The Taros Block Cipher

Phongphun Kijsanayothin and Somnuk Keretho
Department of Computer Engineering, Kasetsart University 50 Phaholyothin Rd. Bangkok 10903
kphongph@lycos.com sk@ku.ac.th

ABSTRACT

We propose a new block cipher called *Taros*. *Taros* has a block size of 128 bits and allows a variable-length key up to 1024 bits whereas the Square block can accept key only 128 bits. The cipher is a 6-round non-Feistel network. The proposed cipher is modified from the Square block cipher. The design of Taros concentrates on the resistance against Square attack cryptanalysis.

KEY WORDS

Block Cipher, Cryptanalysis

1. Introduction

In this paper we propose a block cipher called Taros. The types of building blocks and their interactions have been carefully chosen to allow efficient implementations on a wide range of processors and platforms. Firstly we provide a basic concept needed for understanding the paper. Subsequently, we present structure of Taros. The design rationale of the cipher and its inverse are treated. Afterward, the motivation of design choices and the treatment of the resistance against known types of attacks will be described. With the proposed algorithm, we provide security claims, goals and its advantages.

2. Mathematical Preliminaries

The operations in Taros are defined in byte level representing elements in a finite field $GF(2^8)$, Galois Field. In this section we introduce the basic mathematical concepts necessary for understanding this paper.

The elements of the finite field can be represented in several different ways. For any prime power there is a single field, hence all representations of $GF(2^8)$ are isomorphic. Despite this equivalence, the representation has an impact on the implementation complexity.

A byte b, consisting of bits $b_7\, b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0$, is considered as a polynomial representation with coefficient in {0,1} :

$$b_7\boldsymbol{x}^7 + b_6\boldsymbol{x}^6 + b_5\boldsymbol{x}^5 + b_4\boldsymbol{x}^4 + b_3\boldsymbol{x}^3 + b_2\boldsymbol{x}^2 + b_1\boldsymbol{x} + b_0$$

In addition, the sum of two elements is the polynomial with coefficients that are given by the sum modulo 2 of the coefficients of the two terms.

Example: '0x65'+'0x43'='0x27' (which + means exclusive-or), or:

$$(x^6+x^5+x^2+1) + (x^6+x^1+1) = x^5+x^2+x^1$$

In multiplication, multiplication in $GF(2^8)$ corresponds with multiplication of polynomials modulo an irreducible binary polynomial of degree 8. A polynomial is irreducible if it has no divisors other than 1 and itself.

Example: '0x65'*'0x43'='0x07'

(which * means multiplication) define irreducible binary polynomial equals '11B' ($x^8+x^4+x^3+x^1+1$)

$$(x^6+x^5+x^2+1)*(x^6+x^1+1)$$
$$=x^{12}+x^7+x^6+x^{11}+x^6+x^5+x^8+x^3+x^2+x^6+x^1+1$$
$$=x^{12}+x^{11}+x^8+x^6+x^5+x^3+x^2+x^1+1$$
$$=x^{12}+x^{11}+x^8+x^6+x^5+x^3+x^2+x^1+1 \text{ modulo } x^8+x^4+x^3+x^1+1$$
$$=x^2+x^1+1$$

3. Design rationale

The three criteria taken into account in the design of Taros are as the following:

- Resistance against all known attacks (Square Attack as attacked to AES).
- Speed and code compactness on a wide range of platforms (No affect of byte orientation). For many microprocessors, computation depends on byte-orientation(Big endian – Little endian).
- Simplicity. The risk of implementation errors increases with the complexity of the description. It is therefore advantageous to have a succinct and clear specification that makes an appeal to a pre-understanding of a reader.

Since DES was published [1], the round transformation used in most ciphers has the Feistel Structure. In this structure, the parts of bits of the intermediate round are simply transposed unchanged to another position. The round transformation of the proposed Taros does not has the Feistel structure but it composes of two main distinct invertible uniform transformations instead, as explained method following sections.

4. Specification

4.1 Structure of Taros

Taros is a non-Feistel network operating on 128-bit block. The block cipher uses 8 128-bit subkeys, derived from a user key by the key schedule. The structure of Taros is representing a physical tetrahedron structure as shown in *Figure 1*. The round transformation of Taros is composed of three distinct transformations. Round transformation of Taros is byte-oriented.

As shown in *Figure 2*, the Taros structure is composed of 4 triangles. Each piece is called a "*phrase*". All 4 phrases is called a "*state*". Each *phrase* can be divided into 4 segments (*phrase.TOP*, *phrase.RIGHT*, *phrase.LEFT*, *phrase.CENTER*). In each segment, 8 bit of data is assigned within each segment. So there are all together 16 segments (on 128 bits) in each *state*. For the sake of explanation, all 16 segments are numbered as shown in *Figure 2d*.

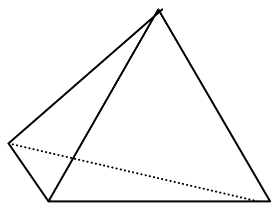


Figure 1: *The tetrahedron*

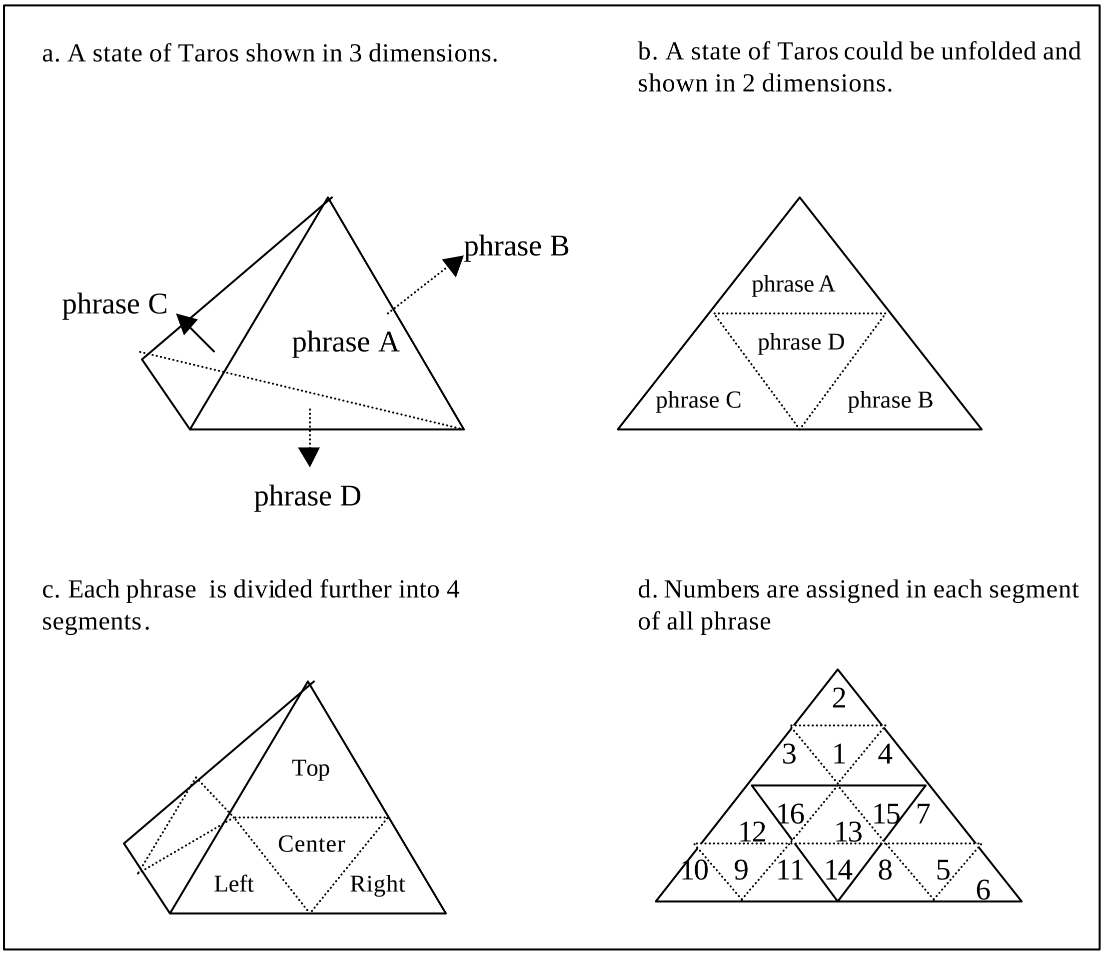


***Figure 2**: A state is divided into 4 phrases and each phrase is divided into 4 segments*

4.2 The round transformation

Let define some operators to perform some useful round transformation on the data. The round transformation is composed of four different operators, as following :

- RotatePhrase
- MixPhrase
- AddSubKey
- Whitening (first round and last round)

4.2.1 The RotatePhrase transformation

The RotatePhrase transformation is designed for high data diffusion, operating on each of the *pharses* independently. For encryption three angles of 4 corners of the tetrahedron rotate counterclockwise (counterclockwise when looking towards the corner *Figure 3*.). In decryption mode, rotation is the opposite direction of the encryption. Shown detail in *Figure 4*.

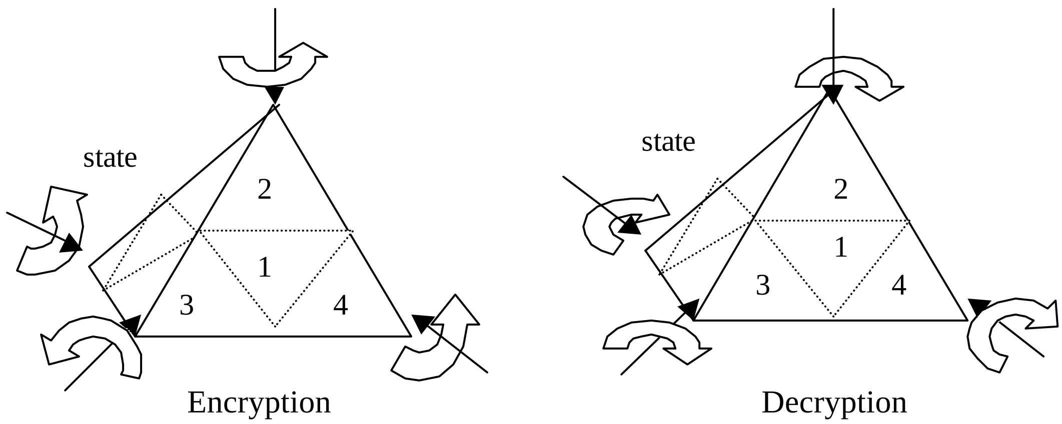


***Figure 3**: RotatePhrase transformation encryption and decryption*

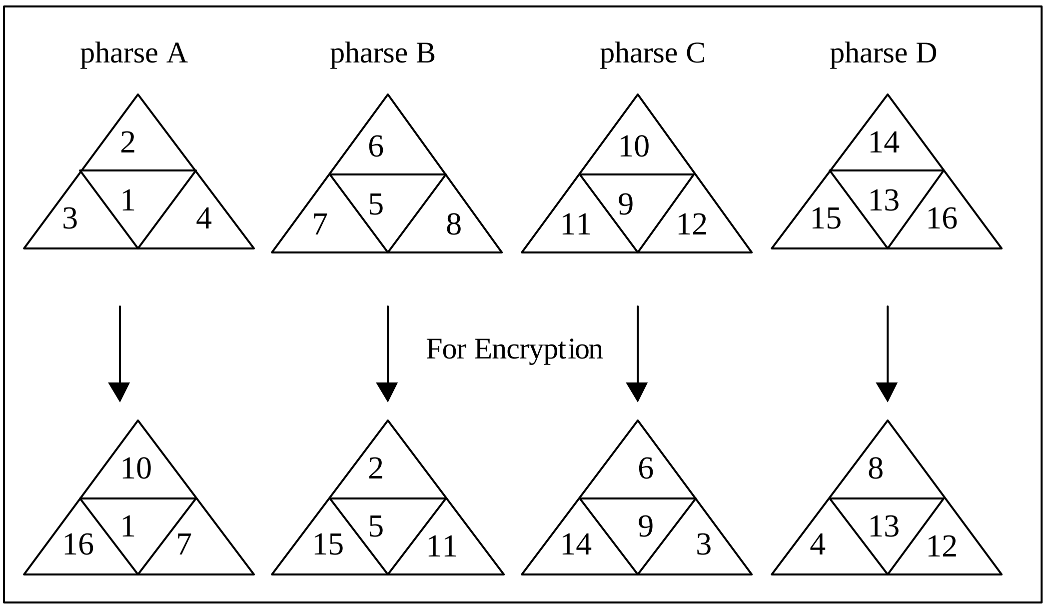


Figure 4: Geometrical representation of the RotatePhrase transformation.

4.2.2 The MixPhrase transformation

The MixPhrase transformation treats the different *phrase* of a *state* completely separately.

4.2.2.1 The Multiplication Polynomial

The four bytes of the *pharse* are mutilplied by polynomial of degree below 4. Multiplication by polynomials is more complicated. Assume we have two polynomials over $GF(2^8)$:

$$a(x) = a_3 x^3 + a_2 x^2 + a_1 x^1 + a_0 \text{ and } b(x) = b_3 x^3 + b_2 x^2 + b_1 x^1 + b_0$$

The product of polynomials a(x) and b(x) equals c(x). The output c(x) can no longer be represented by a 4-byte vector. To reducing c (x),we must modulo c(x) by a polynomial of degree 4. If we give a modulo polynomial equals $x^4 +1$. Then the modular product of a(x) and b(x) is given by

$$c(x) = c_3 x^3 + c_2 x^2 + c_1 x^1 + c_0$$

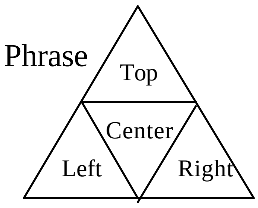
$c_0=a_0*b_0\oplus a_3*b_1\oplus a_2*b_2\oplus a_1*b_3$
$c_1=a_1*b_0\oplus a_0*b_1\oplus a_3*b_2\oplus a_2*b_3$
$c_2=a_2*b_0\oplus a_1*b_1\oplus a_0*b_2\oplus a_3*b_3$
$c_3=a_3*b_0\oplus a_2*b_1\oplus a_1*b_2\oplus a_0*b_3$

4.2.2.2 The MixPhrase Layout

In MixPhrase, the *phrases* of the *state* convert as a polynomials degree 4 over $GF(2^8)$ *Figure 5*. We use polynomial multiplication for data high diffusion. In MixPhrase transformation we want to generate fixed polynomial e(x). We must choose the polynomial e(x) having a properties invertible. The invert matrix of e(x) is used in MixPhrase in a decryption method (polynomials d(x)).. It can represent as:

$$Phrase_{new} = Phrase_{old} \otimes FixPolynomial$$

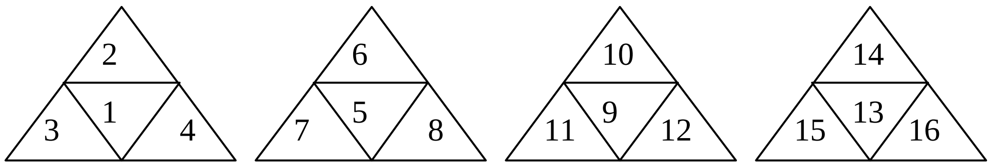
Detail is shown *Figure 6.*



$$Phrase = Phrase_{TOP}\ x^3 + Phrase_{LEFT}\ x^2 + Phrase_{RIGHT}\ x^1 + Phrase_{CENTER}\ x^0$$

$$FixPolynomial = f_3\ x^3 + f_2\ x^2 + f_1\ x^1 + f_0\ x^0$$

Figure 5: *Convert pharse to Polynomial degree 4 over GF(2^8)*



$$\begin{pmatrix} f_0 & f_3 & f_2 & f_1 \\ f_1 & f_0 & f_3 & f_2 \\ f_2 & f_1 & f_0 & f_3 \\ f_3 & f_2 & f_1 & f_0 \end{pmatrix} * \begin{pmatrix} 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \\ 1 & 5 & 9 & 13 \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} & c_{02} & C_{03} \\ c_{10} & c_{11} & c_{12} & C_{13} \\ c_{20} & c_{21} & c_{22} & C_{23} \\ c_{30} & c_{31} & c_{32} & C_{33} \end{pmatrix}$$

$c_{00} = 2 * f_0 + 3 * f_3 + 4 * f_2 + 1 * f_1$
$c_{10} = 2 * f_1 + 3 * f_0 + 4 * f_3 + 1 * f_2$
$c_{20} = 2 * f_2 + 3 * f_1 + 4 * f_0 + 1 * f_3$
$c_{30} = 2 * f_3 + 3 * f_2 + 4 * f_1 + 1 * f_0$
$c_{01} = 6 * f_0 + 7 * f_3 + 8 * f_2 + 5 * f_1$
$c_{11} = 6 * f_1 + 7 * f_0 + 8 * f_3 + 5 * f_2$
$c_{21} = 6 * f_2 + 7 * f_1 + 8 * f_0 + 5 * f_3$
$c_{31} = 6 * f_3 + 7 * f_2 + 8 * f_1 + 5 * f_0$

$c_{02} = 10 * f_0 + 11 * f_3 + 12 * f_2 + 9 * f_1$
$c_{12} = 10 * f_1 + 11 * f_0 + 12 * f_3 + 9 * f_2$
$c_{22} = 10 * f_2 + 11 * f_1 + 12 * f_0 + 9 * f_3$
$c_{32} = 10 * f_3 + 11 * f_2 + 12 * f_1 + 9 * f_0$
$c_{03} = 14 * f_0 + 15 * f_3 + 16 * f_2 + 13 * f_1$
$c_{13} = 14 * f_1 + 15 * f_0 + 16 * f_3 + 13 * f_2$
$c_{23} = 14 * f_2 + 15 * f_1 + 16 * f_0 + 13 * f_3$
$c_{33} = 14 * f_3 + 15 * f_2 + 16 * f_1 + 13 * f_0$

Figure 6 : Geometrical representation of the MixPhrase transformation.

4.2.3 The AddSubKey

In this operation, a Sub Key is applied to the *state* by a simple bitwise exclusive-or. The Sub Key is derived from the Cipher Key by means of the key schedule. The sub key length is equal to 128 bit

4.2.4 Whitening

Whitening, the technique of XORing key material can be used before the first round and after the last round. Tarsos uses the Whitening algorithm for increasing the difficulty of keysearch attacks against the remainder of the cipher. Taros XORs 128 bits of subkey is used before the first round and another 128 bits after the last round.

4.3 Key schedule

The key schedule is designed to be simple and to reuse the cipher components already available. Given a user key, which is a sequence of one or more 128-bit, it produces the 8 subkey required by the cipher. The key schedule is very similar to Blowfish [4]. The subkey array is assigned an initial constant value derived from the matrix used in the cipher. Words from the user key are XORed into the array, starting from the beginning, and restarting from the beginning of the user key when all the user key words are exhausted. A 128-bit block is initialized to zero, and enciphered with Taros, using the subkeys currently in the array. The first subkey words are then replaced with the resulting ciphertext, which is then encrypted again using the new subkeys. The next subkey words are replaced with the ciphertext, and the process will be reiterated until all of the subkey words have been replaced, 8 times for all. According to the algorithm, the Taros key schedule can accept user keys up to 1024 bits long.

The subkeys are calculated using the Taros algorithm. The exact method is as follows

1. Initialize of all subkeys , with fixed bytes (1024 bit).
2. XOR subkeys generating from Step 1 with cipher key (padding).
3. Encrypt the all-zero plaintext bytes with Taros algorithm, using the subkeys creating in Step 1 and 2.
4. Replace the first 128 bit of subkeys from Step 1 and 2 with the output of Step 3.
5. Encrypt the output of Step 3 using the Taros algorithm with the modified subkeys.
6. Replace the second 128 bits of subkeys with the output of Step 5.
7. Reiterate the process until replacing all entries in subkeys array.

5. Motivation for Design Choices

In the following subsections, we will describe the selected transformation algorithms.

5.1 The RotatePhrase transformation

The choice from all possible combinations has been made based on the following criteria:

1. The data on any *phrase* can move to other *phrase*.
2. Resistance against attacks using the Square attack
3. Simplicity

5.2 The MixPhrase transformation

MixPhrase has been chosen from the space of *phrase*-size to *phrase*-size linear transformations according to the following criteria:

1. Invertibility;
2. Linearity in GF(2);
3. Relevant diffusion power;
4. Simplicity of description;
5. Symmetry;

6. Square Attack

The “Square” attack is a dedicated attack on Square that exploits the byte-oriented structure of Square cipher. This attack is also valid for Taros. The attack is a chosen plaintext attack and is independent of the specific choices of RotatePhrase, MixPhrase and AddSubKey.

6.1 Preliminaries

Let a $\wedge$-set be a set of 256 states that are all different in some of the *state* bytes (the active) and all equal in the other *state* bytes (the passive). Applying the transformations AddSubkey on a $\wedge$-set with the positions of the active bytes unchnaged. Applying RotatePharse and MixPhrase to a $\wedge$-set the output of this operation can make the *phrase* having all active byte, an input of *state* with a single active byte gives rise to an output *phrase* with all bytes active.

6.2 The basic attack

Consider a $\wedge$-set in which only one byte is active, then we trace the evolution of the positions of the active bytes through 2 rounds. MixPhrase of the 1^{st} round converts the active byte to a complete *phrase* of active bytes. The three active bytes (phrase.TOP,phrase.RIGHT,phrase.LEFT) of this *phrase* are spreaded over other three *phrases* by RotatePhrase of the 2^{nd} round. MixPhrase of the 2^{nd} round subsequently spreads the active bytes throughout each *phrase*. (All bytes in each *state* are active). This stays a $\wedge$-set until it is transformed to be the input of MixPhrase of the 3^{rd} round.

Hence, all bytes at the input of the 3rd round of the *state* are balanced. By assigning a value to Kx_i (Subkey at round i), the value of *state* (at round i) for all elements of the $\wedge$-set can be calculated from the ciphertexts. If the values of this *tate* are not balanced over $\wedge$-set, the assigned value for the key byte was wrong. Detail in Figure 7.

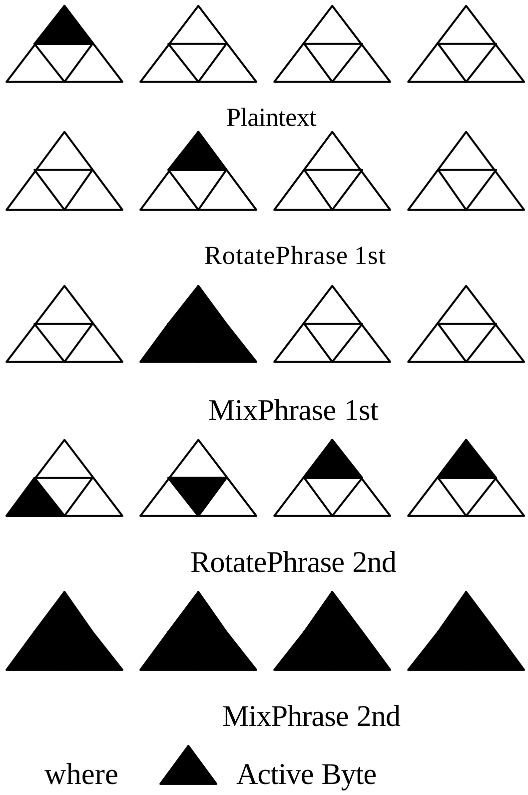


Figure 7**:** *Propagation of activity pattern through a second round*

7. Performance figure

Our estimates are based on the execution time of the KAT and MCT code on a 200 MHz Pentium, running Linux. The JDK1.1.1 Java compiler was used. The performance figures of the Java implementation are given in *Figure 8*.[6]

8. Conclusion and Further work

We propose a block cipher, Taros with 3D representation of tetrahedorn. We could accept a variable-length key up to 1024 bits whereas the Square block cipher can accept only 128 bit. Comparing RotatePharse transformation with the ShiftRow transformation of Rijndale [2], RotatePharse is use byte-substitution but ShiftRow uses matrix multiplication. The proposed Taros cannot be reversible like Twofish [5]and does not contain a non-linear byte substitution (The substitution table S-box). The paper uses only Square Attack crypanalysis for analysis. There are many other cryptanalysis algorithms in which improvement can be further studied.

9. References

[1] *Data Encryption Standard*, Federal Information Processing Standard (FIP), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977.

[2] *The Rijndael Block Cipher*, AES Proposal, Joan Daeman & Vincent Rijme

[3] J. Daemen, L.R. Knudsen and V. Rijmen, "The block cipher Square," Fast Software Encryption, LNC 1267, E. Biham, Ed., Springer-Verlag, 1997, pp. 149-165. Also available as http://www.esat.kuleuven.ac.be/rijmen/square/fse.ps.gz

[4] B. Schneier, "The Blowfish Encryption Algorithm", Dr Dobb's Journal, vol.19 no. 4, April 1994, pp. 38-40

[5] *Twofish: A 128-Bit Block Cipher*, AES Proposal, Bruch Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niel Ferguson

[6] *Report on the NIST Java™ AES Candidate Algorithm Analysis*, Jim Dray

Candidate	Code size	Heap	Key Setup	Encrypt	Decrypt
	(bytes)	(bytes)	(kbits/sec)	(kbits/sec)	(kbits/sec)
DEAL	16965	8624	52	140	140
LOKI97	9744	15016	96	294	294
Mars	18110	4808	107	492	496
Rijndael	12158	18360	**279**	1129	1129
Serpent	39290	**4680**	31	485	462
Taros	**5124**	16560	250	**1225**	**1225**
Twofish	17189	7600	37	379	379

Figure 8 : *Performance table*