Hash Functions Based on Large Quasigroups

Václav Snášel[1], Ajith Abraham[2], Jiří Dvorský[1], Pavel Krömer[1], and Jan Platoš[1]

[1] Department of Computer Science, FEECS, VŠB – Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{vaclav.snasel}@vsb.cz
[2] Department of Computer Science Norwegian University of Science and Technology, Trondheim, Norway
ajith.abraham@ieee.org

Abstract. In this article we discuss a simple hash function based upon properties of a well-known combinatorial design called quasigroups. The quasigroups are equivalent to the more familiar Latin squares and one of their most important properties is that all possible element of certain quasigroup occurs with equal probability. Actual implementations are based on look-up table implementation of the quasigroup, which is unusable for large quasigroups. In contrast, presneted hash function can be easily implemented. It allows us to compute hash function without storing large amount of data (look-up table). The hash function computation is illustrated by experiments summarized in the last section of this paper.

1 Introduction

The need for random and pseudorandom sequences arises in many applications, e.g. in modelling, simulations, and of course in cryptography. Pseudorandom sequences are the core of stream ciphers. They are popular due to their high encryption/decryption speed. Their simple and cheap hardware design is often preferred in real-world applications. The design goal in stream ciphers is to efficiently produce pseudorandom sequences - keystreams (i.e. sequences that possess properties common to truly random sequences and in some sense are "indistinguishable" from these sequences).

Hash functions map a large collection of messages into a small set of message digests and can be used for error detection, by appending the digest to the message during the transmission (the appended digest bits are also called parity bits). The error will be detected if the digest of the received message, in the receiving end, is not equal to the received message digest. This application of hash functions is only for random errors, since an active spoofer may intercept the transmitted message, modify it as he wishes, and resend it appended with the digest recalculated for the modified message.

With the advent of public key cryptography and digital signature schemes, cryptographic hash functions gained much more prominence. Using hash functions, it is possible to produce a fixed length digital signature that depends on the whole message and ensures authenticity of the message. To produce digital signature for a message M, the digest of M, given by $H(M)$, is calculated and then encrypted with the secret key of the sender. Encryption may be either by using a public key or a private key algorithm. Encryption of the digest prevents active intruders from modifying the message

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 521–529, 2009.

and recalculating its check sum accordingly. It effectively divides the universe into two groups: outsiders who do not have access to the key of the encryption algorithm and hence cannot effectively produce a valid checksum, and insiders who do have access to the key and hence can produce valid checksums. We note that in a public key algorithm, the group of insiders consists of only one member (the owner of the private key) and hence the encrypted hash value uniquely identifies the signer. In the case of symmetric key algorithms, both the transmitter and the receiver have access to the secret key and can produce a valid encrypted hash for an arbitrary message and therefore, unique identification based on the encrypted hash is not possible. However, an outsider cannot alter the message or the digest.

In the study of hash functions, Information Theory and Complexity Theory are two major approaches. The methods based on information theory provide unconditional security — an enemy cannot attack such systems even if he/she has unlimited power. This approach is generally impractical.

In the second approach, some assumptions are made based on the computing power of the enemy or the weaknesses of the existing systems and algorithms, and therefore, the security cannot be proven but estimated by the analysis of the best known attacking algorithms and considering the improvements of the hardware and softwares. In other words, hash functions based on complexity theory are computationally secure. In this paper, we concentrate on the second approach.

1.1 Definitions

Definition 1. *A function $H()$ that maps an arbitrary length message M to a fixed length hash value $H(M)$ is a OneWay Hash Function (OWHF), if it satisfies the following properties:*

1. *The description of $H()$ is publicly known and should not require any secret information for its operation.*
2. *Given M, it is easy to compute $H(M)$.*
3. *Given $H(M)$ in the rang of $H()$, it is hard to find a message M for given $H(M)$, and given M and $H(M)$, it is hard to find a message $M'(\neq M)$ such that $H(M') = H(M)$.*

Definition 2. *A function $H()$ that maps an arbitrary length message M to a fixed length hash value is a Collision Free Hash Function (CFHF), if it satisfies the following properties:*

1. *The description of $H()$ is publicly known and should not require any secret information for its operation.*
2. *Given M, it is easy to compute $H(M)$.*
3. *Given $H(M)$ in the rang of $H()$, it is hard to find a message M for given $H(M)$, and given M and $H(M)$, it is hard to find a message $M'(\neq M)$ such that $H(M') = H(M)$.*
4. *It is hard to find two distinct messages M and M' that hash to the same result ($H(M) = H(M')$).*

2 Construction of Hashing Function Based on Quasigroup

Definition 3. *Let Q be a nonempty set with one binary operation $(*)$. Then Q is said to be a grupoid and is denoted by $(Q, *)$.*

Definition 4. *A grupoid $(Q, *)$ is said to be a quasigroup (i.e. algebra with one binary operation $(*)$ on the set Q) satisfying the law:*

$$(\forall u, v \in Q)(\exists! x, y \in Q)(u * x = v \wedge y * u = v).$$

This implies:

1. $x * y = x * z \vee y * x = z * x \Rightarrow y = z$
2. The equations $a * x = b, y * a = b$ have an unique solutions x, y for each $a, b \in Q$.

However, in general, the operation (*) is neither a commutative nor an associative operation.

Quasigroups are equivalent to the more familiar Latin squares. The multiplication table of a quasigroup of order q is a Latin square of order q, and conversely, as it was indicated in [1,2,8], every Latin square of order q is the multiplication table of a quasigroup of order q.

Definition 5. *Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite alphabet, a $k \times n$ Latin rectangle is a matrix with entries $a_{ij} \in A$, $i = 1, 2, \ldots, k$, $j = 1, 2, \ldots, n$, such that each row and each column consists of different elements of A. If $k = n$ we say a Latin square instead of a Latin rectangle. Latin square is called reduced (or in standard form) if both the first row and the left column are in some standard order, alphabetical order being convenient.*

All reduced Latin squares of order n are enumerated for $n \leq 10$ as it is shown in [3]. Let L_n be the number of Latin squares of order n, and let R_n be the number of reduced Latin squares of order n. It is easy to see that $L_n = n!(n-1)!R_n$. The problem of classification and exact enumeration of quasigroups of order greater than 10 probably still remains unsolved. Thus, there are more then 10^{90} quasigroups of order 16 and if we take an alphabet $A = \{0 \ldots 255\}$ (i.e. data are represented by 8 bits) there are at least $256!255! \ldots 2! > 10^{58000}$ quasigroups.

Multiplication in quasigroups has important property: It is proved that each element occurs exactly q times among the products of two elements of Q, q^2 times among the products of three elements of Q and, generally q^{t-1} among the products of t elements of Q. Since there are q^t possible ordered products of t elements of Q, this shows that each element occurs equally often among these q^t products (see [4]).

Definition 6. *Let $H_Q() : Q \to Q$ be projection defined as*

$$H_Q(q_1 q_2 \ldots q_n) = ((\ldots(a * q_1) * q_2 * \ldots) * q_n$$

Then $H_Q()$ is said to be hash function over quasigroup $(Q, *)$. The element a is a fixed element from Q.

Example 1. Quasigroup of modular subtraction has following table representation:

0	3	2	1
1	0	3	2
2	1	0	3
3	2	1	0

The table above defines quasigroup because it satisfies conditions to be Latin Square. Multiplication in the quasigroup is defined in following manner: $a * b = (a + 4 - b) \bmod 4$. It is obvious that the quasigroup is neither commutative ($1*2 = 3, 2*1 = 1$) nor associative. Value of hash function is $H_2(0013) = (((2 * 0) * 0) * 1) * 3 = 2$.

2.1 Sketch of Proof of Resistance to Attacks

Hash function based on quasigroup is iterative process which computes hash value (digest) for message $X = x_1x_2 \dots x_n$. Suppose that $H_Q(X) = d$. Hash function is preimage resistant when it is "impossible" to compute from given digest source message X. The digest d should be factorized into message $Y = y_1y_2 \dots y_n$. In the first step we can divide digest d into two parts y_1 and α_1, where $d = y_1 * \alpha_1$. In the second step value α_1 needs to be divided into y_2 and α_2 ($\alpha_1 = y_2 * \alpha_2$) and so for each element $y_i, 1 \leq i \leq n$. Because each y_i has a same probability of occurrence among products of Q, $|Q|^n$ possible choices should be checked to obtain message Y.

Definition 7. *Quasigroups Q and R are said to be homotopic, if there are permutations satisfying the law:* $(\forall u, v \in R)(u * v = \pi(\omega(u) * \rho(v)))$.

We can imagine homotopy of quasigroups as permutation of rows and columns of quasigroup's multiplication table.

Example 2. Table of quasigroup, which is homotopic with quasigroup of modular subtraction:

0	3	2	1
2	1	0	3
1	0	3	2
3	2	1	0

The table was created from table of modular subtraction. The second and the third row were exchanged. Permutations π, ρ are identities and $\omega = [0213]$. For example $1 * 0 = \omega(1) * 0 = 2 * 0 = 2$.

This example can be considered as a method how to construct new quasigroups. In following text we will use quasigroups homotopic with quasigroup of modular subtraction. Three random permutations will be generated and table will be used to modify original table. Such quasigroup we call "table quasigroup". Disadvantage of this method is huge space complexity (n^2 elements must be stored).

Homotopy gives us possibility to compute result of multiplication without table. Three functions must be chosen to calculate permutations π, ρ, ω. Then the multiplication is defined as follows: $a * b = \pi((\omega(a) + n - \rho(b)) \bmod n)$.

A sequence of n elements were divided into several parts; these were rotated in various directions and exchanged among themselves. Hereafter presented function P1 compute one of these permutations i.e. $P1(x) = \omega(x)$. Two other permutations are implemented in the same way.

```
const unsigned int cQuasiGroupA2::P1(unsigned int x) const
{
   unsigned int Dimension2 = m_Dimension / 2;
   if (x < Dimension2 * 2)
   {
      if (x & 1)
         x = 2 * ((x / 2 + 1) % Dimension2) + 1;
      else
         x = 2 * ((x / 2 + Dimension2 - 1) % Dimension2);
   }
   return x;
}
```

This enables us to work with large quasigroups. Works that are already known use quasigroups of small order only, or only a small parts of certain quasigroup are there used mainly as a key for Message Authentication Code [3]. These are represented as a look-up table in main memory. Hypothesis mentioned above will be tested in next section.

3 Experimental Results

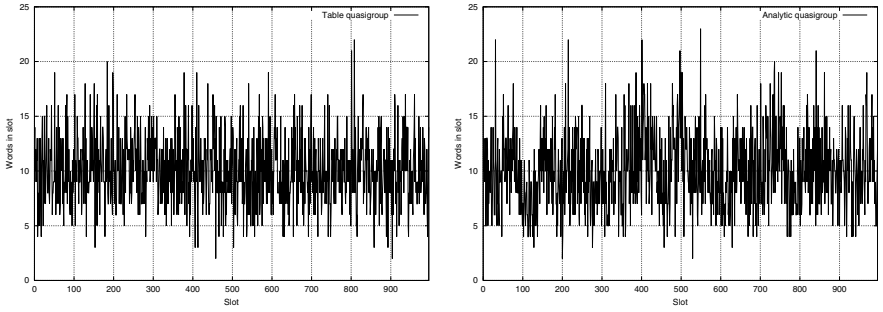
A simple application was created to verify our hypothesis and expectations. Inputs to the application were sets of distinct words, which were extracted from particular text file. The first input was file *bible* from Canterbury Corpus [5] (section Large files) and it was about 4 megabytes long. About 10000 distinct words were extracted from this file. The second was file *latimes* from TREC document corpus (see `http://trec.nist.gov`). The file contained Los Angeles Times volumes 1989 and 1990. And it was about 450 megabytes long. We extracted 200000 distinct words from this text file.

To get statistical data about distribution of words in range of hash values and other properties imaginary hash table have been implemented and values in the table were measured. We observed several parameters:

1. divergences in numbers of words in slots for given size of hash table between our hash function and uniform distribution of words in table,
2. distribution of lengths of slots,
3. how many bits are changed, when one bit in input has been inverted,
4. probability of bits alternation in given position, when one bit in input has been inverted.

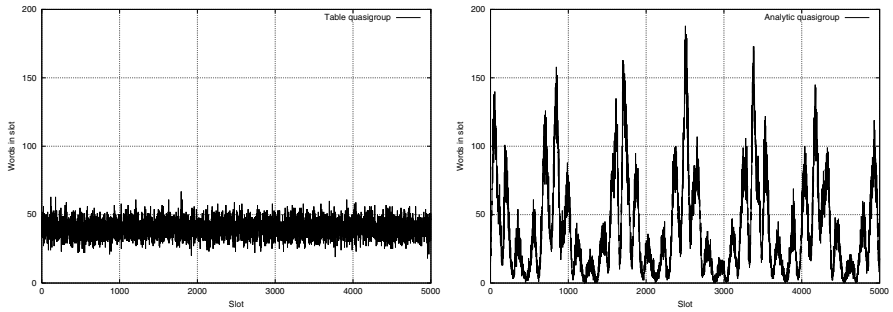
3.1 Distribution of Words in Slots

The distribution of words in slots of hash table is figured in charts 1, 2. It can be seen from charts 1(a) and 1(b) that distribution of words in slots of imaginary hash table is quite uniform, both for table quasigroup and for analytic one. But in chart 2 there are differences between table and analytic quaisgroup. Moreover analytic results have regular shape. This error is caused by constant parameters of functions that compute permutations in analytic quasigroup.



(a) Table quasigroup (b) Analytic quasigroup

Fig. 1. Distribution of words in slots for file *bible*, hash size 997



(a) Table quasigroup (b) Analytic quasigroup

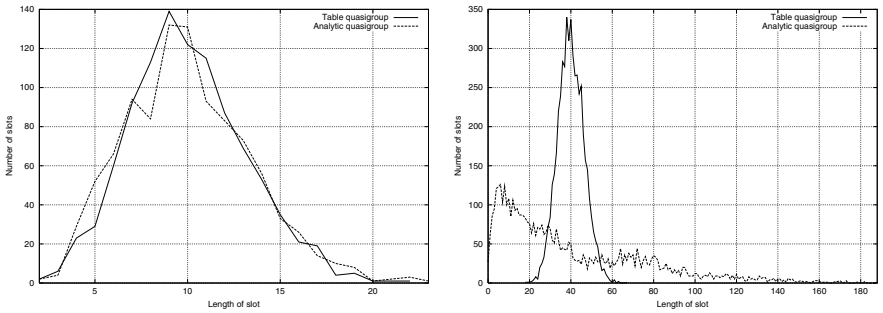
Fig. 2. Distribution of words in slots for file *latimes*, hash size 5003

3.2 Distribution of Lengths of Slots

We can observe good correspondence between distribution of table quasigroup and analytic quasigroup in chart 3(a) for the file *bible*. For file *latimes* there is absolute divergence in chart 3(b).

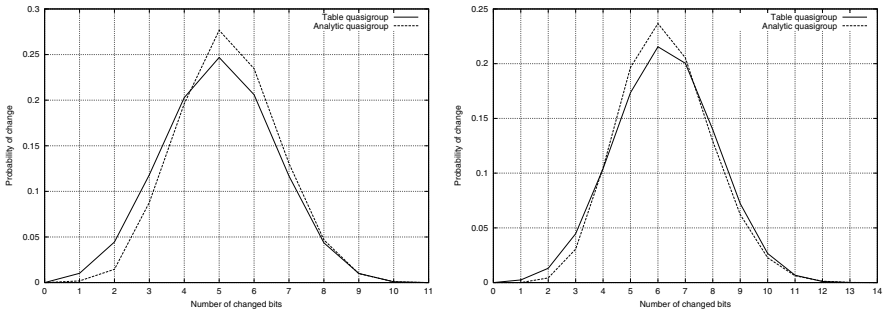
3.3 Probability of Change of Particular Number of Bits

We focus on influence of inputs change on resultant value of hash function. Step by step every bit in each input word was inverted and value of hash function was computed.



(a) *bible*, hash size 997 (b) *latimes*, hash size 5003

Fig. 3. Distribution of slots' length



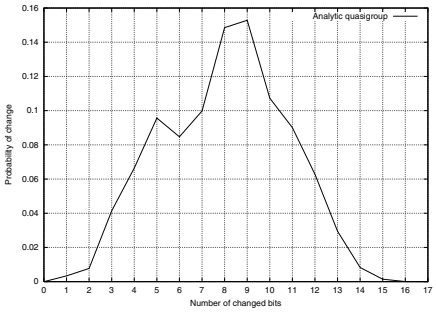
(a) *bible*, hash size 997 (b) *latimes*, hash size 5003

Fig. 4. Probability of change of particular number of bits

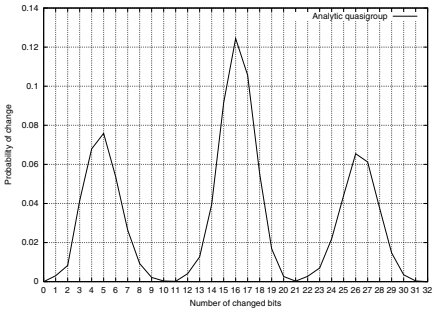
Then Hamming distance between hash values for original word and modified one was measured. It can be seen from chart 4 that both distribution curve has the same shape and they are very close together. It is interesting especially for chart 4(b) with respect of bad characteristic of slots distribution in chart 3(b). Next we perform experiments with analytic quasigroup of order 2^{16} and 2^{32} i.e. for 16 and 32 bit long numbers. Result of the experiment is given in chart 5.

3.4 Probability of Bits Alternation in Given Position

Alternations of bits in specific positions in result of hash function were observed. The experiment runs with the same conditions as previous experiment, but we kept track to positions of changed bits. Only minor errors can be seen (chart 6) between table quasigroup and analytic quasigroup. The changes are uniformly distributed over all bits in resultant hash value.

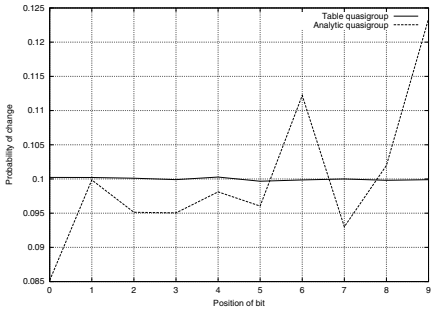


(a) *latimes*, hash size 2^{16}

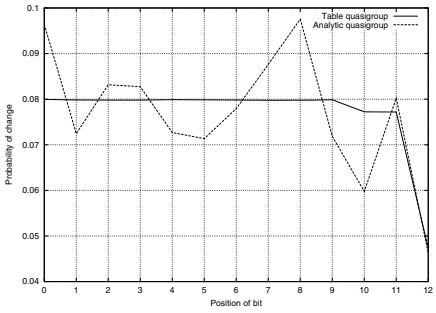


(b) *latimes*, hash size 2^{32}

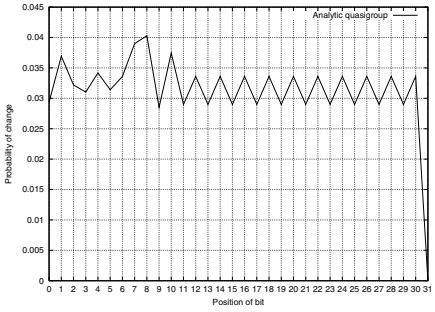
Fig. 5. Probability of change of particular number of bits



(a) *bible*, hash size 997



(b) *latimes*, hash size 5003



(c) *latimes*, hash size 2^{32}

Fig. 6. Probability of particular bit's change

4 Conclusion and Future Works

We presented hash function based on non-associative algebraic structures. This work is continuation of our paper [6]. The presented hash function can be easily implemented. Comparison between look-up table and analytic quasigroup implementation is given.

Analytic quasigroup has some faults in it properties, but there is no need to store large table. For real usage arithmetic of long numbers (i.g. 512 bits) must be adopted. Non-associative structure - neofield - could be incorporated in our future works.

References

1. Belousov, V.D.: Osnovi teorii kvazigrup i lup. Nauka, Moscow (1967) (in Russian)
2. Dénes, J., Keedwell, A.: Latin Squares and their Applications, Akadémiai Kiadó, Budapest. Academic Press, New York (1974)
3. McKay, B., Rogoyski, E.: Latin square of order 10. Electronic Journal of Combinatorics (1995), http://www.emis.de/journals/EJC/Volume_2/cover.html
4. Dénes, J., Keedwell, A.: A new authentication scheme based on latin squares. Discrete Mathematics (106/107), 157–161 (1992)
5. Arnold, R., Bell, T.: A corpus for evaluation of lossless compression algorithms. In: DCC 1997: Proceedings of the Conference on Data Compression, p. 201 (1997)
6. Dvorský, J., Ochodková, E., Snášel, V.: Hash Functions Based on Large Quasigroups. In: Proceedings of Velikonoční kryptologie, Brno, pp. 1–8 (2002)
7. Ochodková, E., Snášel, V.: Using Quasigroups for Secure Encoding of File System. In: Proceedings of the International Scientific NATO PfP/PWP Conference Security and Information Protection 2001, Brno, Czech Republic, pp. 175–181 (2001)
8. Smith, J.D.H.: An introduction to quasigroups and their representations. Chapman and Hall, Boca Raton (2007)