $(Q,+)$ onto itself $\pi: Q \rightarrow Q$. The permutation a group $(Q,+)$ is said to be *non-commutative* iff for all $x, y \in Q$ such that;

$$x + \pi(y) \neq y + \pi(x) \rightarrow x \neq \pi(x) \quad (1)$$

Proposition 1: π is a non-commutative permutation on a finite group Q and satisfy for all $u, v \in Q$ such that;

$$\pi(u) + v = \pi(v) + u \rightarrow u = v \quad (2)$$

Proof: The proposition is converse of definition. Thus, they are equivalent.

Generating permutations

The following subsection describes the basic methodology for generating quasigroups. Other methods can be found in (Ecker and Poch, 1986).

Example 1. Given a set {1, 2, 3, 4, 5, 6, 7, 8, 9} we choose a weight w_i such that {0, $w_i \times 1$, $w_i \times 2$, $w_i \times 3$, $w_i \times 4$, $w_i \times 5$, $w_i \times 6$, $w_i \times 7$, $w_i \times 8$, $w_i \times 9$} mod 10 gives a permutation δ_i of the decimal digits. It is required that w_i is relatively prime to 10.

$$\text{GCD}(w_i, 10) = 1 \quad (3)$$

By choosing w_i=3, the following permutation is generated;

x	0	1	2	3	4	5	6	7	8	9
$\pi(x)$	0	3	6	9	2	5	8	1	4	7

It can also be observed that the permutation fixes 0 and 5. Hence, it is not be completely *non-commutative*.

Quasigroups (Latin squares)

Quasigroups (Latin squares) have many applications especially in the area of cryptography for the development of block ciphers (Marnas, Angelis and Bleris, 2003), in securing file systems and in coding theory for designing error detection and correction codes (Belyavskaya *et al.*, 2003; Schulz, 1991). The application of quasigroups in developing ciphers has been proven to be more efficient than ciphers based on groups and fields (Scielny, 2002). Similarly in coding theory, check digit systems based on quasigroups are more efficient in than modulo m check digit systems (Schulz, 1991). In this subsection we will give some definitions from the theory of quasigroups.

Definition *2* (Scielny, 2002). A quasigroup $(Q,*)$ is an algebraic structure containing a set of elements Q together with a binary operation "$*$". For all $a, b \in Q$, there exist unique solution x, y in Q, such that

$$x * a = b \quad (4)$$
$$a * y = b$$

Proposition 2: (Q, $*$) is a non-associative quasigroup on a finite integer set $\mathbf{Z}_n$ which satisfy for all x, $y \in Q$, such that;

$$(x * y) * a \neq (x * a) * y \rightarrow a \neq y$$

Proof: Given the integers $x, y, a \in Q$ under a binary operation "$*$". If $(x * y) * a = (x * a) * y$ under a non-associative binary operation, this implies that $a = y$. Thus the quasigroup (Q, $*$) is non-associative.

The multiplication table of a Latin square of order n is an $n \times n$ square matrix which forms a quasigroup, such that each element occurs only once in a row and column. The binary operation "$*$" is neither commutative nor associative. A Latin square of order n can be generated using the formula in Equation 7 (Scielny, 2002).

$$LS(n!) = n!\,(n-1)!\,Tn \quad (5)$$

where $T(n)$ denotes the number of reduced Latin squares or order n. A Latin square is said to be a reduced Latin square if the elements of both its first row and first column are in natural order. The next subsection will describe some methods for generating quasigroups.

Generating quasigroups

There are various methods of constructing quasigroups. However, this paper will discuss two methods for constructing quasigroups. The first method was adopted in (Marnas *et al.*, 2003) and (Gligoroski, 2004) and the second method was discussed in (Ecker and Poch, 1986). Both methods are simple, fast and easy to implement and can be used to generate huge quasigroups with more than 1024 bits. We refer interested readers to Meyer (2006) for other methods.

A. Quasigroups based on Permutation

This method was adopted by Gligoroski (2004) and Marnas *et al.*, (2003) for developing cryptographic primitive for cipher.

Definition *3* (Gligoroski, 2004; Marnas *et al.*, 2003)**.** Given that $Z_P = (1, 2,\ldots j\ldots, n)$ is a set of positive finite integers such that the permutation $P = (a_{11}, a_{12}, \ldots j\ldots., n)$ is a permutation of Z_P , which defines the first row of the quasigroup (Latin square). The method of constructing the quasigroup is defined by Equation 6.

$$i * j = i \times a_{1j} \; mod \; P \tag{6}$$

where i is row, j is column, a_{1j} is first row and P is prime such that $P = n + 1$.

Example 2. ((Marnas *et al*., 2003), pp. 187). A randomly generated permutation P is given as $P(\mathbf{Z}_P) = \{3, 5, 2, 1, 6, 4\}$ such that the first row a_{1j} is $P(\mathbf{Z}_P)$.

Table-1. Generating quasigroup of order 6.

*	**1**	**2**	**3**	**4**	**5**	**6**
1	**3**	**5**	**2**	**1**	**6**	**4**
2	2*3 mod7	2*5 mod7	2*2 mod7	2*1 mod7	2*6 mod7	2*4 mod7
3	3*3 mod7	3*5 mod7	3*2 mod7	3*1 mod7	3*6 mod7	3*4 mod7
4	4*3 mod7	4*5 mod7	4*2 mod7	4*1 mod7	4*6 mod7	4*4 mod7
5	5*3 mod7	5*5 mod7	5*2 mod7	5*1 mod7	5*6 mod7	5*4 mod7
6	6*3 mod7	6*5 mod7	6*2 mod7	6*1 mod7	6*6 mod7	6*4 mod7

Table-1 is simplified to the quasigroup in Table-2.

Table-2. Quasigroup of order 6.

*	**1**	**2**	**3**	**4**	**5**	**6**
1	3	5	2	1	6	4
2	6	3	4	2	5	1
3	2	1	6	3	4	5
4	5	6	1	4	3	2
5	1	4	3	5	2	6
6	4	2	5	6	1	3

B. Quasigroups based linear mapping

This method uses a permutation like approach where three positive intergers h, k, l are selected, such that h and k are relatively prime to n. This defines a quasigroup as given in Definition 4 (Ecker and Poch, 1986).

Definition *4.* Given that h, k, l are fixed integers where h and k are relatively prime to n defines a quasigroup on the set $(Z_n) = \{0, 1,\ldots.., n\text{-}1\}$. A quasigroup is defined by the operation

$$a * b = ((h \times a) + (k \times b) + l) \bmod n \tag{7}$$

Example 3. Let h, k, l, n be 2, 3, 5 and 7 respectively such that the GCD (2, 7) = GCD (3, 7) = GCD (5, 7) = 1. A quasigroup is generated as Table-3.

Table-3. Quasigroup of order 6.

*	**0**	**1**	**2**	**3**	**4**	**5**	**6**
0	5	1	4	0	3	6	2
1	0	3	6	2	5	1	4
2	2	5	1	4	0	3	6
3	4	0	3	6	2	5	1
4	6	2	5	1	4	0	3
5	1	4	0	3	6	2	5
6	3	6	2	5	1	4	0

Quasigroup string transformations

There are various applications of qausigroup string transformations in cryptology and coding theory. The work of (Krapez, 2010; Marnas *et al.,* 2003; Smile, 1997) are one of the successful application of quasigroup string transformations in developing cryptographic primitives for stream ciphers. Similarly, quasigroup string transformations was applied in the work of Mileva (Mileva and Markovski, 2010) for the construction of compression function for NaSHA SHA3 hash function. Other applications of quasigroup string transformation as primitives for pseudorandom generators can be found in (Bakeva, 2011; Dimitrova and Markovski, 2004).

Definition *5* ((Mileva and Markovski, 2010), pp. 369). Given a quasigroup $(Q,*)$ with elements $a_1, a_2,\ldots, a_n$

where $a_i \in Q$, $i = 1, 2,\ldots, n$. Let l be a constant leader where $l \in Q$. A quasigroup transformation $E = E_*, l : Q^+ \rightarrow Q^+$ is given as;

$b_1 = l * a_1$
$b_2 = b_1 * a_2$
$b_3 = b_2 * a_3$
…
$b_n = b_{n-1} * a_n$

We give the generalization in Equation 8.

$$E_*, l(a_1 \ldots a_n) = (b_1 \ldots b_n) = \begin{cases} b_1 = l * a_1 \\ b_i = b_{i-1} * a_i \end{cases} \quad i = 2 \ldots n \tag{8}$$

Graphical repesentation of transformation is given in Figure-1.

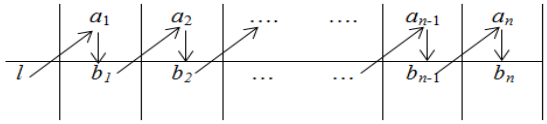


Figure-1. Graphical representation transformation *E*.

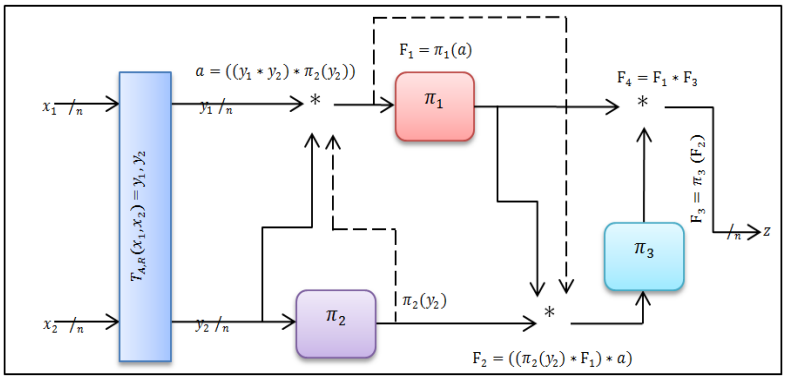


Figure-2. *2n-to-n*-bits compression function based on multi-permutation (3PQ).

The next section will describe the construction of our proposed scheme using permutations and quasigroups and quasigroup string transformations.

3. COMPRESSION FUNCTION BASED ON PERMUTATION AND QUASIGROUPS

This section discusses the method of constructing compression function based on permutation and quasigroups.

Preliminary

The function takes in two *n*-bits inputs ($x_1$and x_2), processes them using mathematical transformation and generates an *n*-bits output (z). The function is defines as follow;

$$F: \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n \tag{9}$$

where F is the compression function

$\{0,1\}^n$ denotes input x_1
$\{0,1\}^n$ denotes input x_2
$\{0,1\}^n$ denotes output z

This paper proposes two classes of compression functions; *single*-permutation (PQ) and *three*-permutations (*multi*-permutations) (3PQ) compression function. In the single-permutation setting, all the permutations are the equal (π_1=π_2=π_3), however the permutations are not equal in the multipermutation setting ($\pi_1 \neq \pi_2 \neq \pi_3$). The permutations and quasigroup adopted in implementating the compression function are both of order 16. The quasigroup transformation we adopted is similar to the transformation in NaSHA (Mileva and Markovski, 2010). However, the major difference is that orthorgonal quasigroup was used in NaSHA, while single quasigroups (isotopes of quasigroups) are used in our function. Secondly, the leader (l) used in transformation for NaSHA are constant for all operations while the leader (l) used in our transformation is dependent on the input. We have demonstrated that changing the leaders will increase the efficiency of the function. The method for generating the leader (l) will be discussed in the subsequent subsections.

Description of compression function

a. Input: The function takes and compute two inputs *n*-bit $x_1$and x_2.

b. Tranformation $T_{A,R}$ (x_1, x_2): Input x_1 and x_2 are processed in a *forward* and *reverse* quasigroup transformation function to produce y_1 and y_2 ($T_{A,R}(x_1, x_2) = y_1, y_2$).

c. Intermediate result a: Result a is generated by multiplying y_1, y_2 and *permutation two* of y_2, $((y_1 * y_2) * \pi_2 (y_2))$.

d. Subfunction F_1: F_1 is generated by computing *permutation one* of a, ($F_1 = \pi_1 (a)$).

e. Subfunction F_2: F_2 is generated by multiplying *permutation two* of y_2, F_1 and a that order respectively, $((\pi_2 (y_2) * F_1) * a)$.

f. Subfunction F_3: F_3 generated by computing *permutation three* of F_2, ($\pi_3 (F_2)$).

g. Output: Hash z of n-bit is generated by multiplying subfunction $F_1 * F_3$.

All multiplications are done using a defined quasigroup Table.

```
1  Algorithm H(x1, x2)
2      y1, y1 = TA,R(x1, x2)
3      y2 = TA,R(x2)
4      F1 ← π1 ((y1 * y2) * π2(y2))
5      F2 ← (π2 (y2) * F1) * ((y1 * y2) * π2(y2))
6      F3 ← π3 (F2)
7      F4 ← F1 * F3
8  return z ← F4
```

Figure-3. Description of multi-permutation compression function (3PQ).

Two types of transformations were applied in our scheme; forward (T_A) and reverse (T_R) transformation. The forward transformation is similar to the transformation described in Definition 5. We give a formal definition of forward and reverse transformation in Definition 6.

Definition 6. Given a quasigroup $(Q,*)$ with elements $x_1 = u_{11}, u_{12},\ldots, u_{1n}$ and $x_2 = u_{21}, u_{22},\ldots, u_{2n}$, where x_1 and $x_2 \in Q$, $i = 1, 2,\ldots, n$. Let l be a leader where $l \in Q$. A quasigroup forward transformation $T_A(x_1, x_2)$ is defined as;

$v_{11} = l * \pi(u_{11})$
$v_{12} = v_{11} * \pi(u_{12})$
$v_{13} = v_{12} * \pi(u_{13})$
…
$v_{n-1} = v_{n-2} * \pi(u_{n-1})$
$v_n = v_{n-1} * \pi(u_n)$

We give the generalization in Equation 10.

$$T_A(x_1,x_2) = T_A\left(\pi(u_{11}\ldots u_{1n}, u_{21}\ldots u_{2n})\right)$$

$$= (v_{11}\ldots v_{1n}, v_{21}\ldots v_{2n})$$

$$= \begin{cases} v_{11} = l * \pi(u_{11}), & v_{21} = v_{1n} * \pi(u_{21}) \\ v_{1i} = v_{1i-1} * \pi(u_{1i}), & v_{2i} = v_{2i-1} * \pi(u_{2i}) \end{cases} \quad i = 2\ldots n \tag{10}$$

Graphical repesentation of forward transformation $T_A(x_1, x_2)$ is given in Figure-4.

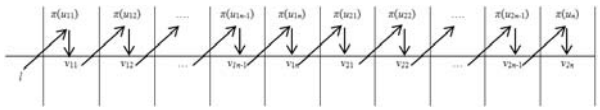


Figure-4. Representation of forward transformation (T_A).

The reverse transformation $T_R(x_1, x_2)$ is given by

$y_{2n} = \pi(v_{2n}) * l$
$y_{2n-1} = \pi(v_{2n-1}) * y_{2n}$
…
$y_{13} = \pi(v_{13}) * y_{14}$
$y_{12} = \pi(v_{12}) * y_{13}$
$y_{11} = \pi(v_{11}) * y_{12}$

We give the generalization in Equation 11.

$$T_R(T_A(x_1,x_2)) = T_R(\pi(v_{11}\ldots v_{1n}, v_{21}\ldots v_{2n}))$$
$$= (y_{11}\ldots y_{1n}, y_{21}\ldots y_{2n})$$
$$= \begin{cases} y_{2n} = \pi(v_{2n}) * l, & y_{1n} = \pi(v_{1n}) * y_{21} \\ y_{2n-i} = \pi(v_{2n-i}) * y_{2n-(i-1)}, & y_{1n-i} = \pi(v_{1n-i}) * y_{1n-(i-1)} \end{cases} \quad i = 1\ldots n \tag{11}$$

Graphical repesentation of reverse transformation

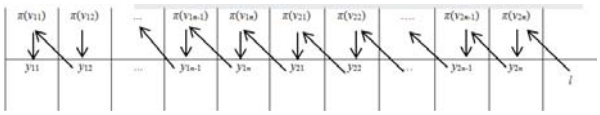


Figure-5. Representation of reverse transformation (T_R) $T_R(T_A(x_1, x_2))$ is given in Figure-5.

where $l = (((((((\pi_1(u_{11}) * \pi_1(u_{12})) * \pi_1(u_{1n-1})) *\ldots* \pi_1(u_{1n})) * \pi_1(u_{21})) * \pi_1(u_{12})) * \pi_1(u_{1n-1})) *\ldots* \pi_1(u_{1n}))$.

The output of the reverse transformation will serve as input to the main compression function. Permutations π_1 and π_2 where implemented in forward and reverse transformation respectively. It can also be observed that the leader l is generated by multipliying all the elements. A slight change in the element will result to a different leader. Note that all multiplications are done under quasigroup operation.

Given the description of transformation provided in Definition 6, the following results are being obtained.

Lemma 1: Changing a bit in the input at any position will significantly affect the resulting output of the transformation.

Proof: Let $a_1, a_2,\ldots, a_i,\ldots, a_n$ be the input of a forward transformation under a non-associative binary operation "$*$". The leader l is determined by multiplying all the input elements as in;
$l = ((a_1 * a_2) * a_i) * a_n$.
The following result is obtained when we compute the input in a forward transformation.

$$T_{A,l}(a_1, a_2,\ldots, a_n) = b_1, b_2,\ldots, b_n \tag{α}$$

By changing a_i with $\tilde{a}_i$, we obtain a new leader

$$l' = ((a_1 * a_2) * \tilde{a}_i) * a_n.$$

Using the new leader l', a different output is be obtained.

$$T_{A,l'}(a_1, a_2,\ldots, a_n) = b_1', b_2',\ldots, b_n' \tag{β}$$

Given that $l \neq l'$, we can conclude that $\alpha \neq \beta$.

Corollary: The proof provided in Lemma 1 shows that the forward and reverse tranfromation is an error detection code.

Example 4 describes the mechanism of the multi-permutation based compression function.

Example 4. A quasigroup and three permutations of order 16 was used in developing the scheme. We adopted a highly non-associative quasigroup from (Meyer, 2006) in Figure-5.4 and three permutations (π_1, π_2, π_3). The following example shows illustrates the working mechanism of our scheme:

x = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f}
$\pi_1(x)$ = {c, 6, 3, e, 2, d, 5, 9, 8, b, f, 1, 7, 4, a, 0}
$\pi_2(x)$ = {e, 3, 4, f, d, 0, b, 5, 9, 6, 1, 3, c, 7, a, 8}
$\pi_3(x)$ = {4, 6, 2, 9, f, d, 7, 8, 5, b, 3, e, a, 1, 0, c}

Given the following inputs
x_1 = 00 00 00 00
x_2 = 00 00 00 00
where x_1 and x_2 is 16 bits each in hexadecimal
a. After forward transformation
v_1 = f9 6b fe 96
v_2 = bf e9 6b fe.
b. After reverse transformation
y_1 = e6 f4 6c b2
y_2 = 18 8d 07 1a.
a. Intermediate result F_1: bf 7f 5f 66
b. Intermediate result F_2: cf e4 01 c8
c. Intermediate result F_3: 1c 0f 46 15
d. Output z: 18 e7 63 51

The proposed scheme is analysed based the following properties of a hash function:

- One way property: Given a input x, it should be easy to compute the hash value h(x).
- Avalanche effect: A slight change in the input should cause significant changes in the output.
- Quasigroup and Permutation attack (Brute force attack): Using all possible combinations quasigroups and permutations to find two dististict input (x_1, x_2) and (x_1', x_2') with the same output (z).

Experimentation and analysis of the result will discuss in the next section.

4. RESULTS AND ANALYSIS

The scheme have been implemented in C++ and run on Intel® Core™ i7-3770 CPU 3.40 GHz, 4 GB RAM and 64-bit Windows 7 operating system. The proposed scheme is composed of five subfunctions; $T_{A,R}(x_1, x_2)$, F_1, F_2, F_3 and F_4. Subdivinding the scheme into subsubfunction provides an easy method to assess the security strenght of the entire compression funtion.

One way function

The proposed scheme has a good function. The result obtained from Example 4 demonstrates the one way property of the multi-permutation compression function. Given the output (z), it is difficult to predict the inputs (x_1, x_2).

Avalanche effect

The aim of this test is to demonstrate how our scheme responds to changes in original inputs. Our compression function was compared with the *2n-to-n*-bit function proposed by Mennink (Mennink & Preneel, 2012) using 16 bits hexadecimals as inputs.

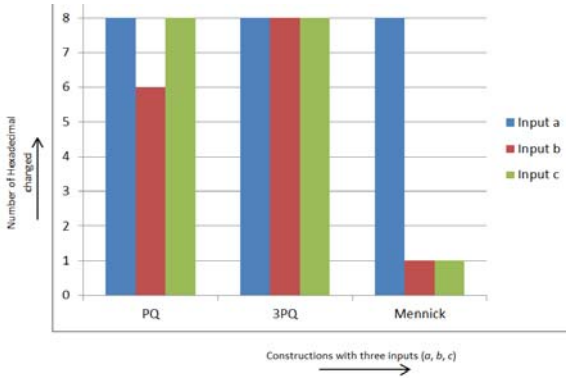
In the experiment, a represent the original input (x_1, x_2), where values are zeros "0", while b and c represent modified inputs. Also, we define another input d where all values are "f", while other inputs f and g are modified. A slight change in the modified input should produce significant changes in the output. This is shown in Figures 6a and 6b.

Storage requirement and execution time comparison

We compared the space required to store each compression function and also its execution time. The result was obtained from the C++ compiler as illustrated in Table-4.

Table-4. Comparing storage and execution time.

		Schemes	
Parameters	PQ	3PQ	Mennick
Exec. time (s)	0.004	0.004	0.099
File Size (kb)	4.96	4.96	1.57



Input a : x_1: 00 00 00 00, x_2: 00 00 00 00
Input b : x_1: 00 00 00 0f, x_2: 00 00 00 00
Input c : x_1: 00 00 00 00, x_2: 00 00 00 0f

Figure-6a. Comparison of three compression functions.

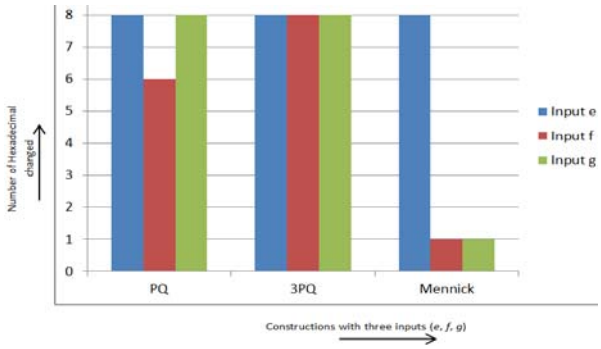
It can be clearly observed that the execution time of our proposed schemes (PQ and 3PQ) is relatively faster than the other two schemes. However, interms of storage requirement, our proposed scheme requires a relatively larger storage than the other two schemes. This is due to the quasigroup adopted in the scheme.

Quasigroup and permutation attack

The core strength of our scheme lies in the quasigroups and permutations. A collision can be found in two equivalent subfunctions F_1 and F_2 if

$F_1: z = \pi_1((y_1 * y_2) * \pi_2(y_2))$
$F_2: z' = \alpha_1((y_1 \bullet y_2) \bullet \alpha_2(y_2))$

where permutations $\pi_1 \neq \alpha_1$, $\pi_2 \neq \alpha_2$, quasigroup $(Q, *) \neq (H, \bullet)$ and $z = z'$. Hence, in order to find a collision using this attack, the attacker has to compute all possible combinations of quasigroups and permutations. Given that there exist more than 40 million non-associative quasigroups (Damm, 2003) and more than 3 million possible permutations of order 10, it is difficult to find a collision using this attack where $F_1 = F_2$.



Input *a*: x_1: 00 00 00 00, x_2: 00 00 00 00
Input *b*: x_1: 00 00 00 0f, x_2: 00 00 00 00
Input *c*: x_1: 00 00 00 00, x_2: 00 00 00 0f

Figure-6b. Comparison of three compression functions.

5. CONCLUSIONS

Constructing primitives for cryptographic hash functions is a challenging exercise due to the difficulty in proving the security and correctness of the underlying primitive (compression function). This necessitates the use of existing block ciphers as the underlying primitive in constructing hash functions. However, using block ciphers as primitives incures huge computational cost which limits the effectiveness and preformance of the hash function. An alternative method was to use few strong keys rather than generating new keys for each encryption. However, this method also incures computational cost. Other alternative was to use few components of the cipher such as permutations, S-Boxes and Boolean (logical) operators, and not its entirety. This method has shown to be promising as indicated by the literature. This paper discusses another perspective to contructing compression function based on quasigroups, permutations and quasigroup string transformations.

The proposed compression functions were evaluated based on three criteria; 1) one way function 2) avalance property 3) quasigroup and permutation attack 4) storage and execution time. The results demonstrates that the compression functions (single and multi-permutation compression function) can be a good candidate for an underlying primitive for a hash function, message authentication code (MAC) or pseudorandom number generator.

However, what remains an open question in this research is whether the compression functions, particularly the single-permutation compression function (PQ), are collision, pre-image and 2nd pre-image resistant and birthday attacks. Given that the compression functions have effective avalance property and resistant to quasigroup and permutation attack does not guarantee their resistance against other types of attacks. Proving the resistance of the compression functions requires proving the correctness of the subfunctions (forward and reverse transformation ($T_{A,R}(x_1, x_2)$), F_1, F_2, F_3 and F_4).

In the future work of this research, we intend to implement larger quasigroup and permutations of order 256 or more. We also intend to perform pre-image attack among other experiments to further evaluate the strength of the scheme from different perspectives.

ACKNOWLEDGEMENT

This research was supported by the Office for Research, Innovation, Commercialization and Consultancy (ORICC), Universiti Tun Hussein Onn Malaysia (UTHM) under Project Vot No. U194.

REFERENCES

Andreeva, E., Bogdanov, A., Mennink, B., Preneel, B., and Rechberger, C. 2012. On security arguments of the second round SHA-3 candidates. International Journal of Information Security, 11(2), 103-120.

Bakeva, V. 2011. Parastrophic quasigroup string processing, (8th Conference on Informatics and Information Technology with International Participation (CIIT 20011)), 19-21.

Belyavskaya, G. B., Izbash, V. I., and Shcherbacov, V. a. 2003. Check character systems over quasigroups and loops. Quasigroups and Related Systems, 10(10), 1-28.

Black, J., Black, J., Cochran, M., Cochran, M., Shrimpton, T., and Shrimpton, T. 2005. On the Impossibility of Highly-Efficient Blockcipher-Based Hash Functions. In Lecture Notes in Computer Science (Vol. 1, pp. 526–541). Springer Berlin Heidelberg.

Damm, M. 2003. On the existence of totally anti-symmetric quasigroups of order 4k + 2. In Computing (Vienna/New York) (Vol. 70, pp. 349-357). Springer-Verlag.

Dimitrova, V., and Markovski, J. 2004. On Quasigroup Pseudo Random Sequence Generators. In Proc. of the 1-st Balkan Conference in Informatics, Y. Manolopoulos and P. Spirakis eds (pp. 21-23).

Dobbertin, H. 1998. Cryptanalysis of MD4. Journal of Cryptology, 11, 253-271.

Ecker, A. and Poch, G. 1986. Check character systems. Computing, 37(4), 277-301.

Gligoroski, D. 2004. Stream cipher based on quasigroup string transformations in Z. arXiv Preprint cs/0403043.

Kanso, a., Yahyaoui, H., and Almulla, M. 2012. Keyed hash function based on a chaotic map. Information Sciences, 186(1), 249-264.

Krapez, A. 2010. An application of quasigroups in cryptology. Math. Maced 8, 47-52.

Lai, X., and Massey, J. L. 1993. Hash Functions Based on Block Ciphers. In InAdvances in Cryptology-EUROCRYPT'92 (pp. 55-70). Springer Berlin Heidelberg.

Lai, X., and Massey, J. L. 2009. Hash Functions Based on Block Ciphers. Journal of Software.

Li, Y., Xiao, D., and Deng, S. 2012. Keyed hash function based on a dynamic lookup table of functions. Information Sciences, 214, 56-75.

Marnas, S. I., Angelis, L., and Bleris, G. L. 2003. All-Or-Nothing Transforms Using Quasigroups All-Or-Nothing Transforms. In 1st Balkan Conference in Informatics (pp. 183-191).

Mennink, B., and Preneel, B. 2012. Hash functions based on three permutations: A generic security analysis. In Lecture Notes in Computer Science (Vol. 7417 LNCS, pp. 330-347). Springer Berlin Heidelberg.

Meyer, K. A. 2006. A new message authentication code based on the non-associativity of quasigroups. In Retrospective Theses and Dissertations. Iowa State University.

Mileva, A., and Markovski, S. 2010. Quasigroup String Transformations and Hash Function Design. In ICT Innovations 2009 (pp. 367-376). Springer Berlin Heidelberg.

Naor, M., and Yung, M. 1989. Universal one-way hash functions and their cryptographic applications. In Proceedings of the twenty-first annual ACM symposium on Theory of computing (pp. 33-43). ACM.

NIST. 2007. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family, 1-22. Retrieved from http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf

NIST, F. 2008. The Keyed-Hash Message Authentication Code. Federal Information Processing Standard Publication, 198, 1-20.

Preneel, B. 1999. The State of Cryptographic Hash Functions. In Lectures on Data Security (pp. 158-182). Springer Berlin Heidelberg.

Preneel, B., Govaerts, R., and Vandewalle, J. 1994. Hash functions based on block ciphers: A synthetic approach. In Advances in Cryptology-CRYPTO' … (pp. 368-378). Springer Berlin Heidelberg. Retrieved from

Rogaway, P. and Steinberger, J. 2008. Constructing cryptographic hash functions from fixed-key blockciphers. In Lecture Notes in Computer Science (Vol. 5157 LNCS, pp. 433-450). Springer Berlin Heidelberg.

Schulz, R.-H. 1991. A note on check character systems using Latin squares. Discrete Mathematics, 97, 371–375.

Scielny, K. O. 2002. Generating quasigroups for cryptographic applications. International Journal of Applied Mathematics and Computer Science, 12(4), 559-570.

Smile, M. 1997. Using quasigroups for one-one secure encoding. In Proc. VIII Conf. Logic and Computer Science "LIRA (Vol. 97, pp. 157-162).

Stam, M. 2008. Beyond uniformity: Better security/efficiency tradeoffs for compression functions. In Lecture Notes in Computer Science (Vol. 5157 LNCS, pp. 397-412). Springer Berlin Heidelberg.

Steinberger, J. 2010. Stam's collision resistance conjecture. In Advances in Cryptology–EUROCRYPT 2010 (pp. 597–615). Springer Berlin Heidelberg.

Wang, X., Lai, X., Feng, D., Chen, H., & Yu, X. 2005. Cryptanalysis of the Hash Functions MD4 and RIPEMD. Advances in Cryptology-EUROCRYPT 2005, 551-551.

Wang, X., Yu, H., and Wang, W. 2009. Cryptanalysis on hmac/nmac-md5 and md5-mac. In In Advances in cryptology-EUROCRYPT (pp. 1-14). Springer Berlin Heidelberg.