Far East Journal of Mathematical Sciences (FJMS)
© 2014 Pushpa Publishing House, Allahabad, India
Published Online: April 2014
Available online at http://pphmj.com/journals/fjms.htm
Volume 86, Number 1, 2014, Pages 23-36

ALGORITHM CONSTRUCTION OF HLI HASH FUNCTION

Rachmawati Dwi Estuningsih[1], **Sugi Guritman**[2] **and Bib P. Silalahi**[2]

[1]Academy of Chemical Analysis of Bogor
Jl. Pangeran Sogiri No 283 Tanah Baru
Bogor Utara, Kota Bogor, Jawa Barat
Indonesia

[2]Bogor Agricultural Institute
Indonesia

Abstract

Hash function based on lattices believed to have a strong security proof based on the worst-case hardness. In 2008, Lyubashevsky et al. proposed SWIFFT, a hash function that corresponds to the ring $R = \mathbb{Z}_p[\alpha]/(\alpha^n + 1)$. We construct HLI, a hash function that corresponds to a simple expression over modular polynomial ring $R_{f,p} = \mathbb{Z}_p[x]/f(x)$. We choose a monic and irreducible polynomial $f(x) = x^n - x - 1$ to obtain the hash function is collision resistant. Thus, the number of operations in hash function is calculated and compared with SWIFFT. Thus, the number of operations in hash function is calculated and compared with SWIFFT.

1. Introduction

Cryptography is the study of mathematical techniques related to aspects

Received: September 19, 2013; Revised: January 27, 2014; Accepted: February 7, 2014
2010 Mathematics Subject Classification: 94A60.
Keywords and phrases: hash function, modular polynomial ring, ideal lattice.

of information security relating confidentiality, data integrity, entity authentication, and data origin authentication. Data integrity is a service, which is associated with the conversion of data carried out by unauthorized parties [5]. To maintain data integrity hash functions can be used. Hash function is a computationally efficient function to map an arbitrary length bitstring to a fixed length bitstring called as *hash value*. The use of hash function in maintaining the information integrity and authentication is in among the creation of digital signatures, virus protection, and software distribution.

Hash function widely used today is generally designed based on Boolean arithmetic, similar to the construction pattern of symmetric key cryptosystem algorithm. However, the hash function has received many attacks to undermine its security properties. The effectiveness of the attacks is associated with the function construction concerned that are generally not accompanied by proof of theoretic security that is based on mathematical computation problems.

It is this that has attracted the interest of cryptography researchers to design hash functions algorithms which is efficient and supported by proof of theoretical security based on mathematical computing problems. Lattice computational problem is expected to be the basis in order to obtain ideal hash functions since it has several advantages from lattice problem, that is promising proofing of strong security based on worst-case hardness, efficient implementation, and lattice based-cryptography which is believed to be secure against quantum computers.

Construction of lattice-based cryptography was first suggested by Ajtai in 1996. Ajtai defined a one-way functions family whose security is footed on the worst-case computing problem from the SVP approximation with n^c factor, in which n is the lattice dimension and constant $c > 0$ [1]. Subsequently, Goldreich et al. showed that the one-way function is a collision resistance hash function [2].

Micciancio used cyclic lattice as a new source of complexity and formed an efficient one-way function from the worst-case complexity assumptions [6]. Peikert and Rosen modified the assumption from function established by Micciancio being able to show that the function is a collision resistance hash function [8].

Lyubashevsky et al. constructed lattice-based hash function algorithms prioritizing the efficiency called SWIFFT [4]. Basically, SWIFFT hash functions is a highly optimized variant of the hash function and is very efficient in practice because of the use of Fast Fourier Transform (FFT) in Z_q.

The purpose of this study is to construct an ideal lattice-based hash function algorithm, analyzing the speed of the hash functions as a result of construction, and comparing it with SWIFFT.

2. Preliminaries

2.1. Hash functions

Cryptographic hash functions play a fundamental role in modern cryptography. Hash functions take a massage as input and produce an output referred to as hash value.

Definition 2.1 [5]**.** A hash function (in the unrestricted sense) is a function h which has, as a minimum, the following two properties:

(1) Compression – h maps an input x of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength n.

(2) Ease of computation – given h and an input x, $h(x)$ is easy to compute.

2.2. Algebra

Suppose $\mathbb{Z}[x]$ is a set of polynomials with integer coefficients. While $\mathbb{Z}_p$ and $\mathbb{Z}_p[x]$ in a row are set of modulo p integers and a set of polynomials with integer coefficients modulo p. Modular polynomial ring noted as $R_{f,p}$

$= \mathbb{Z}_p[x]/f(x)$ is a set of all polynomials of degree at most $n-1$ with coefficients in $\mathbb{Z}_p$. Mathematically it can be written

$$R_{f,p} = \{g \bmod f \mid g \in \mathbb{Z}_p[x]\}$$

$$= \{a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}/a_i \in \mathbb{Z}_p\},$$

where $f(x) = f_0 + f_1x + f_2x^2 + \cdots + f_{n-1}x^{n-1} + f_nx^n \in \mathbb{Z}[x]$.

For any $a(x), b(x) \in R_{f,p}$ which is written as

$$a(x) = \sum_{i=0}^{n-1} a_ix^i \quad \text{and} \quad b(x) = \sum_{i=0}^{n-1} b_ix^i,$$

then the addition operation is defined as follows:

$$a(x) + b(x) = \sum_{i=0}^{n-1}((a_i + b_i) \bmod p)x^i$$

while the multiplication operation is

$$a(x) \otimes b(x) = \sum_{i=0}^{n-1}\left(\left(\sum_{i+j=k} a_ib_j\right) \bmod p\right)x^k \bmod f(x).$$

The simpler representation of $R_{f,p}$ and which is easier to implement in a computer is that $R_{f,p}$ can be considered as a set of all modulo *p* integer vectors in dimension *n*, i.e.

$$R_{f,p} = \{(a_0, a_1, a_2, ..., a_{n-1})/a_i \in \mathbb{Z}_p\} \in \mathbb{Z}_p^n.$$

So it can be seen that the operation of addition on the ring $R_{f,p}$ is the sum of modulo *p* integer vector. Suppose for any $a, b \in R_{f,p}$ written $a = (a_0, a_1, a_2, ..., a_{n-1})$ and $b = (b_0, b_1, b_2, ..., b_{n-1})$, then

$$a + b = (a_0 + b_0, a_1 + b_1, a_2 + b_2, ..., a_{n-1} + b_{n-1}).$$

While the scalar *k* and vector *a* multiplication operation can be viewed as a summation of as much *k* times of vectors *a*. Thus $R_{f,p}$ against the vector operation is lattice $\mathbb{Z}_p^n$. A lattice that is defined from certain polynomial ring is called *ideal lattice*.

Lyubashevsky and Micciancio showed that to obtain the hash functions that have impact resistant properties, monic and irreducible polynomial f should be chosen. Monic polynomial is a polynomial with the coefficient of the highest power of x is one. While a polynomial is irreducible if it cannot be represented as a product of lower degree polynomial [3].

3. Algorithm Construction

Hash function is constructed based on the results of algebraic operations on modular polynomial ring $R_{f,p} = \mathbb{Z}_p[x]/f(x)$. In this study $f(x) = x^n - x - 1$, which constitutes monic and irreducible polynomials is chosen. The election of modulo $f(x) = x^n - x - 1$ led the multiplication of any two members of modular polynomials ring $R_{f,p}$ to be

$$x^n = (x^n - x - 1) + (x + 1) \Leftrightarrow x^n \operatorname{mod}(x^n - x - 1) = x + 1$$

$$\Leftrightarrow x^n \equiv x + 1(\operatorname{mod}(x^n - x - 1)).$$

From these results, taken $p(x) = x$, then

$$\begin{aligned} p(x)a(x) &= (a_0 x + a_1 x^2 + \cdots + a_{n-2}x^{n-1} + a_{n-1}x^n) \operatorname{mod}(x^n - x - 1) \\ &= a_{n-1}(x+1) + a_0 x + a_1 x^2 + \cdots + a_{n-2}x^{n-1} \\ &= a_{n-1} + (a_{n-1} + a_0)x + a_1 x^2 + \cdots + a_{n-2}x^{n-1} \end{aligned}$$

and furthermore obtained

$$x^2 a(x) = a_{n-2} + (a_{n-2} + a_{n-1})x + (a_{n-1} + a_0)x^2 + a_1 x^3 + \cdots + a_{n-3}x^{n-1},$$

$$\begin{aligned} x^3 a(x) = a_{n-3} &+ (a_{n-3} + a_{n-2})x + (a_{n-2} + a_{n-1})x^2 + (a_{n-1} + a_0)x^3 \\ &+ a_1 x^4 + \cdots + a_{n-4}x^{n-1}. \end{aligned}$$

In general, for $i = 1, 2, \ldots, n$ apply

$$x^i a(x) = a_{n-i} + (a_{n-i} + a_{n-i+1})x + \cdots + (a_{n-1} + a_0)x^i + a_1 x^{i+1}$$
$$+ \cdots + a_{n-i-1}x^{n-1}.$$

So the multiplication algorithm in $R_{f,p}$ which is implemented in a computer would be easier if it is represented in a form of vector as follows:

```
Input: vectors $a = (a_0, a_1, a_2, \ldots, a_{n-1})$ and $b = (b_0, b_1, b_2, \ldots, b_{n-1})$
       in $R_{f,p}$.
Output: vector $c$ as a hash multiplication of $a$ and $b$ in $R_{f,p}$;
    1. Initialization $c := b_0 a \bmod p$ and $w := a$.
    2. For integer $i = 1$ to $i = n - 1$, count:
        a. $v := (\hookrightarrow w)$, where $\hookrightarrow w$ denotes the rotation of $w$ to the right
           one unit.
        b. $w := v + d_i$, where $d_i$ is the vector whose all components are
           0 except the $i + 1$ component is $a_{n-i}$.
    3. If $b_i \neq 0$, count $c := (c + b_i w) \bmod p$.
    4. Return (c).
```

The steps in the above algorithm are essentially a modular integer matrix multiplication $AB \equiv C(\bmod p)$, i.e.

$$\begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \cdots & a_1 \\ a_1 & a_{n-1} + a_0 & a_{n-2} + a_{n-1} & \cdots & a_1 + a_2 \\ a_2 & a_1 & a_{n-1} + a_0 & \cdots & a_2 + a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_{n-1} + a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix} (\bmod p).$$

From here an ideal lattice-based hash function family can be constructed, that is by selecting parameter *m*, *n*, *d* an integer where *m* is divisible by *n* and *p* is prime. The input of this hash function is $x \in \mathbb{Z}_d^m$ with $d < p$. This input is divided into m/n vectors namely $x = x^{(1)}, x^{(2)}, ..., x^{(m/n)} \in \mathbb{Z}_d^m$, with $x^{(i)} \in \mathbb{Z}_d^n$, $i = 1, 2, ..., m/n$. The key of hash functions is any two vectors $a^{(1)}, b \in R_{f,p} \cong \mathbb{Z}_p^n$. It is of this key that other vectors $a^{(i)} = a^{(i-1)} \otimes x^{(i-1)} + b \in R_{f,p}$ for $i = 2, 3, ..., m/n$ will be generated. Furthermore, hash function family can be defined:

$$\mathcal{H}_a(f, p, m, n) = \{h_a : \mathbb{Z}_d^m \to R_{f,p} \cong \mathbb{Z}_p^n\}$$

which $h_a(x) = \sum_{i=1}^{m/n}(a^{(i)} \otimes x^{(i)} + b)$.

The following algorithm for hash functions:

Keywords: two arbitrary vectors $a^{(1)}, b \in R_{f,p} \cong \mathbb{Z}_p^n$.

Input: $x = x^{(1)}, x^{(2)}, ..., x^{(m/n)} \in \mathbb{Z}_d^m$ with $x^{(i)} \in \mathbb{Z}_d^n$, $i = 1, 2, ..., m/n$.

Output: $h_a(x) = \sum_{i=1}^{m/n}(a^{(i)} \otimes x^{(i)} + b)$ as a result of multiplication and addition the ring $R_{f,p}$;

```
1. Initialize $h := 0$.
2. For integers $i = 1 s/d i = m/n$, compute:
    a. $k^{(i)} = a^{(i)} \otimes x^{(i)}$.
    b. $h^{(i)} = k^{(i)} + b$.
    c. $a^{(i+1)} = h^{(i)}$.
    d. $h = h + h^{(i)}$.
3. Return (h).
```

Because the hash function must then be compression in nature, so the output length must be smaller than the length of the input, therefore the parameters chosen above must meet $n \log p \langle m \log d \Leftrightarrow m \rangle \frac{n \log p}{\log d}$.

4. Speed Analysis

Speed analysis means counting the number of operations in constructed hash algorithm functions. First, the number of operations in the summation algorithm in ring $\boldsymbol{R}_{\boldsymbol{f}, \boldsymbol{p}}$ will be counted as follows:

1. An assignment operation as initial statement to the variable as the number of vector elements to be summed.
2. The core of this summation operation, is in the statement block 'if' namely an operation to check whether one of the vectors to be summed is the zero vector then:
    a. If one of the vectors is a zero vector then there is an assignment which considers the result of summation constitutes as another vector. In other words, the summation operation of modulo *p* is not needed.
    b. If both vectors are not zero vectors, then the addition operation of modulo *p* summated to *n* times, shall be made.

However, in the multiplication algorithm in ring**,** there are two algorithms. First, the multiplication algorithm of modulo *p* vector and scalar. Then, the algorithm which turns vector to the right as much as one unit. Therefore, before calculating the number of multiplication operations in the ring, the operation of scalar and vector multiplication of modulo *p* will be counted as follows:

1. An assignment operation that state initial variables *n* is the number of vector elements.
2. In calculating the result of multiplication there are *n* times of modulo *p* operations.

While the number of operating in algorithms turns vector one unit to the right:

1. An assignment operation expressing early *n* variables is the number of vector elements.
2. In turning vectors there are two operations, namely taking the last element of the vector and then insert the element as the first element of the vector.

The calculation of many operations of two algorithms above can be used to calculate the number of operations of two vector multiplication algorithms in the ring, i.e.:

1. There are 2 assignment operations.
2. Multiplication of scalar and modulo *p* vector which involves *n* operation of modulo *p* multiplication.
3. In the statement block 'for' those repeated $n-1$;
   a. There are three operations i.e. rotating the vector, summation, and insertion.
   b. If $b_i \neq 0$ then there are *n* operations of modulo *p* multiplication and *n* operations of modulo *p* summation.

The number of hash function algorithms can be computed as follows:

1. There are five operating assignments.
2. In a block of statements 'for' those repeated m/n times, i.e.:
   a. Four assignments.
   b. A multiplication in the ring $\boldsymbol{R}_{\boldsymbol{f},\,\boldsymbol{p}}$ which involves n^2 multiplication operation of modulo *p*.
   c. Two additions in the ring $\boldsymbol{R}_{\boldsymbol{f},\,\boldsymbol{p}}$ each of which involves *n* operation of modulo *p* summation.

If parameter $d = 2$ is taken then hash function algorithm does not involve multiplication in ring $\boldsymbol{R}_{\boldsymbol{f},\,\boldsymbol{p}}$. So it only involves summation ring $\boldsymbol{R}_{\boldsymbol{f},\,\boldsymbol{p}}$, which consists of n summation operations of modulo p which is repeated m/n times. So hash function algorithm proceeds with parameter $d = 2$ and input with length m consisting of m operations of modulo p summation.

Besides calculating the vast number of operations in algorithm, experiment of calculating the length of time the program calculates the hash value is also conducted. In this experiment parameters as in SWIFFT, namely $d = 2$, $n = 64$, $p = 257$ were taken, while the length of the input (m) is taken within 6 points. To obtain the value of time, here each m value is repeated 5 times and then the average is calculated, so that obtained the following results:

Table 1. Running program HLI time with different input lengths

Input length (*m*)	Time (sec)
1024	0,33
2048	0,643
4096	1,373
8192	2,718
16384	5,357
32768	10,745

In the graph:

From Figure 1, it can be seen that for binary inputs the result is the running time linear to length of input.

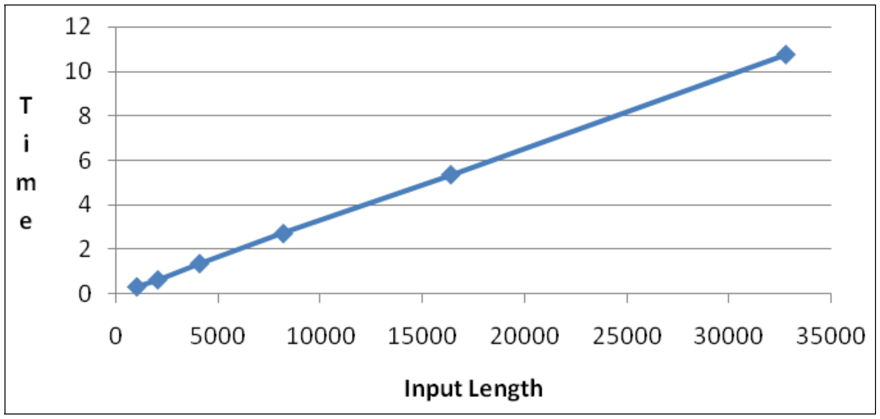

Figure 1. Figure HLI program running time with parameters $n = 64$, $d = 2$, and $p = 257$.

5. Security Analysis

Security of a hash function can be seen from two properties i.e. one-way and collision resistant. The proof of hash function based on lattice is one-way follow the proof of Micciancio [6]. The proof of hash functions based on lattice with $f(x)$ over Z irreducible is resistant collisions has been demonstrated by Lyubashevsky and Micciancio [3].

According Lyubashevsky and Micciancio [3] to obtain collision-resistant properties the polynomial $f(x)$ must be:

a. $f(x)$ is a monic polynomial of degree n, irreducible over Z.

b. For every unit vectors $u, v \in \mathbb{Z}_p[x]/\langle f(x)\rangle$, product of u and v is a short vector, it is means $\| uv \|$ is limited to $\sqrt{n}$.

Polynomial $f(x) = x^n - x - 1$ is a monic polynomial of degree n because the coefficient of leader is one. Parameter n is adjusted for application on computer that is power of 2. For $n = 8, 16, 32, 64, 128, 256, 512, 1024,$ polynomial $f(x) = x^n - x - 1$ is irreducible. It is checked by mathematical software. For point *b* is shown by this theorem.

Theorem. *For every unit vector* $u, v \in \mathbb{Z}_p^n \cong \mathbb{Z}_p[x]/\langle f(x)\rangle$, *if* $h = u \odot v$, *then* $\| h \| < \sqrt{n}$.

Proof. For every unit vector $u, v \in \mathbb{Z}_p^n \cong \mathbb{Z}_p[x]/\langle f(x)\rangle$, the polynomial representation can be written by $u(x) = x^j$ and $v(x) = x^k$ with $j, k = 0, 1, 2, \ldots, n-1$. So the representation of *h* is

$$h(x) = u(x)v(x) \bmod f(x) = x^{j+k} \bmod f(x).$$

If $j + k < n$, then it is proved because $h(x) = x^{j+k}$ so that *h* is a unit vector and $\| h \| = \sqrt{1}$. If $j + k = n$, then it is proved because $h(x) = 1 + x$ so that $\| h \| = \sqrt{1^2 + 1^2} = \sqrt{2}$. If $j + k > n$, then there is $l = j + k - n$ with $1 \le l \le n - 2$ so that

$$h(x) = (x^l \cdot x^n) \bmod f(x) = x^l(1 + x) \bmod f(x) = x^l + x^{l+1} \bmod f(x).$$

Because of $l \le n - 2$ then $l < n$ and $l + 1 \le n - 2 + 1 < n$ so obtained $h(x) = x^l + x^{l+1}$. Therefore $\| h \| = \sqrt{1^2 + 1^2} = \sqrt{2}$. □

Although the theorem states that $\| h \|$ is limited to $\sqrt{n}$, but from the proof $\| h \|$ is $\sqrt{2}$ that smaller than $\sqrt{n}$ for $n > 2$.

6. Comparison

Here is a comparison between HLI and SWIFFT:

Table 2. Comparison of key sizes and HLI SWIFFT with $n = 64$, $d = 2$, $m = 1024$, $p = 257$

Size (byte)	SWIFFT	HLI
Input	1024	1024
Key	8192	1025
Output	513	513

Table 3. Comparison SWIFFT and HLI

Description	SWIFFT	HLI
Key	m/n arbitrary vectors in $\mathbb{Z}_p^n$	arbitrary vectors in $\mathbb{Z}_p^n$
Calculation	Using the Fast Fourier Transform	Using modular polynomial ring operating in $R_{f,p} = \mathbb{Z}_p[x]/f(x)$
Time	$O(n)$	$O(n)$

7. Conclusion

Hash function constructed with parameter *m*, *n*, *d* are integer and prime number *p* is

$$h_a(x) = \sum_{i=1}^{m/n} (a^{(i)} \otimes x^{(i)} + b),$$

where the input of hash function is $x = x^{(1)}, x^{(2)}, \ldots, x^{(m/n)} \in \mathbb{Z}_d^m$ and the key is $a^{(1)}, b \in R_{f,p} \cong \mathbb{Z}_p^n$. The hash function involves only n^2 of multiplication of modulo *p*. However, if the input of hash function is a binary number then the hash function HLI involves only *n* multiplication operations of modulo *p*. So the time spent is almost equal to SWIFFT. The advantage of HLI is key size which is smaller than SWIFFT.

References

[1] M. Ajtai, Generating hard instances of lattice problems, Proceedings of the 28th Annual ACM Symposium on Theory of Computing, May 22-24, Philadelphia, PA, USA, 1996, pp. 99-108.

[2] O. Goldreich, S. Goldwasser and S. Halevi, Collision-free hashing from lattice problems, Technical Report TR96-056, Electronic Colloquium on Computational Complexity, 1996.

[3] V. Lyubashevsky and D. Micciancio, Generalized compact knapsacks are collision resistant, ICALP'06, Proceedings of the 33rd International Conference on Automata, Languages and Programming - Part II, 2006, pp. 144-155.

[4] V. Lyubashevsky, D. Micciancio, C. Peikert and A. Rosen, SWIFFT: A modest proposal for FFT hashing, Proceedings of Fast Software Encryption, 2008.

[5] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[6] D. Micciancio, Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions, Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002, pp. 356-365.

[7] D. Micciancio and O. Regev, Worst-case to average-case reduction based on Gaussian measures, Proc. 45th Annual IEEE Symp. on Foundations of Science 2004, pp. 372-381.

[8] C. Peikert and A. Rosen, Efficient collision-resistant hashing from worst-case assumptions on cyclic lattice, 3rd Theory of Cryptography Conference (TCC), 2006.