Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 46 (2015) 759 – 765

Procedia Computer Science

International Conference on Information and Communication Technologies (ICICT 2014)

Design of Hash Algorithm using Latin Square

Rajib Ghosh*, Suyash Verma, Rahul Kumar, Sanoj Kumar, Siya Ram

Department of CSE, National Institute of Technology Patna, Bihar – 800005, INDIA

Abstract

In this paper, we have proposed a new and an efficient cryptographic hash function based on random Latin squares generated through padding, shift operations and non-linear transformations. The developed hash scheme satisfies basic as well as desirable properties of an ideal hash function. Generation of Latin Squares for each plain text block strengthens the hash function. Removal of Duplicates and random padding have been implemented to achieve confusion and diffusion. Use of repeated lookups on Latin squares, non-linear transformations and complex shift operations further increase the strength of our cryptographic hash function at a low computational overhead and ensures that the hashing algorithm satisfy the principal properties of pre-image resistance and collision resistance.

Keywords: *Latin Square; Hash Function; Cipher; Padding; Non-Linear Transformation*

1. Introduction

HASHING[1,2,5] is the transformation of an input string of characters into a usually shorter and unique fixed-length key. Hashing is used in various fields like databases, encryption algorithms, data structures, cryptography etc.

In databases, hashing is used to index and retrieve items because they expedite the process of finding an item in a database using the shorter hashed key rather than finding it using the original value. In data structures, hashing is a method to accelerate the process of storing, searching, inserting and deleting data into and from an array using a unique and shorter hashed key. The basic idea here is to compute the position of a data within the array using a

* Rajib Ghosh. Tel.: +91-8084023813
E-mail address: rajib.ghosh@nitp.ac.in

function that returns the position. The function is called ‘hash function’ and the array is called ‘hash table’.
A cryptographic hash function transforms an input string (or ‘message’) of any length into a unique and fixed-size string, called the hash value (or sometimes termed as a digital fingerprint, a checksum, a message digest or simply a digest).

An ideal cryptographic hash function possess following principal properties:

- **Compression**: Irrespective of the size of input *x*, the length of the output $y = h(x)$ is less that of *x* and the generated hash *y* is of a fixed-size regardless of the length of the input.
- **Efficiency**: For an input *x*, it must be easy to compute $h(x)$ and the computational effort required to compute $h(x)$ should not grow too fast with the increase in the length of *x*.
- **Pre-image Resistance**: Given a hash *y*, it should be practically impossible to compute a text that has a hash *y* i.e. given hash value h(x) , it should be very hard to find any m such that h(x) = h(m).
- **Second Pre-image Resistance :** Given an input *x*, it should be infeasible to compute $x' \neq x$ such that $h(x) = h(x')$.
- **Collision Resistance**: It should be infeasible to find two distinct inputs x’ and x, such that h(x’) = h(x).
- **Security**: The security of a hash algorithm comes from its ability to generate one-way hashes. The hash algorithm itself is made available to the public.

In this paper, we design an efficient hash function suitable for cryptography and satisfying the above properties. Section 2 explains recent developments in the field of cryptographic hash functions by different researchers. Section 3 describes the basic concepts behind Latin squares and operations on them to be used in our scheme. Design of the algorithm is given in Section 4. In Section 5, we carry out analysis of our scheme and present results to prove its strength. Section 6 gives the concluding remarks and scope for future work in this direction.

2. Recent Works

A lot of study has been done recently in the area of generation of efficient and secure cryptographic hashing algorithms which can support today’s lightweight cryptographic applications. Researchers Denis X. Charles and Kristin E. Lauter[1] have proposed constructing collision resistant hash functions using two families of expander graphs – families of Ramanujan graphs constructed by Lubotzky-Phillips-Sarnak and Pizer respectively. They have shown that when the hash function is constructed from one of Pizer’s Ramanujan graphs, collision resistance follows from hardness of computing isogenies between super-singular elliptic curves. Though inefficient to be applied in all situations, the hash generated from such expander graphs would be appropriate in scenarios where high security is desired. Some other researchers like Václav Snášel, Ajith Abraham, Jiří Dvorský, Pavel Krömer, and Jan Platoš[2] have used combinatorial designs called quasigroups for generating hashes. The implementation of their hash function depends on how look-up table has been implemented and as such they have used quasigroups of small order to make the hash function more efficient. Equivalent to the concept of quasigroups, a more familiar concept of Latin Squares has been used by Saibal K. Pal, Diwakar Bhardwaj, Rajat Kumar & Varun Bhatia[3] to develop a light weight hash function by employing repeated lookups on Latin Square, non-linear transformations and complex shift operations.

Jean-Philippe Aumasson, Luca Henzen, Willi Meier, María Naya-Plasencia[4] have presented a hash function family QUARK, inspired by stream cipher Grain and by the block cipher KATAN(among the lightest secure ciphers) which can be used for message authentication, stream encryption or authenticated encryption. They have experimented that the lightest instance of QUARK, U-Quark, conjecturally provides at least 64-bit security against all attacks (collisions, multi collisions, distinguishers, etc.) while consuming on average 2.44 µW at 100 kHz in 0.18 µm ASIC, showing that it compares well to previous tentative lightweight hash functions. In another instance, Iftach Haitner and Omer Reingold[5] have reexamined the notion of *interactive hashing*, introduced by Naor, Ostrovsky, Venkatesan and Yung, to yield a more versatile interactive hashing theorem while proving the security of a variant of Naor et al. protocol.

3. Latin Square

LATIN SQUARE[3] is an $n \times n$ square matrix containing n different symbols such that every symbol occurs exactly once in each row and exactly once in each column. Given below is a 3 x 3 Latin square having 1, 2 and 3 as three unique elements in each row and column.

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix}$$

Fig. 1. Latin Square

A Latin square L (n, n) is said to be reduced or normalized if and only if both its first row and first column consists of elements sorted in ascending order.

3.1 Lookup Operation on Latin Square

The result of the look up operation, B = $\{b_0, b_1, ..., b_n\}$ performed on a Latin square is the value found on the given row and column number of the Latin square. Mathematically, the operation is given by

$$b_i = \begin{cases} l * a_0, & i = 0 \\ b_{i-1} * a_i, & l \le i \le n-1 \end{cases}$$

where A = $\{a0, a1, ..., an\}$ is the input message block to be hashed and l is the leader indicating the row position to be addressed. Leader l is used only for the first lookup. Successively, the lookup result of the previous lookup is used to compute the row number for the next lookup. The figure provides a better understanding of the simple lookup operation

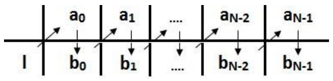

Fig. 2. Lookup Operation on Latin Square

4. Design of Hash Algorithm

4.1 Functions

- **Padding :** For each plain text block we have used random padding based on plain text blocks instead of constant padding with a single character to support two functions :

  (i) Latin Squares have been generated from plain text blocks and repetition is not allowed in Latin Squares

  (ii)To support confusion and diffusion while enciphering of the plain text block

  - **Method of Padding**

    (i) Pad the plain text block to twice the block size as following

    (a)Determine the last character of Plain Text Block P. Call this as *element* and find value of *key* from the Hash Map table for element

    (b) Calculate ($\sum(P_{even} * i^3_{even}) + \sum(P_{odd} * i^3_{odd})$)/blockSize. Call this as *determiner(det)*

    (c) Calculate *key = (key + det)* % n, n = number of different characters stored in Hash Map with their values

    (d) Pad the character corresponding to *key* from the HashMap

    (ii) Remove duplicates from the padded plain text block

(iii) Extract sub-list of size = block size from the above obtained output

- **Generate Latin Square :** The generated Latin Square is not commutative and as such it makes it very difficult for an attacker to reconstruct the original message from a given message digest.

  (a) In the first row of the Latin Square, put the padded plain text block
  (b) Fill the subsequent rows after right shifting the plain text block of the upper row by (block size/2)-1 characters

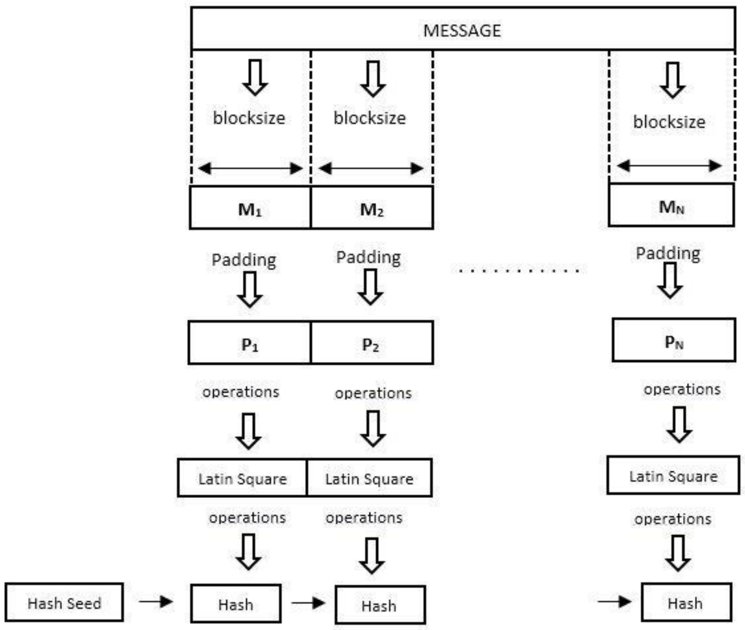

Fig. 3. Hash Algorithm Design

4.2 Hash Algorithm

Input: Message to be hashed (M)

Output: A hash of length of desired number of characters as string.

Scheme:

- Extract the first block P from the input plain text
- Modify P by removing duplicates and random padding to form pTemp
- Generate a Latin Square L based on pTemp
- Generate cipher text cTemp of pTemp
- Perform Lookup on Latin Square L as follows: for each character i of the plain text block calculate an array T1

  $T1_1 = (pTemp_i * cTemp_i)*IV$, if $i = 1$ (where IV is the initialization vector)
  $T1_i = (pTemp_i * cTemp_i)*T1_{i-1}$, if $i > 1$

- Modify T1 by removing duplicates and padding to block size to form T2
- Generate a Latin Square M1 from T2

- Generate cipher text CT2 of T2
- Perform Lookup on Latin Square L as follows: for each character i of the plain text block calculate an array T3

  $T3_1 = (T2_i * CT2_i)*IV$, $i = 1$ (where IV is the initialization vector)
  $T3_i = (T2_i * CT2_i)*T1_{i-1}$, $i > 1$

- Generate T4 by mixing T1 and T3 as follows :

  $T4_i = T1_i$, $i <=$ (length of T1)/2
  $T4_i = T3_i$, (length of T1)/2 $< i <=$ (length of T3)

- Shift right T4 by (length of T4)/2 -1 characters
- Generate T5 by removing duplicates from T4 and adding to T5 characters of the hash of the last plain text block
- Create Table M2 from T5 of dimension T5.size x T5.size
- Generate Hash for the current block by performing lookup on M2 and last Hash block by choosing leader as the last character of T5
- Repeat the above steps for all plain text blocks

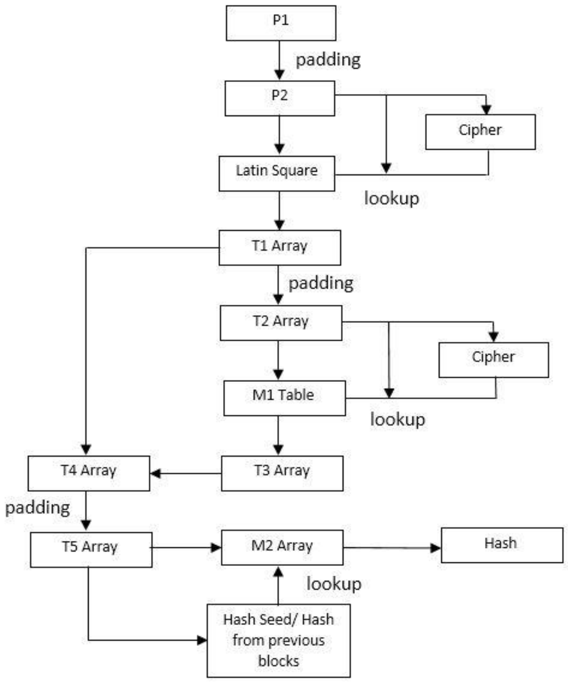

Fig. 4. Generation of Hash for a single plain text block

5. Results and Observations

We have used two sets of input data for analysis of this hash function. Firstly, we used an input set that contained all the permutations of the message “HELLOWORLD” (i.e. 604800 input messages with same characters but in all possible orders) with an output hash of size 48 characters. The message was chosen such that there are several repetitions of a character in the word. This input set showed the variations in the hash output for the different combinations of the same characters in the input message.
Secondly, we used an input set of messages that differed from a particular input message by just one character. This input set was chosen to show the variation in the output hash value on changing just character in a particular

input message. The input showed variations at many positions on the output hash for a change at one position in the input messages.

From the above two sets, we made the following observations with the hash outputs for the respective inputs:

- *Simple:* It is easy to compute a hash value for any input message. 'Easy' here refers to time complexity or the complexity of realizing the operations in a computer
- *Pre-image resistance:* A combination of multiple lookups in the Latin square, circular left shifts & non linear transformations makes it very hard to predict the input from the output
- *Collision Resistance:* For both the input sets, we did not find any repeating hash value. Therefore, it seems extremely unlikely that two different messages m1 and m2, with only one character different or with same characters but in different order, have hash (m1) = hash (m2)
- *Fixed size output:* The output is fixed of desired number of characters

6. Comparative Analysis

Our basic idea for designing a hash algorithm is inspired from "A New Cryptographic Hash Function based on Latin Squares and Non-linear Transformations" - Saibal K. Pal, Diwakar Bhardwaj, Rajat Kumar & Varun Bhatia, 2009 IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6–7 March 2009[1].

The operations used in the algorithm proposed by us are different from the ones used in the above paper that leads to more confusion and diffusion in the generated hash.

- **Latin Square**:- In our algorithm, a Latin Square is generated for each plain text block, unlike the above paper where a common random Latin Square is used for all plain text blocks.

- **Padding**:- We have used random padding to twice the block size instead of padding with constant characters of the plain text block. This random padding is necessary for the generation of Latin Square from a plain text block and also to lower the dependencies between a plain text block and its output hash. In our algorithm we have used random padding as follows:-

  (a) Determine the last character of Plain Text Block P. Call this as element and find value of key from the Hash Map table for element
  (b) Calculate $(\sum(P_{even} * i^3_{even}) + \sum(P_{odd} * i^3_{odd}))$/block Size. Call this as determiner(det)
  (c) Calculate key = (key + det) % n, n = number of different characters stored in Hash Map with their values
  (d) Pad the character corresponding to key from the Hash Map

- **Removing Duplicates** :- After padding, duplicates have been removed so as to generate a Latin Square. This step further lowers the possibility of determining the original message from a given output hash

7. Conclusion

In this paper we have introduced a design for Hash Function based on Latin Squares which we have implemented in RSA Digital Signature Scheme. This Hash Function is simple, robust and follows all basic properties of a hash like pre-image resistance, second pre-image resistance and collision resistance. The strength of our hash function lies in the generation of Latin Squares which depends on each plain text and is not commutative. As such, it makes it difficult for an attacker to create the original message from a given hash. Moreover, the use of complex shift operations and non-linear operations strengthens our hash algorithm. The hash function can be used for message integrity, entity authentication, validation, and non-repudiation.

Our hash function is at par with other available hash functions in terms of computation speed, collision resistance and pre-image resistance. We look forward to test and analyse our hash function exhaustively. We hope that the experimental results mentioned in this project about the cryptographic hash function will be interesting enough to attract attention of other fellow researchers.

References

1. Denis X. Charles and Kristin E. Lauter. Cryptographic Hash Functions from Expander Graphs. *International Association for Cryptologic Research* 2007

2. Václav Snášel, Ajith Abraham, Jiří Dvorský, Pavel Krömer, and Jan Platoš. Hash Functions Based on Large Quasigroup. *Springer-Verlag Berlin Heidelberg 2009*

3. Saibal K. Pal, Diwakar Bhardwaj, Rajat Kumar & Varun Bhatia. A New Cryptographic Hash Function based on Latin Squares and Non-linear Transformations. *IEEE International Advance Computing Conference (IACC 2009) Patiala*, India, 6– 7 March 2009.

4. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, María Naya-Plasencia. QUARK: A Lightweight Hash. *International Association for Cryptologic Research* 2012

5. Iftach Haitner, Omer Reingold. A New Interactive Hashing Theorem. *International Association for Cryptologic Research* 2013