JP Journal of Algebra, Number Theory and Applications

Published Online: April 2015
http://dx.doi.org/10.17654/JPANTAApr2015_177_188
Volume 36, Number 2, 2015, Pages 177-188 **ISSN: 0972-5555**

CONSTRUCTION OF FAMILY OF HASH FUNCTIONS BASED ON IDEAL LATTICE

Sugi Guritman, Nur Aliatiningtyas, Teduh Wulandari and Muhammad Ilyas

Mathematics Department
Faculty of Mathematics and Natural Sciences
Bogor Agricultural University
Indonesia

Abstract

Cryptographic hash function is a function $h : D \to R$ with $|D| \gg |R|$ which has the security properties: one-way (computationally not feasible to calculate $x \in D$ from known $y \in R$ so that $y = h(x)$) and collision resistant (not feasible determining $x \neq y$ so that $h(x) = h(y)$). Family of hash functions based on ideal lattice is formulated as matrix multiplication $\mathbf{y} = h_{\mathbf{A}}(\mathbf{x}) = \mathbf{Ax}$ with computation involving ring arithmetic $\mathbb{Z}_p[x]/\langle f(x)\rangle$. The main problem is to construct an efficient algorithm from the formulation with the objective: the security properties are met and the key size is reduced. Related to this issue and following the results of previous studies, in this article, general algorithms for the family of hash function based on ideal lattice are constructed. In this case, $\mathbb{Z}_p[x]/\langle f(x)\rangle$ family is defined as the arithmetic ring determined by the family of irreducible trinomial $f(x) = f_0 + f_i x^i + x^n$ with $f_i, f_0 \in \{-1, 1\}$. Then, as the key: one

Received: December 18, 2014; Accepted: February 24, 2015
2010 Mathematics Subject Classification: 94A60.
Keywords and phrases: ring polynomial arithmetic algorithms, hash function, ideal lattice.
Communicated by K. K. Azad

randomly chosen $\mathbf{a} \in \mathbb{Z}_p[x]/\langle f(x)\rangle$ and randomly selected t trinomial $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_t$, then the compression function of hash function ideal lattice is defined as a recursive process that processes the message block $\mathbf{x}$ to the value $\mathbf{y} = \sum_{i=1}^{t} \mathbf{a} \odot \mathbf{x} \pmod{\mathbf{f}_i}$. The study also covers issues related to processing speed and security aspects of the hash function construction results.

1. Introduction

According to Menezes et al. [9], cryptography is the study of mathematical techniques related to information security purposes such as confidentiality, data integrity, entity authentication, and data origin authentication. One of cryptographic primitives related to the purpose of data integrity and authentication security is a *hash function*. The hash function is a function $h : D \rightarrow R$ with $|D| \gg |R|$ and has the security properties: *one-way* and *collision resistance*. The function h is *one-way* if it is computationally infeasible to calculate $x \in D$ of knowing $y \in R$ so that $y = h(x)$, and h collision resistant if computationally not feasible to determine $x, y \in D$ with $x \neq y$ so that $h(x) = h(y)$.

The rise of attacks on the security properties of the hash function which the construction is based on *boolean* arithmetic, intensified the researchers to turn to the construction of cryptographic hash functions with the security is based on *lattice computational problems*. Lattice Λ with dimension n is the set of all integer linear combinations of n linearly independent vectors with the problem: *computationally it is infeasible to determine the shortest vector in* Λ.

Started by Ajtai [1] that defines a one-way hash function families with security rests on lattice computational problems. In this case, for integer positive parameters: d, m, n, with $m > n$, and p primes, choose a matrix $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ at random, hash function $h_{\mathbf{A}}$ with *key* $\mathbf{A}$ is defined as

$$h_{\mathbf{A}} : \mathbb{Z}_d^m \rightarrow \mathbb{Z}_p^n, \text{ where } \mathbf{y} = h_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}.$$

Refining the work from Ajtai, Goldreich et al. [5] proved that $h_{\mathbf{A}}$ is collision resistant. Then, security assumption is strengthened in: [3], [10] and [11] also shown an aspects of computational efficiency is not enough when it comes to the application.

Furthermore, the attempt on efficiency listed in [12] and [7] that generalized the structure of **A** from the arithmetic *field* $\mathbb{Z}_p$ to the modular polynomial arithmetic ring $\mathbb{Z}_p[x]/\langle f(x)\rangle$ for any polynomial $f(x) \in \mathbb{Z}[x]$. Lattice with column vectors of **A** as the basis and defined on modular arithmetic polynomial ring is called *Ideal Lattice*. Apparently, hash function based on collision resistant lattice is not ideal for any $f(x)$. Finally, it has been proved in the article [7] that the hash function based on ideal lattice is collision resistant guaranteed if $f(x)$ is chosen to satisfy two properties, namely the irreducible over $\mathbb{Z}$ and for every unit vectors **u** and **v**, the ring product of **u** and **v** is a short vector (limited by $\sqrt{n}$).

Guritman et al., in the article [6], have constructed an efficient algorithms for an arithmetic ring family that is determined by irreducible trinomial family $f(x) = f_0 + f_i x^i + x^n$ with $f_i,\ f_0 \in \{-1, 1\}$. By using the arithmetic, the purpose of this article is to construct a family of hash function in which the compression functions is defined as a recursive process that processes binary message block **x** to the value $\mathbf{y} = \sum_{i=1}^{t} (\mathbf{a} \odot \mathbf{x}_i) \bmod \mathbf{f}_i$. In this case, the vector $\mathbf{a} \in \mathbb{Z}_p[x]/\langle f(x)\rangle$ and trinomial $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_t$ is defined as key parameters chosen randomly and uniformly. The construction of this hash function is a generalization of the construction of hash function in [14]. The schematic discussion includes three sections. Section 2 contains a review of algorithms for modular arithmetic ring polynomial. Section 3 is a discussion of core construction algorithm related to hash function is concerned. Finally, Section 4 discusses the security aspects of a hash function construction.

2. Modular Arithmetic Ring Polynomial

In this section, we describe the algorithm for arithmetic ring polynomial modular $\mathbb{Z}_p[x]/\langle f(x)\rangle$ with parameters primes p and f is a polynomial of three terms (trinomial) is defined as

$$f(x) = f_0 + f_i x^i + x^n, \text{ where } f_i,\ f_0 \in \{-1, 1\} \tag{1}$$

with integer i selected in the interval $1 \le i \le n-1$. A complete review of this topic refers to article [6].

We denote $\mathbb{Z}_p = \{0, 1, 2, ..., p-1\}$ as a field over prime integer (hereinafter, simply called *prime field*) with the addition and multiplication operation modulo p. Then, $\mathbb{Z}_p[x]$ as a polynomial ring over $\mathbb{Z}_p$ with the addition and multiplication operation over $\mathbb{Z}_p$. Then, $\mathbb{Z}_p[x]/\langle f(x)\rangle$ as modular ring polynomial whose members all polynomials over $\mathbb{Z}_p$ and with degree at most $n-1$ with the addition and multiplication operation modulo $f(x)$. In this case, $\mathbb{Z}_p[x]/\langle f(x)\rangle$ also has a structure as vector space over $\mathbb{Z}_p$ with the addition and multiplication polynomial operation.

From the fact that the vector space $\mathbb{Z}_p[x]/\langle f(x)\rangle \cong \mathbb{Z}_p^n$, then the computational aspects is much simpler from each

$$a(x) = (a_0 + a_1 x + \cdots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1}) \in \mathbb{Z}_p[x]/\langle f(x)\rangle$$

can be represented (isomorphic) as vector data

$$\mathbf{a} = (a_0, a_1, ..., a_{n-2}, a_{n-1}) \in \mathbb{Z}_p^n.$$

As a result, the amount of computational operations in the modular ring $\mathbb{Z}_p[x]/\langle f(x)\rangle$ is as efficient as computing the vector addition operation modulo p. Moreover, the efficiency of the multiplication operation $\mathbb{Z}_p[x]/\langle f(x)\rangle$ described as follows.

Let $a(x), b(x) \in \mathbb{Z}_p[x]/\langle f(x)\rangle$ represented by vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$, and $[a(x) \cdot b(x)] \bmod f(x)$ is represented as $(\mathbf{a} \odot \mathbf{b}) \bmod \mathbf{f}$ multiplication operation in $\mathbb{Z}_p^n$. Suppose f in equation (1) represented as an ordered pair $\mathbf{f} = (f_0, j) \in \{-1, 1\} \times \{\pm 1, \pm 2, ..., \pm(n-1)\}$ with $i = |j|$, $f_i = 1$ if $j > 0$, and $f_i = -1$ if $j < 0$. As an illustration, for $n = 64$, $\mathbf{f} = (1, -37)$ is a representation of the trinomial $f(x) = 1 - x^{37} + x^{64}$. Thus, the calculation of $xa(x) \bmod f(x)$ can efficiently be demonstrated through the following algorithm.

Algorithm 1 (**Rotation-Substitution Algorithm**)

Input: Integer n with $n > 1$, odd prime p, ordered pair $\mathbf{f} = (f_0, j)$ as a representation of trinomial $f(x)$, and vector $\mathbf{a} = (a_0, a_1, a_2, ..., a_{n-1}) \in \mathbb{Z}_p^n$ as a representation of $a(x) \in \mathbb{Z}_p[x]/\langle f(x)\rangle$.

Output: The vector $\mathbf{c} = (c_0, c_1, c_2, ..., c_{n-1})$ as $xa(x) \in \mathbb{Z}_p[x]/\langle f(x)\rangle$

1. $\mathbf{c} := (\hookrightarrow \mathbf{a})$, where $\hookrightarrow \mathbf{a}$ denotes the rotation of **a** one component to the right.
2. $\mathbf{c} := \mathbf{subs}(0, -f_0 a_{n-1}, \mathbf{c})$ denotes the substitution 0th component from **c** with $(-f_0 a_{n-1})$.
3. If $j > 0$, compute $s = a_{j-1} - a_{n-1}$, $\mathbf{c} := \mathbf{subs}(j, s, \mathbf{c})$, and
   if $j < 0$, compute $s = a_{j-1} + a_{n-1}$, $\mathbf{c} := \mathbf{subs}(-j, s, \mathbf{c})$.
4. **return**(**c**).

Furthermore, since $a(x)b(x) \bmod f(x)$ can be written as

$$((b_0 \cdot a(x)) + b_1(x \cdot a(x)) + b_2(x^2 \cdot a(x)) + \cdots + b_{n-1}(x^{n-1} \cdot a(x))) \bmod f(x)$$

the calculation of $(\mathbf{a} \odot \mathbf{b}) \bmod \mathbf{f}$ efficiently is demonstrated through the following algorithm.

Algorithm 2 (**Operation Algorithm** $(\mathbf{a} \odot \mathbf{b}) \bmod \mathbf{f}$)

Input: Vector $\mathbf{a} = (a_0, a_1, a_2, ..., a_{m-1})$ and $\mathbf{b} = (b_0, b_1, b_2, ..., b_{n-1})$ in the ring $\mathbb{Z}_p^n \cong \mathbb{Z}_p[x]/\langle f(x)\rangle$.

Output: The vector $\mathbf{c} = (c_0, c_1, c_2, ..., c_{n-1})$ as the product of **a** and **b** in the ring $\mathbb{Z}_p^n$.

1. Initialization $\mathbf{c} := b_0\mathbf{a}$ (denotes a scalar times vector) and $\mathbf{w} := \mathbf{a}$.

2. For integer $i = 1$ to $i = n - 1$, calculate:

   (a) $\mathbf{w} :=$ **RotSubs**$(\mathbf{w}, \mathbf{f})$ call Algorithm 1.

   (b) If $b_i \neq 0$, calculate $\mathbf{c} := \mathbf{c} + b_i\mathbf{w}$.

3. **return**(**c**).

Since the speed of Algorithm 1 is constant on the growth of the value of *n*, the most dominant influence computational efficiency of Algorithm 2 is the number of addition and multiplication operations modulo *p*. In this case, we could observe that Algorithm 2 involves at most n^2 multiplication operation modulo *p* and at most $n(n-1)$ addition operation modulo *p*. Related to Algorithm 2 as a subroutine in the construction of a hash function algorithm (described in Section 3), it is assumed that **b** is a binary vector. Therefore, it is easily observed that Algorithm 2 only requires an average of $\left(\frac{(n-1)n}{2}\right)$ addition operation modulo *p*, equivalent to $\left(\frac{(n-1)n}{2}\right)\lg p$ bits operation. By setting the parameter *p* as constant, the size of the efficiency of Algorithm 2 is $\mathcal{O}(n^2)$ bit operations[1].

[1] $\mathcal{O}$ (read "big oh") is a measure of the speed of a mathematical algorithm asymptotically. Formal definition can refer to standard text books containing the algorithm discussion.

3. Hash Function Construction

Following the definition of the hash function and its construction techniques in the [9], the definition and construction of hash functions in this article is enough to define the compression function just as well as construct the algorithm. In this case, the hash function is constructed based on lattice, the family of hash functions that compression is defined as

$$h_{\mathbf{A}} : \mathbb{Z}_d^m \to \mathbb{Z}_p^n \text{ with } \mathbf{y} = h_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

with $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ is viewed as a key $(\text{length}(mn \lg p)\,\text{bits})$ are chosen at random after the positive integer parameter values m, n, d (d is relatively small value) is set, with compression condition $m \approx \frac{2n \lg p}{\lg d}$. The process of computing a matrix multiplication $\mathbf{Ax}$ which means involving $(m \times n)$ multiplication operation modulo p.

Furthermore, matrix multiplication formula $\mathbf{Ax}$ can be written from the multiplication operation in $\mathbb{Z}_p[x]/\langle f(x)\rangle$, i.e.,

$$\mathbf{y} = \sum_{i=1}^{t} \mathbf{a}_i \cdot \mathbf{x}_i.$$

In this case, $m = tn$ and $(\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_t)$ is seen as key (with length $(m \lg p)$ bits) are t vector in $\mathbb{Z}_p^n$ chosen at random. $\mathbf{a}_i \cdot \mathbf{x}_i$ is a representation of a multiplication operation modulo $f(x)$, that is clear that the computational process becomes more efficient than before. This formula is defined as compression function formula of the hash function family based on ideal lattice.

With the aim of shortening the key and accelerating the computation process of a hash function family computing based on ideal lattice, we now define the compression function of the hash function defined recursively

$\mathbf{y}_0 = \mathbf{a}$, and for $k = 1, 2, ..., t$ calculate $\mathbf{y}_k = (\mathbf{y}_{k-1} \odot \mathbf{x}_i) \bmod \mathbf{f}_i$, then

$$\mathbf{y} = \sum_{i=1}^{t} \mathbf{y}_i.$$

In this case, $\mathbf{a}$ and $(\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_t)$ is seen as a key, $\mathbf{a}$ is chosen randomly and uniformly in $\mathbb{Z}_p^n$ and $(\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_t)$ were selected randomly and uniformly t ordered pairs in the set $\{-1, 1\} \times \{\pm 1, \pm 2, ..., \pm(n-1)\}$ and computing $(\mathbf{y}_{i-1} \odot \mathbf{x}_i) \bmod \mathbf{f}_i$ using Algorithm 2. More specifically, the compression procedure of the above formula is as follows.

1. Set positive integer parameters m, n and t with $m = nt$, also set the value of a prime p such that $m \approx 2n \lg p$.
2. Select randomly and uniformly key $\mathbf{a}$ and $(\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_t)$.
3. Take binary message $\mathbf{x} \in \mathbb{Z}_2^m$, divided into t blocks, namely: $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t)$, then $\mathbf{x}_k \in \mathbb{Z}_2^n$ for $k = 1, 2, ..., t$.
4. Calculate compression function value $\mathbf{y}$ of the input message $\mathbf{x}$ using the following algorithm.

Algorithm 3 (**Compression Function Algorithm**)

Input: Message block $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t)$, means $\mathbf{x}_k \in \mathbb{Z}_2^n$ for $k = 1, 2, ..., t$.

Output: The vector $\mathbf{y} \in \mathbb{Z}_p^n$.

1. Initialization $\mathbf{z} := \mathbf{a}$, $\mathbf{y} = \mathbf{0}$.
2. For integer $k = 1$ to $i = t$, calculate:
   (a) $\mathbf{z} := (\mathbf{z} \odot \mathbf{x}_k) \bmod \mathbf{f}_k$ call Algorithm 2.
   (b) $\mathbf{y} := \mathbf{y} + \mathbf{z}$.
3. **return**($\mathbf{y}$).

It is clear that the computing process above will define a family of compression functions $h : \mathbb{Z}_2^m \to \mathbb{Z}_p^n$ which means compressing binary message **x** with length *m* bits to **y** with length $(n \lg p) \approx \frac{m}{2}$ bits. Moreover, because the computing process $(\mathbf{z} \odot \mathbf{x}_k) \bmod \mathbf{f}_k$ defines a matrix multiplication $\mathbf{A}^{(k)}\mathbf{x}_k$, then the computational process of the compression function *h* also defines matrix multiplication $h(\mathbf{x}) = \mathbf{A}\mathbf{x}$ with

$$\mathbf{A} = (\mathbf{A}^{(1)} \quad \mathbf{A}^{(2)} \quad \cdots \quad \mathbf{A}^{(t)}) \in \mathbb{Z}_q^{n \times m}.$$

From the fact that Algorithm 2 contains $\frac{(n-1)n}{2}$ addition operations modulo *p*, it is clear that the process of computing the compression function *h* on average contains $\left(\frac{(n-1)n}{2}\right)t = \frac{(n-1)m}{2}$ additional operations modulo *p*, equivalent to $\left(\frac{(n-1)m}{2}\right)\lg p$ bits operation.

4. Security Aspects of Construction Result

Since $\mathbf{a} \in \mathbb{Z}_p^n$ and $\mathbf{f}_k \in \{-1, 1\} \times \{\pm 1, \pm 2, ..., \pm(n-1)\}$, the key length of the compression function *h* as the result of the construction is approximately $[(n \lg p) + t \lg 2n]$ bits. If the key is assumed to be chosen at random and uniform, then the chances of guessing the key is $2^{-[(n \lg p)+t \lg 2n]}$. As an illustration, with the value of the parameter $p = 257$, $n = 64$, and $t = 16$, then the key length is approximately 624 bits so it is not feasible to predict. With the same parameter values, the key size is much shorter than the key size of the compression function SWIFFT [8] which is about 8192 bits.

Based on the computational process, when the key is chosen randomly, then the compression function *h* as the result of construction also defines the matrix **A** which is also random. Consequently, *h* is a one-way function with the following explanation. To explain that *h* is a one-way function, we have

to show that given any value $\mathbf{x}$ from known value $\mathbf{y}$, the equation $\mathbf{Ax} = \mathbf{y}$ is a computational problem. Notice that the equation $\mathbf{Ax} = \mathbf{y}$ can be written as

$$(\mathbf{a}_1 \;\; \mathbf{a}_2 \;\; \cdots \;\; \mathbf{a}_m)\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \mathbf{y} \Leftrightarrow \sum_{j=1}^{m} x_j \mathbf{a}_j$$

$$= \mathbf{y} \Leftrightarrow \sum_{j=1}^{m} x_j \mathbf{a}_j + (-y)$$

$$= \mathbf{0} \Leftrightarrow (\mathbf{a}_1 \;\; \mathbf{a}_2 \;\; \cdots \;\; \mathbf{a}_m \;\; -\mathbf{y})\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \Leftrightarrow (\mathbf{A}|\mathbf{y})\tilde{\mathbf{x}} = \mathbf{0}.$$

This means that $\tilde{\mathbf{x}}$ is a vector in the orthogonal space $\Lambda^{\perp}$ of $\Lambda = \mathcal{L}(\mathbf{B})$ lattice generated by the basis $\mathbf{B} = (\mathbf{A}|\mathbf{y})^T$. Since $\tilde{\mathbf{x}}$ is a binary vector, given any values of $\mathbf{x}$ from the known value $\mathbf{y}$ means finding a basis for $\Lambda^{\perp}$ which the members are short vectors. This is the lattice computational problem.

Equally, the occurrence of collision of h means there exists $\mathbf{x}' \neq \mathbf{x}$ so

$$\mathbf{Ax}' = \mathbf{Ax} \Leftrightarrow \mathbf{A}(\mathbf{x}' - \mathbf{x}) = \mathbf{0}.$$

In other words, $(\mathbf{x}' - \mathbf{x}) \in \Lambda^{\perp}$, where $\Lambda = \mathcal{L}(\mathbf{B})$ with $\mathbf{B} = \mathbf{A}^T$. Since $\mathbf{x}$ and $\mathbf{x}'$ are binary vectors, $(\mathbf{x}' - \mathbf{x})$ is a short vector. Thus, determining the occurrence of collision of h means that finding a basis for $\Lambda^{\perp}$ which the members are short vectors. This is also the lattice computational problem.

Therefore, compression function *h* as the result of the construction is one-way function and collision resistance based on lattice computational problem.

References

[1] M. Ajtai, Generating hard instance of lattice problems, Complexity of Computations and Proofs, Vol. 3 of Quad. Math., pp. 1-32, Dept. Math. Sconda University Napoli, Caserta, 2004, Preliminary version in STOC, 1996.

[2] K. Bebtahar, D. Page, J. Silverman, M. Saarinen and N. Smart, LASH, Technical Report, 2nd NIST Cryptographic Hash Function Workshop, 2006.

[3] J. Y. Cai and A. Nerurkar, An improved worst-case to average-case connection for lattice problems, Proc. 38th IEEE Symp. on Foundations of Computer Sci. (FOCS), 1997, pp. 468-477.

[4] P. A. Fuhrmann, A Polynomial Approach to Linear Algebra, Springer-Verlag, 1996.

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

[6] S. Guritman, N. Aliatiningtyas, T. Wulandari and M. Ilyas, Aritmetik Ring Polinomial untuk Konstruksi Fungsi Hash Berbasis Latis Ideal, In J. Matematika dan Aplikasinya (JMA) 1(1) (2013).

[7] V. Lyubashevsky and D. Micciancio, Generalized compact knapsacks are collision resistant, 33rd International Colloquium on Automata, Languages and Programming (ICALP), 2006.

[8] V. Lubyashevsky, D. Micciancio, C. Peikert and A. Rosen, SWIFFT: modest proposal for FFT hashing, FSE 2008, 2008.

[9] R. P. Menezes, P. C. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, Inc., 1997.

[10] D. Micciancio, Improved crytographic hash functions with worst-case and average-case connections, Proc. 34th ACM Symp. on Theory of Computing (STOC), 2002, pp. 609-618.

[11] D. Micciancio and O. Regev, Worst-case to average-case reductions based on Gaussian measures, Proc. 45th Annual IEEE Symp. on foundations of Computer Sci. (FOCS), 2002, pp. 609-618.

[12] C. Peikert and A. Rosen, Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices, 3rd Theory of Cryptography Conference (TCC), 2006, pp. 145-166.

[13] C. C. Pinter, A Book of Abstract Algebra, McGraw-Hill Inc., 1990.

[14] Rachmawati Dwi Estuningsih, Sugi Guritman and Bib P. Silalahi, Algorithm construction of HLI hash function, Far East J. Math. Sci. (FJMS) 86(1) (2014), 23-36.