LTSC-128: Stream Cipher Based on the Intractable Shortest Vector Problem in Lattice

Khaled Suwais, Azman Samsudin
School of Computer Sciences
Universiti Sains Malaysia
Penang 11800
Malaysia
khaled@cs.usm.my, azman@cs.usm.my

Journal of Digital Information Management

ABSTRACT: *LTSC-128 is a high secure stream cipher based on the hardness of the Shortest Vector Problem in Lattice space. The cipher is based on vectors multiplication over finite field, where these vectors are represented by polynomials to enhance the performance of keystream generation. The key size is 128 bits and there is no attack faster than exhaustive key search has been identified. The experimental results showed that the encryption rate of LTSC-128 is about 50Mbit/second. LTSC-128 cipher has additional feature that its performance can be increased by applying multithreading technique on multi-core processors. The multithreaded version of LTSC-128 has achieved better encryption rate of 71Mbit/second on dual-core processor.*

Categories and Subject Descriptors
E.3 [**Data Encryption**] Data encryption standard; **D.4.6** [Security and Protection**]** Cryptographic controls

General Terms: Data encryption; Cryptographic analysis, Vector problem, Lattice

Keywords: Stream cipher, Shortest vector problem, Lattice, NP-hard problem, Multithreading, Encryption

Received: 11 November 2010; Revised 17 December 2010; Accepted 20 December 2010

1. Introduction

Stream cipher algorithm is responsible for generating a cryptographically strong keystream to perform encryption operations on a given plaintext. Generally, stream cipher is used in environment where encryption need to be done in a continuous stream of bits (rather than by block as in block cipher). Stream ciphers can be categorized into two categories: hardware-based and software-based. The hardware-based stream ciphers are commonly based on the Linear Feedback Shift Registers (LFSRs) (e.g. SNOW (Ekdahl and Johansson 2000), LILI-128 (Dawson et al. 2000), etc.). On the other hand, software-based stream ciphers (e.g. RC4 (Rivest 1992), SEAL (Rogaway and Coppersmith 1998), etc.) rely on Boolean functions and bit manipulation in their computations. A cryptographically secure stream cipher should satisfy two essential requirements. These requirements are summarized by two questions: how secure is the generated keystream against cryptanalysis attacks and how strong is the generated keystream statistically? In addition, one should also take into consideration the performance factor so that the performance and the level of security provided by the algorithm are balanced.

Both categories of stream ciphers are proven to provide high performance ciphers according to the results presented in (Rogaway and Coppersmith 1998, Boesgaard et al. 2003). However, researches show that those ciphers either do not reach the required level of security or that the generated keystream is statistically weak (Scott et al. 2001,Christophe 2001). Therefore the current design of stream ciphers, which rests on using simple logical and mathematical operations or LFSR, show the imperfection of the stream cipher.

The core of any stream cipher is presented by the keystream generation procedure, where the input key is treated based on specific and pre-defined formula of that cipher. In this paper we are presenting a new stream cipher based on the hardness of the Shortest Vector Problem (SVP) in lattice system. SVP is an NP-hard problem (Khot 2005) and it is the core of the proposed cipher to ensure higher security compared to the existing stream ciphers discussed earlier.

The SVP along with three other intractable problems are considered as the four most important pivots in current public-key cryptosystems as appeared in the literature. The other three problems are the Integer Factorization Problem (IFP), Discrete Logarithm Problem (DLP) and Elliptic Curve Discrete Logarithm Problem (ECDLP). The first use of SVP was formally introduced in 1998 (Hoffstein et al. 1998), in a public-key cryptosystem called NTRU cryptosystem. The general idea of SVP rests on finding the shortest non-zero vector b in lattice $\boldsymbol{B}$, where $\boldsymbol{B}$ is the basis of the lattice δ.

The basis $\boldsymbol{B}$ can be seen as $n \times m$ matrix, where vectors b forms the column of $\boldsymbol{B}$ such as:

$$\boldsymbol{B} = \begin{pmatrix} | & \cdots & | \\ \overrightarrow{b_1} & \cdots & \overrightarrow{b_m} \\ | & \cdots & | \end{pmatrix} \quad (1)$$

The workflow of LTSC-128 is divided into three phases: Initialization Phase (IP), Keystream Generation Phase (KGP) and Encryption Phase (EP) (see Section 3). The SVP is applied in the second phase of LTSC-128 to generate the keystream. In LTSC-128, the mathematical operations and vector representation of SVP in lattice space was inspired by NTRU public-key cryptosystem, where the vectors are represented by polynomials for efficiency purposes as will discuss later in this document.

Employing NP-hard problem such as SVP in stream cipher will strengthen the security of the generated keystream against cryptanalysis attacks, since there is no known method that can solve NP-hard problems in polynomial time. LTSC-128 provides higher security than several other well known ciphers intended to be used in hardware and software implementations.

From a technical point of view, the use of NP-hard problems in cryptographic primitives decreases the performance of the corresponding primitives due to its costly operations. Thus, the use of the new multi-core technology becomes necessary in

order to enhance the performance cryptographic primitives based on NP-hard problems. Technically, gaining higher performance on multi-core platform can be achieved by using multithreading techniques, where multiple threads are created to accomplish the internal calculations of the corresponding cryptographic primitives. From another perspective, multithreading techniques can be applied on the structural design of the algorithm (e.g. stream cipher) such that multiple keystreams are generated and used to encrypt a stream of plaintext as described by the model proposed in (Suwais and Samsudin 2008).

The paper is organized as follow: Section 2 provides the background and notations used. Section 3 describes the structure of the proposed algorithm. Implementation and performance evaluation is discussed in Section 4. Security evaluation and statisitcal analysis are presented in Section 5 and 6 respectively. Finally, concluding remarks are presented in Sections 7.

2. Definitions and Preliminaries

Lattice δ is the set of linear independent vectors $\overrightarrow{b_i}$ defined as:

$$\delta = \sum_{i=1}^{n} r_i \overrightarrow{b_i}, r_i \in Z \quad (2)$$

The matrix $\boldsymbol{B}$ is known as the basis of lattice δ and it consists of all vectors $\overrightarrow{b_i}$ (see Eq.(1)). The dimension of δ is the number of vectors in $\boldsymbol{B}$. The vectors in lattice can be represented as truncated polynomial $\overrightarrow{\rho}$ of degree n-1 as shown in Eq.(3):

$$\overrightarrow{\rho}(x) = a_n x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_o \quad (3)$$

where $a_0, a_1, \cdots a_n$ are binary coefficients in $\{0,1\}$ and all the operations are defined over the ring $R = Z[X]/(X^n-1)$. The sum of two polynomials is computed by adding the coefficients of variables with the same powers. One important operation that is needed during the keystream generation is the Convolution Multiplication of polynomials in $\boldsymbol{B}$. The convolution multiplication of polynomials in $\boldsymbol{B}$ is denoted by $\otimes$ and defined as follows:

$$\overrightarrow{\boldsymbol{L}} \otimes \overrightarrow{\boldsymbol{M}} = \sum_{i=0}^{k} \overrightarrow{\boldsymbol{L}_i}\overrightarrow{\boldsymbol{M}}_{k-1} + \sum_{i=k+1}^{n-1} \overrightarrow{\boldsymbol{L}_i}\overrightarrow{\boldsymbol{M}}_{n+k-1} = \sum_{i+j \equiv 1 (mod\ k)} \overrightarrow{\boldsymbol{L}_i}\overrightarrow{\boldsymbol{M}_i} \quad (4)$$

where k and i denotes the poisitions of coefficients in polynomials $\boldsymbol{M}$ and $\boldsymbol{L}$ respectively. The convolution product of two polynomials is treated as coefficients multiplication, where a polynomial is represented by a binary string of coefficients $[a_n, a_{n-1}, \cdots, a_1, a_0]$. For example, let $\overrightarrow{\boldsymbol{M}}$ and $\overrightarrow{\boldsymbol{L}}$ be two truncated polynomials of degree 8 with binary coefficients as the following:

$$\overrightarrow{\boldsymbol{M}}(x) = x^7 + x^6 + x^2 + 1 \quad (5)$$

$$\overrightarrow{\boldsymbol{L}}(x) = x^7 + x^5 + x^4 + 1 \quad (6)$$

Then $\overrightarrow{\boldsymbol{M}}$ and $\overrightarrow{\boldsymbol{L}}$ can be represented as two vectors as follow:

$$\overrightarrow{\boldsymbol{M}} = [1,1,0,0,0,1,0,1] \quad (7)$$

$$\overrightarrow{\boldsymbol{L}} = [1,0,1,1,0,0,0,1] \quad (8)$$

To illustrate the idea of convolution product of two polynomials, let $[k_1, k_2, k_3]$ and $[u_1, u_2, u_3]$ be two polynomials, then the convolution multiplication is performed as follows:

$$[k_1, k_2, k_3] \otimes [u_1, u_2, u_3] = k_1 \cdot [u_1, u_2, u_3] + k_2 \cdot [u_3, u_1, u_2] + k_3 \cdot [u_2, u_3, u_1]$$
$$= [k_1u_1,\ k_1u_2,\ k_1u_3] + [k_2u_3,\ k_2u_1,\ k_2u_2] + [k_3u_2,\ k_3u_3,\ k_3u_1]$$
$$= [k_1u_1 + k_2u_3 + k_3u_2,\ k_1u_2 + k_2u_1 + k_3u_3,\ k_1u_3 + k_2u_2 + k_3u_1]$$

Resulted polynomial from the convolution multiplication or any other operations are reduced modulo q, where q is a pre-defined value. The other important operation that is being carried out in our implementation is the inverse of a given polynomial. The inverse of polynomial $\overrightarrow{\boldsymbol{M}}$ modulo q is denoted by $\overrightarrow{\boldsymbol{M}}_q^{-1}$ such that:

$$\overrightarrow{\boldsymbol{M}}_q^{-1} \otimes \overrightarrow{\boldsymbol{M}} \equiv 1\ (mod\ q) \quad (9)$$

For a given polynomial $\overrightarrow{\boldsymbol{M}}$, we write ϑ to denote the number of one's coefficients in the polynomial. In practice, the convolution product of two polynomials with binary coefficients requires ϑ_n operations. Thus, for further speed optimization we apply the concept of Small Hamming Weight Polynomial multiplication (Hoffstein and Silverman 2000) by dividing the polynomials used in our system into smaller polynomial with less number of ϑ coefficients. Therefore, the computation of the multiplication is defined as:

$$\overrightarrow{\boldsymbol{L}}(X) \times \overrightarrow{\boldsymbol{M}}(X) = \overrightarrow{\boldsymbol{L}}_1(X) \times (\overrightarrow{\boldsymbol{L}}_2(X) \times \overrightarrow{\boldsymbol{M}}(X)) \quad (10)$$

where $\overrightarrow{\boldsymbol{L}}_1$(X) and $\overrightarrow{\boldsymbol{L}}_2(X)$ are obtained by splitting $\overrightarrow{\boldsymbol{L}}(X)$ with fewer number of ones in ϑ_1 and ϑ_2 for $\overrightarrow{\boldsymbol{L}}_1$ and $\overrightarrow{\boldsymbol{L}}_2$ respectively. Therefore the multiplication where and are obtained by splitting with fewer number and of smaller polynomials requires $(\vartheta_1 + \vartheta_2)n$ operations, where n is the degree of the polynomial. Such polynomial multiplication can be implemented in computer programs as a set of shift and XOR operations, which efficiently affect the performance of the polynomial multiplication process.

3. Proposed Stream Cipher LTSC-128

The use of SVP of high dimensional lattice as the hard problem of LTSC-128 is promising since there is no algorithm that is able to solve the SVP problem in polynomial time. Moreover, this problem has been implemented in public-key cryptography (e.g. NTRU cryptosystem) and has shown good results in terms of performance and security. LTSC-128 implements lattice and vectors in polynomial representation for higher efficiency as discussed in Section 2. Low Hamming Weight polynomial multiplication concept is applied by dividing the polynomial of degree n into smaller polynomials with fewer one's ($\vartheta = \frac{n}{3}$).

LTSC-128 works as follows: an input key of 128-bit generates a keystream of 128-bit length per round. The keystream is divided into 4-words (sub-keystream). These sub-keystream bits are XOR'ed with the plaintext bits (one word of plaintext at a time) to generate the ciphertext bits. LTSC-128 is designed in three phases: IP, KGP and EP as described in the following sections.

3.1 Initialization Phase (IP)

At this phase, we aim to extract two polynomials, $\overrightarrow{\boldsymbol{M}}$ and $\overrightarrow{\boldsymbol{L}}$ in order to use them in the keystream generation phase. The two polynomials are extracted from the input key $\boldsymbol{K}$ of 128-bit length. The key $\boldsymbol{K}$ is transformed to its binary representation, and from those binary bits we obtain the two polynomials of degree n = 347 (the value of n is chosen to achieve higher security as proved in Section 5, such that $(n-1)/2$ is prime). Let ϑ_M and ϑ_L be the number of 1's coefficients of polynomials $\overrightarrow{\boldsymbol{M}}$ and $\overrightarrow{\boldsymbol{L}}$ respectively, such that $\vartheta_M = \vartheta_L = n/3 = 116$.

The extraction of polynomial $\overrightarrow{\boldsymbol{L}}$ (setting the coefficients of $\overrightarrow{\boldsymbol{L}}$) is done by selecting the 128-bit of the input key from the Least Significant Bit (LSB) position to the Most Significant Bit (MSB) position, such that the first selected bit from $\boldsymbol{K}$ is the first coefficient of $\overrightarrow{\boldsymbol{L}}$ until we obtain $\vartheta_L = 116$. If ϑ_L is less than 116, which is the

common case, we set the last coefficients of $\vec{L}$ by selecting r bits of K from MSB position to LSB position until we satisfy the condition $\vartheta_L = 116$. In rare situation, the majority of the input key bits are zeros with fewer ones, which means the input key is not capable to provide polynomial $\vec{L}$ with sufficient number of ones. In that case, we reset $(0 \text{ to } 1)$ the bits of polynomial $\vec{L}$ starting from the location $\frac{n}{2}$ moving toward the LSB of $\vec{L}$. This procedure continues until we hit the threshold $\vartheta_L = 116$. The purpose of this procedure is to achieve preferable distribution of ones coefficients in the new generated polynomial.

On the other hand, the coefficients of the second polynomial $\vec{M}$ are also extracted from the input key by using similar mechanism. We set the first coefficients by selecting the 128-bits of the input key from MSB position to LSB position, such that the first selected bit from K is the first coefficient of $\vec{M}$, until we obtain $\vartheta_M = 116$. If ϑ_M is less than 116, we set the last coefficients of $\vec{M}$ by selecting r bits of K from LSB position to MSB position until we satisfy the condition $\vartheta_M = 116$. Similar to the rare case discussed in generating polynomial $\vec{L}$, we need to reset some bits in $\vec{M}$ to hit the threshold $\vartheta_M = 116$. However, the procedure of resetting the zero bits in $\vec{M}$ starts from the location $\frac{n}{2}$, moving toward the

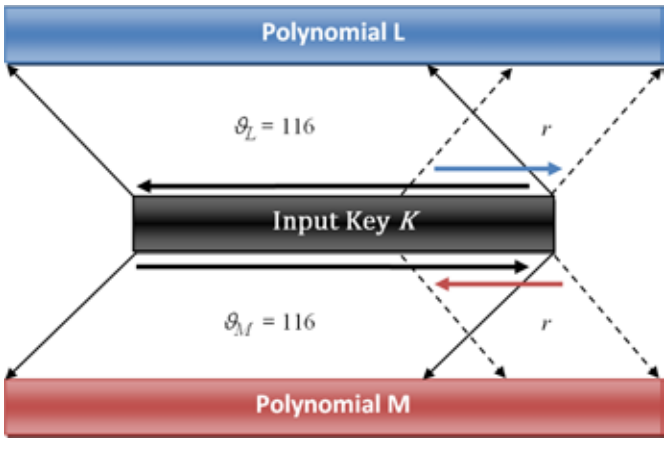


Figure 1. Initializing polynomials $\vec{L}$ and $\vec{M}$ by extracting the coefficients from K

MSB of $\vec{M}$. Figure 1 illustrates the extraction of the initialization value for polynomials $\vec{L}$ and $\vec{M}$ from the input key.

Finally, another two parameters $(\vec{P}, q)$ are needed during the KGP phase to generate the keystreams. As proved in (Hoffstein and Silverman 2000), Polynomial $\vec{P} = x + 2$ and integer $q = 128$ are two ideal units in $Z[X]$. The generated polynomial $\vec{M}$ will be in the form $\vec{M} = 1 + (2 + x) \times \vec{M}$.

3.2 Keystream Generation Phase (KGP)

The generated and initialized polynomials and variables in IP phase, form the main parameters for the KGP phase. The formula for a single round of keystream K_s generation is given as follows:

$$K_s = \vec{P} \otimes \vec{M}_q \otimes \vec{L} \quad (11)$$

where $\vec{P}$, $\vec{M}$ and $\vec{L}$ are extracted and initialized in the previous phase, and $\vec{M}_q$ is the inverse polynomial $(\vec{M} \ mod \ q)$ (see Eq.(9)).

In order to generate new keystream, KGP uses a counter C_k, which is firstly initialized from the input key K by Eq.(12):

$$C_k = \sum_{i=1}^{n} (K[i](mod\ n)) \quad (12)$$

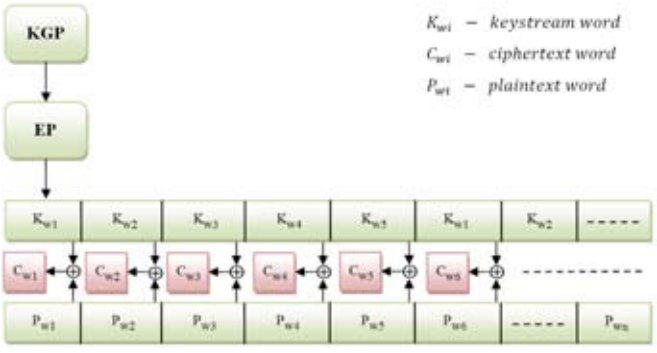


Figure 2. The design of EP Phase implemented in LTSC-128

When the KGP is required to generate a new keystream for encryption, counter C_k is incremented and added (addition modulo q) to $\vec{L}$ and $\vec{M}$. At this point, another auxiliary counter $(C_d)$is required in generating new keystream. The counter $C_d$is firstly initialized such that $C_d = C_k$. The two counters C_k and C_d are then used by Eq.(13) for one round of keystream generation (K_{si})as follows:

$$K_{si} = [\vec{P} \otimes (\vec{M} \oplus C_{k_{i+1}})_q^{-1} \otimes (\vec{L} \oplus C_{d_{i+1}})](mod\ q) \quad (13)$$

where $C_{ki+1} = \vec{M} \oplus [C_{ki} \ll LCK_5]+1$and $C_{di+1} = \vec{L} \oplus [C_{di} \ll LCD_5]+1$. The two variables LCK_5 and LCD_5 are the integer values represented by the last 5 bits of both C_k and C_d respectively. The size of the resulted K_s is 347-bit block. The first 340-bits are divided into 4 chunks of 85-bit block. Only the second and third segments (Seg2 and Seg3) are considered in the final keystream since they showed better statistical properties compared to the rest of the bits. The rest of the bits (Seg1 and Seg4) are found statistically weak, and therefore they are not considered in the final keystream.

3.3 Encryption Phase (EP)

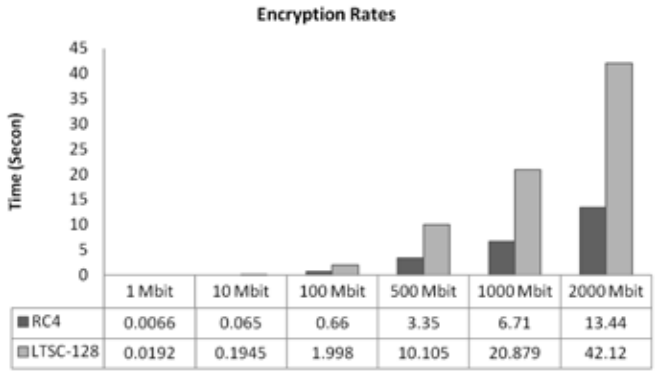
The encryption process of LTSC-128 is based on applying exclusive-OR (XOR) operation between the keystream bits and the plaintext bits. The generated K_s of 160-bit length is divided into five words (32-bit each). The encryption is accomplished by XOR'ing one word of plaintext with one word of the keystream as portrayed by Figure 2.

Notice that when EP has used the whole generated bits at time t_i, the KGP will be called to generate new keystream as highlighted in the KGP phase.

4. Implementation and Performance Evaluation

We compare the performance of LTSC-128 against the widely adapted RC4 stream cipher, which is based on simple substitutions and shift operations. Both ciphers were coded in C++ using MinGW-2.05. NTL library is used to handle the polynomial arithmetic in LTSC-128. For testing purposes, we ran both ciphers on a workstation with Intel Core 2 Duo CPU 2.13 GHz processor, with 2GB RAM.

In terms of performance, single threaded LTSC-128 is about three times slower than RC4. Relatively, LTSC-128 is a good alternative with regards to the high security provided compared to the broken RC4. The encryption rate of LTSC-128 is 49.671 MBits/second compared to the RC4 encryption rate which is 152.173 MBits/second on the same machine. Figure 3 illustrates the performance comparison between LTSC-128 and RC4 with different plaintext sizes.



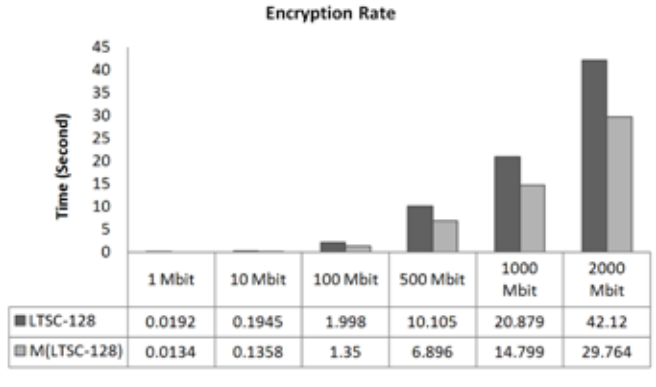
	1 Mbit	10 Mbit	100 Mbit	500 Mbit	1000 Mbit	2000 Mbit
RC4	0.0066	0.065	0.66	3.35	6.71	13.44
LTSC-128	0.0192	0.1945	1.998	10.105	20.879	42.12

Figure 3. Encryption rate for RC4 and LTSC-128 stream ciphers

Recently, a high performance multithreaded model for stream ciphers was proposed in (Suwais and Samsudin 2008) to enhance the performance of stream ciphers, where multiple tasks are performed concurrently. The model is divided into three components: thread creator, keystream generator and plaintext encoder.

Applying the multithreaded model on LTSC-128 requires immediate trigger of the thread creator component of the multithreaded model, resulting in four threads. Two threads are responsible of generating keystreams, while the other two threads are responsible of encrypting stream of plaintext. Subsequently, the four threads will be directly connected to the keystream generation and encryption phases of LTSC-128, via the keystream generator and plaintext encoder components of the multithreaded model. However, LTSC-128 uses single counter to generate new keystream each round. But the multithreaded model is designed to use two counters in parallel.

The multithreaded model helps LTSC-128 to perform faster since the workload of LTSC-128 is divided among multiple threads, and all those threads are operating concurrently on multi-core processors. Thus multithreading on multi-core machine provides LTSC-128 with higher utilization on the new generation of processors. Applying this model on LTSC-128 shows better performance compared to the results discussed above. Figure 4 illustrates the performance enhancement obtained by applying the multithreaded model on a dual core machine. The encryption rate of the multithreaded LTSC-128 has significantly increase to 71.602 Mbit/s compared to the sequential LTSC-128 encryption rate, 49.671 Mbit/s, as shown previously.



	1 Mbit	10 Mbit	100 Mbit	500 Mbit	1000 Mbit	2000 Mbit
LTSC-128	0.0192	0.1945	1.998	10.105	20.879	42.12
M(LTSC-128)	0.0134	0.1358	1.35	6.896	14.799	29.764

Figure 4. Performance Improvement gained from applying the multithreaded model on M(LTSC-128)

5. Security Analysis

The security of LTSC-128 rests on the intractability of the SVP problem, which is an NP-hard problem. At present, no algorithm can solve the SVP in polynomial time. Besides NP-hard problem, the security of LTSC-128 is also being measured by its statistical property. In this section, we examine LTSC-128 against cryptanalaysis attacks and test the statistical attributes of the generated keystream from the KGP phase.

5.1 Brute Force Attack

The length of the input key used in LTSC-128 was chosen to specifically thwart Brute Force attack. Brute Force attack on the 128-bit input key of LTSC-128 is infeasible due to the huge key space, 2^{128} possible keys. Hypothetically, an attacker can recover the secret key by trying all possible polynomials $\vec{M} \in \zeta_M$ or $\vec{L} \in \zeta_L$ (where ζ denotes polynomial space), and test if $\vec{P} \otimes \vec{M}_q \otimes \vec{L} \ (mod\ q)$ has small entries (polynomial with less number of one's coefficients). Since $\vec{L}$ and $\vec{M}$ are of the same order and equal in the number of non-zero coefficients, we consider only polynomial $\vec{M}$ in our analysis. The security of SVP is determined by $\#\zeta_M$. The security level (Hoffstein et al. 1998) for SVP of dimension n is then given by Eq.(14):

$$\sqrt{\#\zeta_M} = \frac{1}{\vartheta_M!}\sqrt{\frac{n!}{(n-2\vartheta_M)!}} \tag{14}$$

Based on Eq.(14), it requires the attacker to carry out approximately 2^{132} operations to brute force LTSC-128. The same procedure has to be applied on polynomial $\vec{L}$ to attack our generator, which double the calculation requirements. Therefore, LTSC-128 is secure against brute-force attack due to the infeasibility of carrying out huge amount of calculations on both polynomials $\vec{M}$ and $\vec{L}$.

5.2 Time-memory Trade-off Attack

The sucess of this kind of attacks depnds on the size of the state space. Ciphers with small state space are highly subjected to this attack. Theprocedure of this attack is carried out as follows. Create a sorted list of output sequences and their corresponding states. Then, the received output sequence is matched with the one stored in the list. If matching is found, the cipher state before producing the output sequence can also be found in the list. Resulting in a successful key recovery (Ekdahl and Johansson 2000).

We found that time-memory trade-off attack is not feasible to be applied on LTSC-128 stream cipher. The state space is found to be too larg ($2^{347+128}$) compared to the key size of the space(2^{128}).

5.3 Balance

The initialization phase of LTSC-128 intends to extract two polynomials $\vec{M}$ and $\vec{L}$ from the input key K. We found it hard to recover the input key by attacking polynomials $\vec{M}$ and $\vec{L}$ due to the balance property that we added to this phase. The extracted polynomials have equal number of 0's and 1's ($\vartheta_M = \vartheta_L = n/3\ 116$). The initialization phase has eliminated the statistical biased that could be found during the process of polynomials extraction. From the other perspective, attacker is required to perform $2^{347}+2^{347}$ operations to recover the input key, based on the fact that polynomials $\vec{M}$ and $\vec{L}$ are of the degree 347. Therefore, this kind of attacks is infeasible due to the huge number of operations which are required to recover the input key.

5.4 Chosen Input Differential Attack

This is yet another powerfull attack where small changes on the input values is expected to propagate through the cipher.

In LTSC-128, changing one bit in the input key will affect the entire six variables in the keystream generator.

These variables are the counters (C_k, C_d), two polynomials ($\vec{M}$, $\vec{L}$) of the degree 347 and two shifting variables (LCK_5, LCD_5). LTSC-128 is found secure against this kind of attacks since changing one bit of the input key will affect the entire variables of the sequence generation cycle. The resulting sequence which is mapped from the input sequence is obtained in a nonlinear fashion by performing bit-shifting, modulo operation and convolution multiplications. In other words, the $i-th$ bit of the output sequence depends on the first, second, third,..., and $j-th$ bit of the input (where $0 \leq i \leq j$).

5.5 LLL Lattice Basis Reduction Algorithm

Lenstra et al. [1982] described a polynomial time algorithm called LLL-reduction algorithm, for transforming basis $\boldsymbol{B} = (b_1, b_2, ..., b_n)$ of Lattice $\vec{L}$ into an LLL-reduced basis $\grave{\boldsymbol{B}} = (\grave{b}_1, \grave{b}_2, ..., \grave{b}_n)$. The LLL-reduction algorithm is commonly used as cryptanalysis tool to attack lattice-based primitives, since it can find a relatively shorter vector in the reduced lattice basis. The new basis $\grave{\boldsymbol{B}}$ is considered LLL-reduced if it satisfies the following two properties (Cohen 1993):

$$|\mu_{i,j}| \leq \frac{1}{2} \qquad for\ i \ \leq \ j < i \leq \ n \tag{15}$$

$$|b_i^*|^2 \geq (\frac{3}{4} - \mu_{i,i-1}^2)|b_{i-1}^*|^2 \tag{16}$$

where $\mu_{i,j} = \frac{\langle b_i, b_j' \rangle}{\langle b_j', b_j' \rangle}$, and $b_i^* = (b_1^*, b_2^*, \ldots, b_n^*)$ is the Gram-Schmidt orthogonal b,binjjized basis generated from $\boldsymbol{B}$ (Weisstein 2008).

The LLL-reduction algorithm converts lattice basis $\boldsymbol{B}$ into $\grave{\boldsymbol{B}}$ in two steps transformation. The first step is called size-reduction that aims to perform the transformation using formula in Eq.(17) (LaMacchia 1999):

$$b_i \leftarrow b_i - \lceil \mu_{i,j} \rceil b_j \tag{17}$$

where $\lceil \ \rceil$ denotes the ceiling function. This step is followed by another step called the exchange-step, in which LLL searches for the smallest value of i such that b_i and b_{i-1} are not satisfying Eq.(17). Accordingly, LLL swaps the two vectors as shown in Eq.(18) :

$$b_{i-1} \leftarrow b_i \ , \ b_i \leftarrow b_{i-1} \tag{18}$$

The exchange-step is important to force obedience of LLL-reduction condition (Eq.(16)). LLL-reduction algorithm alternately performs the first and the second steps until both Eq.(15) and Eq.(16) are satisfied. The keystream generation of LTSC-128 is similar to the key generation of NTRU, where the generation is based on the hardness of SVP. The parameter set ($\boldsymbol{n} = 107, \boldsymbol{q} = 64$) of NTRU was not expected to be very secure when it was first designed, but it was clarified in (Gama and Nguyen 2008) that no NTRU-107 lattice has ever been broken by lattice reduction. Therefore, higher lattice dimension will logically increase the difficulty of breaking LTSC-128 lattice.

6. Analysis

It is important to statistically test the keystream sequence of bits, to determine the randomness of the generated keystream sequences. One way to test LTSC128 statistically is by carrying out the Diehard Tests Suite. Diehard Tests Suit was first presented by Marsaglia (Marsaglia 2003) and it consists of 18 core tests. These 18 tests are producing a total of 215 *p*-values uniform on $[0,1)$. The Diehard Suite requires approximately 68 million 32-bit numbers for a successful run of the 18 tests. The tests included in the Suite, with their abbreviation are: Birthday Spacing Test (BDAY), Overlapping 5-Permutation Test (OPERM), Binary Rank Test for matrices (RANK31), Binary Rank Test for matrices (RANK32), Binary Rank Test for matrices (RANK68), Bitstream Test (BSTM), DNA Test (DNA), OPSO Test (OPSO), OQSO Test (OQSO), Count-The-1's Test on stream of bytes (CBYTE), Count-The-1's Test for specific bytes (CSBYTE), Parking Lot Test (PARKL), Minimum Distance Test (MDIST), 3Dspheres Test (3DS), Squeeze Test (SQEZ), Overlapping Sums Test (OSUM), Run Test (RUN) and Crap Test (CRAP). Results obtained from the 18 tests are summarized in Table 1. The p-value of each test shows that LTSC-128 passed the tests with confidence of 99%, within the success range $0.01 < p-values < 0.99$.

Test Name	p-value	Conclusion
BDAY	0.5302	Success
OPERM	0.3686	Success
RANK31	0.4348	Success
RANK32	0.3297	Success
RANK68	0.4778	Success
BSTM	0.4650	Success
DNA	0.4998	Success
OPSO	0.5589	Success
OQSO	0.3079	Success
CBYTE	0.7048	Success
CSBYTE	0.5496	Success
PARKL	0.5886	Success
3DS	0.4921	Success
SQEZ	0.8739	Success
OSUM	0.5772	Success
RUN	0.3186	Success
CRAP	0.7995	Success
MDIST	0.4522	Success

Table 1. p-values and conclusion for Diehard test on LTSC-128

The distribution of the overall *p*-value from the 18 Diehard tests as shown in Table 2, indicates that the output of LTSC-128 is uniform on [0, 1). The distribution of the *p*-values is considered uniform on [0, 1), since the *p*-values are almost equally distributed among the 10 sub-ranges (from 0.0 to 1.0). Therefore it is safe to conclude that, LTSC-128 is suitable for the use of cryptographic primitive.

p-values Range	Observed Percent (%)	Expected Percent (%)
0.0− 0.1	11	10
0.1− 0.2	9	10
0.2− 0.3	7	10
0.3− 0.4	12	10
0.4− 0.5	13	10
0.5− 0.6	7	10
0.6− 0.7	13	10
0.7− 0.8	12	10
0.8− 0.9	8	10
0.9− 1.0	8	10

Table 2. The distribution of all Diehard p-values for LTSC-128

7. Conclusions

LTSC-128 stream cipher based on NP-hard problem is proposed. The structure of the algorithm is divided into three phases: IP, KGP and EP. The core of the LTSC-128 is the KGP which is based on the Shortest Vector Problem (SVP) in lattice space. In terms of security, LTSC-128 is highly secure since there is no algorithm yet that can solve the SVP in polynomial time. Statistically, LTSC-128 passes the Diehard Test Suite with a uniformed distribution of the produced p-values. The performance of sequential LTSC-128 is approximately three times slower than RC4, but due to its multithreaded design, LTSC-128 showed better performance when run on multi-core machine.

8. Acknowledgment

The authors would like to express their thanks to Universiti Sains Malaysia for supporting this study which is funded by the Malaysian Ministry of Science, Technology and Innovation via its eScience Fund research grant.

References

[1] Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J., Scavenius, O. (2003). Rabbit: A New High-Performance Stream Cipher, *In: Proc. of Fast Software Encryption*, LNCS-2887: 307-329.

[2] Christophe, C.(2001). Guess and Determine Attack on SNOW, NESSIE Public Report NES/DOC/KUL/WP5/011/a. http://www.cryptonessie.org.

[3] Cohen, H. (1993). A Course in Computational Algebraic Number Theory, Springer-Verlag, New York.

[4] Dawson, E., Clark, A., Golic, J., Millan, M., Penna, L., Simpson, L. (2000), The LILI-128 Keystream Generator, *in Proc. of First NESSIE Workshop*, Heverlee,Belgium. http://www.cryptonessie.org.

[5] Ekdahl, P., Johansson, T. (2000). SNOW-A New Stream Cipher, *In: Proc. of First NESSIE Workshop*, Heverlee, Belgium p.47-61.

[6] Gama, N., Nguyen, P. (2008). Predicting Lattice Reduction, *In: Proc. of EUROCRYPT 2008*, LNCS-4965:31-51.

[7] Hoffstein, J., Pipher, J., Silverman, J. (1998). NTRU: A Ring Based Public Key Cryptosystem, *In Algorithmic Number Theory: Third International Symposium (ANTS 3)*. LNCS-1423: 267-288.

[8] Hoffstein, J., Silverman, J. (2000). Optimizations for NTRU. *In: Proc. of Public Key Cryptography and Computational Number Theory*. de Gruyter,Warsaw.

[9] Khot, S. (2005). Hardness of Approximating the Shortest Vector Problem. *Journal of the ACM*, 52. 789-808.

[10] LaMacchia, B. (1999). Lattice Basis Reduction Algorithms, Farcaster, http://www.farcaster.com/papers/sm-thesis/node7.html.

[11] Lenstra, A., Lenstra, H., Lovasz, L.(1982). Factoring polynomials with rational coefficients, *Mathematische Annalen*, 261. 515-534.

[12] Marsaglia, G.(2003). DIEHARD: A Battery of Tests of Randomness, The Marsaglia Random Number CDROM http://stat.fsu.edu/pub/diehard/.

[13] Rivest, R.(1992). The RC4 Encryption Algorithm, RSA Data Security Inc., Document No, 003-013005-100-000000.

[14] Rogaway, P., Coppersmith, D.(1998). A Software-Optimized Encryption Algorithm, *Journal of Cryptology*. 11.273-287.

[15] Scott, R., Mantin, I., Shamir, A.(2001). Weaknesses in the Key Scheduling Algorithm of RC4. *Selected Areas in Cryptography*, LNCS-2259. 1-24.

[16] Suwais, K., Samsudin, A.(2008). High Performance Multithreaded Model for Stream Cipher, *International Journal of Computer Science and Network Security*, 8. 228-233.

[17] Weisstein W.(2008). Gram-Schmidt Orthonormalization, MathWorld-A Wolfram Web Resource. http://mathworld.wolfram.com/ Gram-SchmidtOrthonormalization.html.