The “PYRAMIDS” Block Cipher

Hussein Ahmad Al Hassan [1], Magdy Saeb [2], Hassan Desoky Hamed[1]
[1] Dept. of Mathematics, Faculty of Science, Cairo University.
[2]Arab Academy for Science, Technology and Maritime Transport,
Computer Engineering Department, Alexandria, Egypt.

E-mail: mail@magdysaeb.net

***Abstract*:** The "PYRAMIDS" Block Cipher is a symmetric encryption algorithm of a 64, 128, 256-bit plaintext block, that accepts a variable key length of 128, 192, 256 bits. The algorithm is an iterated cipher consisting of repeated applications of simple round transformations with different operations and different sequences in each round.

Keywords: block cipher, encryption, decryption, cryptography, algorithm, data communication security.

I. INTRODUCTION

In the last few years the merging of computer and communications technologies has been developing progressively. The need of security to protect the data through networks has become of vital importance. This area of research has become quite active in the recent years. Encryption algorithms have received wide attention and study. A lot of algorithms were published in this area such as RC6 [14], Rijndael (AES) [8], Serpent [3], Two-fish [16], Mars [6], Khazad [1]. These algorithms vary in security levels, design, structure, and throughput. All of these algorithms are using a fixed round function in the encryption and decryption processes. Explicitly, if the algorithm is to be attacked then the same scenario is applied irrespective of the user’s key. However, if the round function is changed with every different user’s key then the attacker needs to analyze the algorithm for every user’s key. In this work, we make full use of this simple fact to design an algorithm that is of high security and at the same time adaptable to hardware implementations. This work starts with the design rationale, and then move to the description of the algorithm itself. We then move to discuss the algorithm specifications and its key scheduling process.

II. DESIGN RATIONALE

The cipher is a 64, 128, 256-bit length, which also accepts a variable key length 128, 192, 256 bits. The PYRAMIDS is an iterated cipher consisting of a repeated application of a simple round transformation with different operations, and different sequence in each round. Moreover, we take into account, in the design criteria, robustness against the cryptanalysis, speed, simplicity, and suitability for hardware applications.

III. THE ROUND TRANSFORMATION

The round transformation is composed of different transformations. These transformations are addition +, xor ($\oplus$), right rotation (>>>), and ordered operation ($\otimes$). By considering the block of the text W is consisting of 4 words w_0, w_1, w_2, w_3, the sub-keys are $k_i^0, k_i^1, k_i^2, k_i^3$, and Rot is the rotation. The pseudo code of the ***i***-th round is shown next.

```
Round( W, k )
{       $w_1 = w_1 + k_i^1;$
        $w_2 = w_2 + k_i^2;$
        $w_0 = w_0 \oplus ((w_2 \otimes_i^1 k_i^0) >>> Rot_i^1);$
                                        // $f_i^1$
        $w_3 = w_3 \oplus ((w_1 \otimes_i^2 k_i^3) >>> Rot_i^2);$
                                        // $f_i^2$
        $w_1 = w_1 \oplus w_0;$
        $w_2 = w_2 \oplus w_3;$
        $w_0 = w_0 >>> Rot_i^2;$
        $w_3 = w_3 >>> Rot_i^1;$
        $(w_0, w_1, w_2, w_3) = (w_2, w_0, w_3, w_1);$
}
```

The pseudo code of decryption round is given by:

```
InvRound( W, k )
{   $(w_0, w_1, w_2, w_3) = (w_1, w_3, w_0, w_2);$
    $w_0 = w_0 >>> (L - Rot_i^2);$
                        // L is the word Length
$w_3 = w_3 >>> (L - Rot_i^1);$
$w_1 = w_1 \oplus w_0;$
$w_2 = w_2 \oplus w_3;$
$w_0 = w_0 \oplus ((w_2 \otimes_i^1 k_i^0) >>> Rot_i^1);$
                    // inverse of $f_i^1$
$w_3 = w_3 \oplus ((w_1 \otimes_i^2 k_i^3) >>> Rot_i^2);$
                    // inverse of $f_i^2$
    $w_1 = w_1 - k_i^1;$
    $w_2 = w_2 - k_i^2;$ }
```

The computational flow graph of the algorithm is shown in Figure 1. The algorithm utilizes the same operations for encryption and decryption procedures. During the execution cycle, each round function is executed using a different order of operations. Accordingly, this variable-round-calling will appreciably add to the algorithm security level.

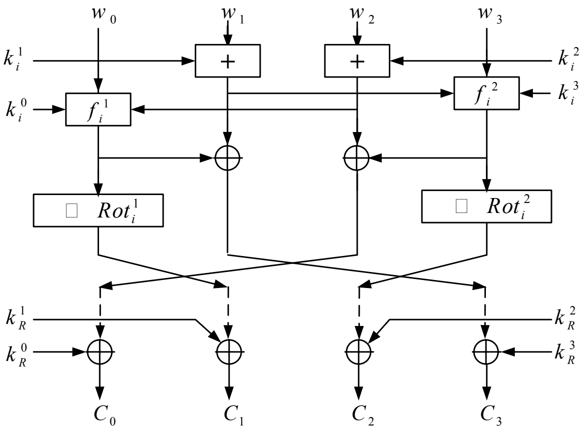

Figure 1: The Computational graph of PYRAMIDS' round.

The number of rounds is determined by the data size of the cipher since the key is expanded using different primitive equations depending on the size of the block. When the user key is less than 256 bit long, then it is expanded to 256 bit long, so number of rounds is depending on the data size and not the key size.

IV. The Algorithm Specifications

The PYRAMIDS provides a little variation between encryption and decryption procedures. The pseudo code for the encryption procedure is given by:

```
Encryption(Plaintext, Ciphertext, k, $\otimes$, Rot)
{for i=0 to R-1
   Round(Plaintext, Ciphertext, k, $\otimes$, Rot);
 for j=0 to 4
   Ciphertext[j]=Ciphertext[j]$\oplus k_{4R+1}^j$ }
```

The pseudo code for the decryption procedure is given by:

```
Decryption(Plaintext, Ciphertext, k, $\otimes$, Rot)
{for j=0 to 4
  Ciphertext[j]=Ciphertext[j]$\oplus k_{4R+1}^j$
 for i=R-1 down to 0
  InvRound( Ciphertext, Plaintext, k, $\otimes$, Rot);}
```

V. The Key Scheduling Process

The PYRAMIDS cipher uses *4*(R+1)* sub-keys. These sub-keys are derived from the user's input key using the key scheduling algorithm. This scheduling algorithm is explained as follows:

- The basic operations used in the key scheduling algorithm are:

x << *y*: shift the bits of *x* to left by the amount *y*.
x >> *y*: shift the bits of *x* to right by the amount *y*.
mod (%), bitwise AND (&).
The procedure employed to derive the sub-keys is as follows:

- The constants are constructed in the following way:

```
if (not odd($C_i = (C_{i-1} << 2) + (C_{i-1} >> 1)$))
     $C_i ++;$
```

Where *i* =1, 2, 3, …; and C_0 for 16, 32, 64-bit word are chosen randomly. In our implementation we choose C_0 randomly from 10000 digits of the fraction of the Golden Ratio. These starting constants are 0x1D5B, 0x5571D066, and 0xEB6F1276660AA498, for 16-bit word, 32-bit word, and 64-bit word respectively. These constants are required to avoid weak keys such as in the case of IDEA [11].
The key scheduling algorithm has three phases for every word length. Figure 2 depicts schematically these phases.

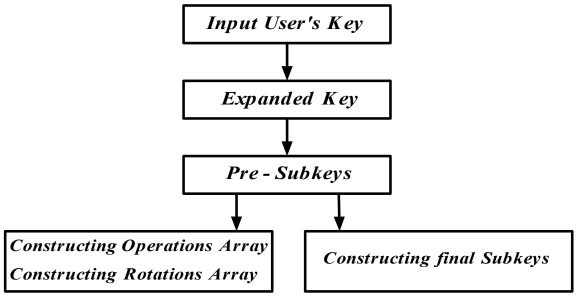
Input User's Key
Expanded Key
Pre - Subkeys
Constructing Operations Array
Constructing Rotations Array
Constructing final Subkeys

Figure 2: Computational graph of the key scheduling algorithm

These three phases are explained as follows:
Phase 1: In this phase, the input user's key is expanded and modified as follows:
Considering the number of words as q, then when the algorithm is executed in 16-bit word length mode, the 256-bit key length can be represented with q=16 words and the expanding procedure is:

```
for i=0 to q-1
    ExpandedKey[i]=(UserKey[i % s]
           <<<(UserKey[i % s] $\oplus$ C[i])) $\oplus$ C[q-1-i];
```

Where *s* is the length of input user's key.

Phase 2: In this phase, the pre-keys are constructed by passing the input user's key and constants through the round of the cipher.

```
Round(Subkey[0..3], Expandedkey[0..3],C[0..3],
                    $\otimes$ , Rot );
Round(Subkey[4..7], Expandedkey[4..7],C[4..7],
                    $\otimes$ , Rot );
```

Phase 3: In this phase, the sub-keys obtained from the previous phase are used to order the rotations' and operations' arrays. They are also used to construct the final sub-keys which are utilized through the encryption and the decryption procedures. The pseudo code of ordering the rotations and operations is as follows:

```
for i=0 to t
{   IndexRotation  [i]=subkey[ i % q] % (t-i);
    IndexOperation[i]=subkey[ (t-i) % q] % (t-i);
}

for i=0 to t
{ Swap(Rotation[i], Rotation[IndexRotation [i]]);
  Swap(Operation[i], Operation[IndexOperation[i]]);
}
```

The variable *t* represents the number of operations in each table for different word length.
The tables of initial rotations and operations are shown in appendix A. The final sub-keys are calculated as follows:

```
for i=  16 to 4*R+1
  subkey[i]=((((((subkey[i-16] $\oplus$ subkey[i-15])
                  + subkey[i-10])&subkey[i-9])
                  + subkey[i-2]) $\oplus$ subkey[i-1])
                   >>> subkey[i-3]) $\oplus$ C[i]
```

The key scheduling algorithm in the final sub-keys generations achieves fast avalanche. This is accurate since the final sub-keys generations are based on the primitive polynomial given by:

$$Q(x) = x^{16} + x^{15} + x^{14} + x^{7} + x^{6} + x + 1 \text{ in GF(2).}$$

These sub-keys are computed for 16-bit words. The final sub-keys for other word lengths are calculated as follows:

```
for i=  8 to 4*R+1
      subkey[i]=((((((subkey[i-8] $\oplus$ subkey[i-7])
                   +subkey[i-5])&subkey[i-4])
                   +subkey[i-3]) $\oplus$ subkey[i-1])
                    >>> subkey[i-2]) $\oplus$ C[i]
```

The final sub-keys generations achieve fast avalanche because it is based on the primitive polynomial:

$$Z(x) = x^{8} + x^{7} + x^{5} + x^{4} + x^{3} + x + 1 \text{ in GF(2).}$$

The arrays of the operations, rotations, and sub-keys are then sent to the encryption/ decryption procedure.

VI. Hardware Implementation Suitability

An implementation in the software of the proposed algorithm PYRAMIDS is one of the choices available to the user. However, in this algorithm, the primitive operations are integer addition, integer subtraction, xor$\oplus$, circular left (<<<), circular right (>>>).
These operations are very well-supported on modern microprocessors. They can be executed extremely fast, and all of them, naturally, are much faster than the multiplication operations used in some algorithms. We believe, based on our experimentation that is to be published in another work, that this algorithm is suitable for hardware-

implementation with high levels of throughput and reasonably consumed implementation area.

Moreover, this algorithm does not use look-up tables during encryption/decryption, nor does it use s-boxes. It just uses a sequence of low level operations and rotations. These features suggest that the algorithm architecture is relatively easy to fit onto the hardware chip. However, the use of the key schedule makes the algorithm need some additional memory to store the sub-keys to be used through the encryption/decryption processes.

VII. The Correlation Assessment

The correlation assessment technique is used to check whether there is a relation between the input and the output of the algorithm. We provide a test of our algorithm for a 32-bit word. By constructing *n* plaintexts represented in decimal format. The *i-th* plaintext is considered as

$$p_i\left(x_0^i, x_1^i, x_2^i, x_3^i\right).$$

The corresponding *n* cipher-texts are also presented in decimal format. The corresponding *i-th* cipher-text to the *i-th* plaintext is

$$c_i\left(y_0^i, y_1^i, y_2^i, y_3^i\right).$$

These *n* cipher-texts are encrypted under a fixed key that is chosen randomly. We calculate the correlation in the way that the word x_j^i in the *i-th* plaintext is corresponding to the word y_j^i in the *i-th* cipher-text, where $j = 0,1,2,3$, and $i = 1,\ldots,n$.

The correlation formula is given by:

$$Correl(P,C) = \frac{\sum_{i=1}^{n}\sum_{j=0}^{3}(x_j^i - \bar{x})(y_j^i - \bar{y})}{\sqrt{\sum_{i=1}^{n}\sum_{j=0}^{3}(x_j^i - \bar{x})^2 \sum_{i=1}^{n}\sum_{j=0}^{3}(y_j^i - \bar{y})^2}} \qquad (1)$$

The results are shown in appendix B. These results do not show any regularity or periodical behavior of the outputs.

VIII. Potential Attacks on the Algorithm

In this section we discuss potential attacks on the algorithm and show its robustness against various types of attacks. We divide these attacks into two different types; those which do not depend on the cipher design and those which depend on cipher design. These are discussed as follows:

8.1 Attacks not cipher design-dependent

These attacks are summarized as follows:

8.1.1 Dictionary Attack

The dictionary attack depends on the block size, and for the n-bit block size, the attacker needs 2^n different plaintexts to be able to encrypt arbitrary messages under an unknown key. For our proposed algorithm with a 64-bit block size, 2^{64} different plaintext blocks are required. For a 128-bit block size, 2^{128} different plaintext blocks are required. While for a 256-bit block size, 2^{256} different plaintexts blocks are required.

8.1.2 Modes of Operations

This attack is applicable to any deterministic block cipher. It depends on $2^{n/2}$ encrypted plaintext blocks in CBC or CFB mode for n-bit block size. The attacker can expect to find two equal cipher-text blocks, which enable the attacker to compute the xor of the corresponding plaintext [10].

For PYRAMIDS with a 64-bit block, the complexity is 2^{32}, with a 128-bit block the complexity is 2^{64}, and with a 256-bit block the complexity is 2^{128}.

8.1.3 Key-Collision Attacks

This attack can be used to forge a message with complexity depending only on the key size k, this complexity is $2^{k/2}$ [2].

For PYRAMIDS, the complexity of a 128-bit key length is 2^{64}. The complexity of a 196-bit length is 2^{98}, and the complexity of a 256-bit key length is 2^{128}.

8.2 Attacks that depend on the design of the Cipher

These attacks are summarized as follows:

8.2.1 Differential and linear Cryptanalysis

Let us assume that we have a set of r pairs of plaintexts/cipher-texts, and then the attacker will try to find differential or linear property between the plaintext/cipher-text with a high probability to exploit it in extracting some bits of the user key. In the proposed algorithm, the attacker will not be able to know the sequences of the operations and the rotations used in the algorithm since the order of the sequences and operations depends on the user key. To find the differential and linear property of the plaintext/ciphertext pairs, the attacker has to find all the properties for every sequence of operations and rotations with a high probability. In the case of 64-bit algorithm, we have 6 different operations, and 16 different rotations. The permutation of the operations is 6!. The permutation of the rotation is 16!. This provides us with $6!*16! \approx 2^{54}$ different sequences. That means that the attacker has to do 2^{54} different studies. For every study, the attacker has to find the linear or differential properties, and then uses the available pairs of plaintexts/ciphertexts to extract some bits from the key. Subsequently, the attacker has to perform exhaustive search to find the remaining bits of the key. However, this attack will consume time and effort more than the exhaustive key search itself. Moreover, if the attacker has r pairs of plaintexts/ciphertexts, then he/she has to use all the pairs to extract l bits from the key. Then he/she has to apply plaintexts/ciphertexts $r\ 2^{54}$ times depending on the different sequences. These operations have to be less than the exhaustive key search to be considered better than the exhaustive search attack. For a 128-bit key length, the operations are of the order of 2^{128}. By considering that the attacker has extracted l bits then the operations are $r\ 2^{54}$, and the exhaustive search for the rest of the remaining bits are 2^{128-l}. Therefore, the attack will be better than exhaustive key search if $2^{128} > 2^{128-l}\ r\ 2^{54}$. That is, $2^{l} > r\ 2^{54}$. However this is quite difficult to achieve. For example, if the attacker has $r = 50$ plaintexts, then the attacker has to extract more than 60 bits from the input user's key to be able to achieve the attack faster than the exhaustive key search, i.e. $l > 60$ bits. With this number of plaintexts, it is difficult to extract these bits from the input user's key faster than the exhaustive key search. The same study can be performed for other key lengths.

8.2.2 Higher Order Differential

This attack exploits the d-th order of the differential of a function of nonlinear order d which is constant [9]. This attack can be applied just for ciphers with few rounds and fixed round function. Then this attack is not applicable in our proposed algorithm by reason of the nature of the round function. Additionally, this attack is applicable for ciphers with a few rounds, which is different from our case.

8.2.3 Slide Attacks

The slide attack and the advanced slide attack are not applicable to this cipher. This is correct since the round function is not weak [4][5]. Also, the round function is changed every round where it does not satisfy $f[r] = f[r+1]$.

8.2.4 Boomerang Attack

This attack depends on differential properties [17]. For PYRAMIDS, as we have shown in section 8.2.1, it is very difficult to successfully achieve this type of attack. In other words, PYRAMIDS will be immune against the boomerang attack.

8.2.5 Algebraic Attack

In this attack, the attacker tries to construct algebraic equations of the cipher and the key, and hopes to solve these equations. Moreover this type of attack is applicable for cipher with s-boxes [7].
However, in our cipher this type of attack is not applicable. This is due to the non sequence of applying the round function. Additionally there are no s-boxes.

8.2.6 Interpolation Attack:

The interpolation attack depends on the components of the cipher with simple algebraic structures to construct a mathematical expression with low complexity [18]. In this attack, the attacker constructs a polynomial using pairs of plaintext/ciphertext. If the degree of the constructed polynomial is small, only a few pairs of plaintext/ ciphertext are needed to solve the polynomial coefficients. This attack is applicable on ciphers with a few rounds, while it is infeasible for more than a few rounds. Moreover, the polynomial will have a different form for every different key due to the sequence of the operations in the cipher.

8.2.7 Square Attack:

Applying Square attack[b] on PYRAMIDS cipher is very difficult. This is due to the change in the round function each round, hence tracing the active words is different for each key. For each key, the operations on the active words will be different in sequence, so the attacker has to guess the sequences of these operations used in the round function, and then the attacker will try to find the key by discarding wrong keys depending on active words and passive words. This work will be done for all sequence and this same to be infeasible.

IX. STATISTICAL ANALYSIS

The PYRAMIDS block cipher is tested using sixteen statistical tests (Frequency. Frequency within a bloc, Runs, Longest run of ones in a block, Binary matrix rank, Discrete Fourier transform, non-overlapping template, overlapping template, Universal statistical, Lempel-Ziv compression, linear complexity, serial, approximate entropy cumulative sums, random excursions, random excursions variant)[12][15]. The results do not indicate any deviation from random behavior. These tests are essential but not sufficient for security. A final word on this algorithm can be summarized as follows: The 3DES was developed to increase the security by using 2 keys, or 3 keys and these keys were used in this sequence $E_{k_1}(D_{k_2}(E_{k_3}(x)))$ or $E_{k_1}(D_{k_2}(E_{k_1}(x)))$ because the key length of DES is 56-bits (fixed length). However, this is not our case, we have variable key length, variable block length, so 3PYRAMIDS is not needed. For more security we can advise to increase number of rounds. We have chosen the algorithm to be called "Pyramids" since, similar to the Giza Pyramids; it may not reveal all of its secrets.

X. Software Implementation

The performance of the proposed algorithm is tested on Intel's Pentium Pro microprocessor 800 MHz, 256 KB cash memory, 192 MB memory, and Windows XP operating system with service pack2. The source code was written in Borland C Builder 6.0. The results demonstrate that the timing estimate showed that PYRAMIDS has a performance better than MESH-64, MESH-96, MESH-128 [12],[13], Rijndael (AES) [8], except for Rijndael with 128-bit block length and 128-bit key length. The performance comparison is shown appendix C.

XI. Application on images

To verify that the resulting cipher emulates almost communication white noise, we have applied it on a series of images. These images are shown in Appendix D. Investigating the set of encrypted images and comparing it to RC6 encrypted images, one concludes that the algorithm output emulates random noise. It is clear that there are no biases to any part of the original images contrary to images that are RC6-encrypted.

Summary and Conclusion

We have presented in this paper a block cipher with variable block length 64, 128, 256 bits, variable key length 128, 192, 256 bits. The special features of the proposed cipher are summarized as follows:

- The algorithm is an iterated cipher consisting of a simple round transformation with different operations and different sequences in each round.
- The sequence of the operations depends on the input user's key. In this respect, we can reason that the algorithm provides a dynamically-changed procedure for every different user's key.
- Encryption and decryption use the same operations. This constructs an efficient code. Moreover, the key scheduling algorithm uses the same round function of PYRAMIDS which gives rise to a more compact code.
- As depicted from Figure C.1, the throughput of the proposed algorithm is comparable to others ciphers. It is even faster than AES for most cases.
- The algorithm is secure since it shows robustness against various types of attacks as was discussed before. We have chosen the algorithm to be called "Pyramids" since, similar to the Giza Pyramids; it may not reveal all of its secrets.
- As shown in Table C.1 and Figure C.1, the algorithm provides a performance that surpasses the performance of MESH.
- Encrypting images shows that the algorithm emulates almost complete random noise with better results as compared to RC6.
- The structure of the algorithm and its low level operations provide a good basis for its potential in hardware implementations.

Based on the discussed comparison, one can conclude that the security and the performance of the proposed algorithm are satisfactory for multi-level security applications of today's networks.

References

[1] Barreto, P.S.L.M., V. Rijmen, "The Khazad legacy-level block cipher," First open NESSIE Workshop, Leuven, 13-14 November 2000.

[2] Biham, E., How to forge DES-Encrypted Messages in 228 Steps, Technical Report CS884, Technion, August 1996.

[3] Biham, E., R. Anderson, and L.R. Knudsen. Serpent: A New Block Cipher Proposal. In S. Vaudenay, editor, 5th Fast Software Encryption Workshop, LNCS 1372, pp 222-238. Springer-Verlag, 1998.

[4] Biryukov, A., and D. Wagner. Advanced Slide Attacks. In B. Preneel, editor, Advances in Cryptology, Eurocrypt'00, LNCS 1807, pp. 589-606. Springer-Verlag, 2000.

[5] Biryukov, A., and D. Wagner. Slide Attacks. In L.R. Knudsen, editor, 6th Fast Software Encryption Workshop,

LNCS 1636, pp. 245-259. Springer-Verlag, 1999.

[6] Burwick, C., D. Coppersmith, E. D'Avignon, R. Genario, S. Halevi, C. Jutla, S.M. Matyas, Jr, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic. MARS - a Candidate Cipher for AES. 1st AES Conference, California, USA, Aug 1998. http://csrc.nist.gov/encryption/aes/.

[7] Courtois, T., and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Quadratic Equations. In Y. Zheng, editor, Advances in Cryptology, Asiacrypt'02, LNCS 2501, pp. 267-287. Springer-Verlag, 2002.

[8] Daemen, J., and V. Rijmen. AES Proposal: Rijndael. 1st AES Conference, California, USA, 1998. http://www.nist.gov/aes.

[9] Knudsen, L. R., "Truncated and Higher-Order Differentials in Fast Software Encryption," 2nd International Workshop, LNCS 1800, pp. 196-211, Springer-Verlag, 1995.

[10] Knudsen, L. R., Block Ciphers – Analysis, Design and Applications, Ph.D. Thesis, Arhus University, Denmark, 1994.

[11] Lai, X., J.L. Massey, and S. Murphy. Markov Ciphers and Differential Cryptanalysis. In D.W. Davies, editor, Advances in Cryptology, Eurocrypt'91, LNCS 547, pp. 17-38. Springer-Verlag, 1991.

[12] Menezes, A. J., P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. CRC Press, Boca Raton, FL. 1997

[13] Nakahara Jr., J., V. Rijmen, B. Preneel, and J. Vandewalle, "The MESH Block Ciphers in Information Security Applications," 4th International Workshop, WISA 2003, Lecture Notes in Computer Science 2908, Springer-Verlag, pp. 458-473, 2004

[14] Rivest, R.L., M.J.B. Robshaw, R. Sidney and Y.L. Yin. The RC6 Block Cipher. v11, August 20, 1998. www.rsa.com/rsalabs/aes/.

[15] Rukhin, A., J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo," A Statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication, pp. 800-22, 2001.

[16] Schneier, B., J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. "Twofish: a 128-bit Block Cipher," 1st AES Conference, California, USA, Aug 1998. http://csrc.nist.gov/encryption/aes/.

[17] Wagner, D.,. "The Boomerang Attack," 6th Fast Software Encryption Workshop, LNCS 1636, pp. 156-170. Springer-Verlag, 1999.

[18] T. Jakobsen, L. R. Knudsen, "The Interpolation attack on block cipher," Fast Software Encryption, LNCS 1267, Springer-Verlag, pp. 28-40, 1997.

[19] J. Daemen, L. R. Knudsen, and V. Rijmen, "the block cipher SQUARE," Fast Software Encryption, LNCS 1267, Springer-Verlag, pp. 149- 165, 1997.

Acknowledgement: The authors would like to thank the reviewers for their numerous and helpful comments. The close scrutiny and constructive observations have greatly improved the final version of this paper.

APPENDIX A

The following tables demonstrate the order of operations and rotations. The order of these operations and rotations is changed depending on the input user's key in the key scheduling algorithm. Number of operations and rotations is increased for words of 32, 64-bit long.

Table A .1: the rotations in stage -1 of 64-bit block (word length is 16-bit)

Rotation in Round i	Rotation Value	Rotation in Round i	Rotation Value
0-Rot^1	1	4-Rot^1	9
0-Rot^2	2	4-Rot^2	10

1-Rot^{1}	3	5-Rot^{1}	11
1-Rot^{2}	4	5-Rot^{2}	12
2-Rot^{1}	5	6-Rot^{1}	13
2-Rot^{2}	6	6-Rot^{2}	14
3-Rot^{1}	7	7-Rot^{1}	15
3-Rot^{2}	8	7-Rot^{2}	8

Table A.2: the operations in stage -1 of 64-bit block (word length is 16-bit)

Operation in Round i	Operation	Operation in Round i	Operation
0-$\otimes^{1}$	X AND Y	4-$\otimes^{1}$	X AND ~Y
0-$\otimes^{2}$	X OR Y	4-$\otimes^{2}$	X OR ~Y
1-$\otimes^{1}$	X AND ~Y	5-$\otimes^{1}$	~ X AND Y
1-$\otimes^{2}$	X OR ~Y	5-$\otimes^{2}$	~X OR Y
2-$\otimes^{1}$	~ X AND Y	6-$\otimes^{1}$	X AND Y
2-$\otimes^{2}$	~X OR Y	6-$\otimes^{2}$	X OR Y
3-$\otimes^{1}$	X AND Y	7-$\otimes^{1}$	X OR ~Y
3-$\otimes^{2}$	X OR Y	7-$\otimes^{2}$	~X ANDY

APPENDIX B

The algorithm is a word oriented algorithm. In this test, we chose x_0^i randomly, while x_1^i, x_2^i, x_3^i are fixed through the test. We choose $x_1^i = 0,\ x_2^i = 0,\ x_3^i = 0$. We sort the n plaintexts in ascending order. The calculated correlation for $n = 65535$ using equation (1) is given by $Correl(P,C) = -0.0014$. When the correlation is close to ± 1 then the correlation is very high, and vice versa. We notice that the correlation is very close to zero. That means there is no correlation between the input and the output of the algorithm. Moreover, the graphs of the input plaintext and the output cipher do not show any regularity or periodical behavior of the output. This is clearly shown in Figures B.1, B.2, B.3, and B.4. The plaintext is considered x_0^i because all other words are zeros while we have four-word output, $y_0^i, y_1^i, y_2^i, y_3^i$.

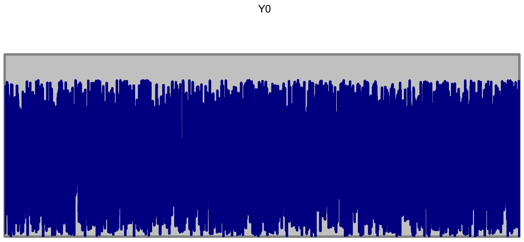

Figure B.1: The first word from the output of the input plaintexts

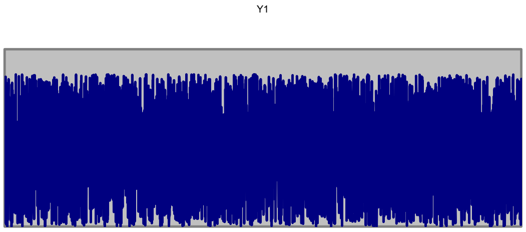

Figure B.2: The second word from the output of the input plaintexts

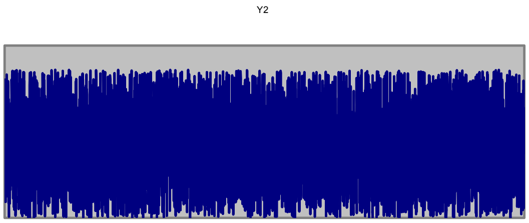

Figure B.3: The third word from the output of the input plaintexts

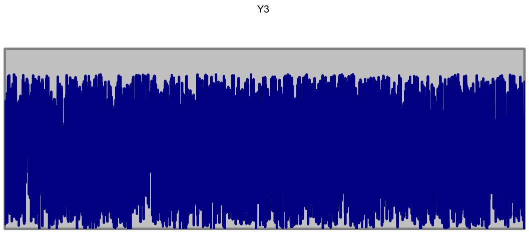

Figure B.4: The fourth word from the output of the input plaintexts

APPENDIX C

The following table and the accompanying chart provide a comparison of various block ciphers throughputs.

Table C.1: The performance of different algorithms

Algorithm Name	Mbps	Cycles/byte	Cycles/Block
MESH 128	8.7	506.7	8107
MESH 96	11.5	383.9	4606
MESH 64	15.1	291	2331

PYRAMIDS 64	15.6	281.9	2255
Rijndael 192-256	26.8	163.9	3934
Rijndael 256-256	27.2	161.8	5177
Rijndael 256-196	27.3	161.1	5156
Rijndael 256-128	27.6	159.3	5097
Rijndael 128-256	28.5	154.6	2474
Rijndael 192-196	28.8	153	3672
Rijndael 192-128	30.1	146.1	3507
Rijndael 128-196	32.6	134	2158
PYRAMIDS 256	33.4	131.6	4212
PYRAMIDS 128	36.8	119.4	1911
Rijndael 128-128	37.9	116.2	1859
RC6 128	75.9	57.9	927

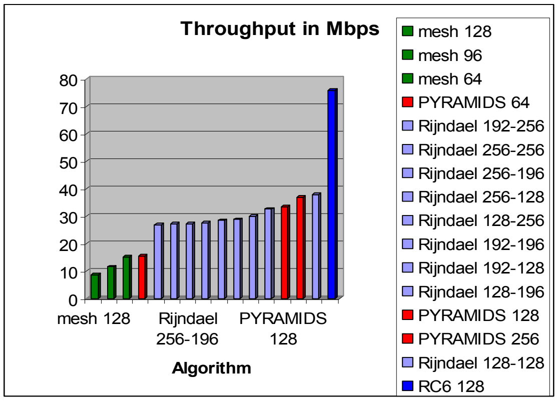

Figure C.1: The graph of the speed of different algorithms

APPENDIX D

To verify that the resulting cipher emulates almost color noise we have applied PYRAMIDS and RC6 on images. These results are shown in the following figures:

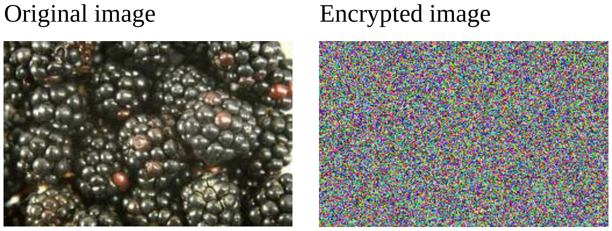

Figure D.1: The original and encrypted images using PYRAMIDS with 128-bits

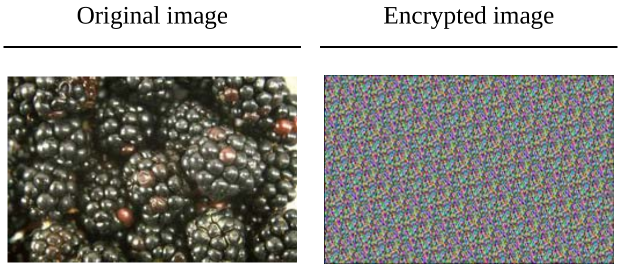

Figure D.2: The original and encrypted images using RC6

It is clear that RC6 shows some regular patterns while PYRAMIDS' output emulates almost random noise. This is mainly due to the scheduling algorithm and the key-dependent round ordering idea.

Explanation to the Ordered Operations:
By considering that the cipher has two rounds with initial sequence of operations $\otimes_0^1 = AND$, $\otimes_0^2 = OR$, $\otimes_1^1 = AND\sim$, $\otimes_1^1 =\sim OR$, then for a k_1 we presume that the operations sequence obtained from key scheduling is as follows:

$\otimes_0^1 = AND$, $\otimes_0^2 =\sim OR$,

$\otimes_1^1 = OR$, $\otimes_1^2 = AND\sim$.

In general we can write the first and third words as

$w_0^i = f_i^1\left(w_0^{i-1}, w_2^i, k^0, \otimes_i^1\right)$,

$w_3^i = f_i^2\left(w_1^i, w_3^{i-1}, k^3, \otimes_i^2\right)$ respectively,

where i is the round number, and k^0, k^3 are the sub-keys of the words w_0, w_3. Then the round functions will have the following form:

In the first round f_1^1, f_1^2 will be:

$$w_0 = w_0 \oplus ((w_2 \otimes_1^1 k_1^0) >>> Rot_1^1) = f_1^1$$

$$w_0 = w_0 \oplus ((w_2 AND\ k_1^0) >>> Rot_1^1)$$

$$w_3 = w_3 \oplus ((w_1 \otimes_1^2 k_1^3) >>> Rot_1^2) = f_1^2$$

$$w_3 = w_3 \oplus ((\sim w_1 OR\ k_1^3) >>> Rot_1^2)$$

In the second round f_2^1, f_2^2 will be as follows:

$$w_0 = w_0 \oplus ((w_2 \otimes_1^1 k_1^0) >>> Rot_1^1) = f_1^1$$

$$w_0 = w_0 \oplus ((w_2 AND\ k_1^0) >>> Rot_1^1)$$

$$w_3 = w_3 \oplus ((w_1 \otimes_1^2 k_1^3) >>> Rot_1^2) = f_1^2$$

$$w_3 = w_3 \oplus ((\sim w_1 OR\ k_1^3) >>> Rot_1^2)$$

In the second round f_2^1, f_2^2 will be as follows:

$$w_0 = w_0 \oplus ((w_2 \otimes_2^1 k_2^0) >>> Rot_2^1) = f_2^1$$
$$w_0 = w_0 \oplus ((w_2 OR\ k_2^0) >>> Rot_2^1)$$

$$w_3 = w_3 \oplus ((w_1 \otimes_2^2 k_2^3) >>> Rot_2^2) = f_2^2$$
$$w_3 = w_3 \oplus ((w_1 AND \sim k_2^3) >>> Rot_2^2)$$

For another key k_2 the sequence of the operations may have this sequence

$\otimes_0^1 = \sim OR$, $\otimes_0^2 = AND \sim$, $\otimes_1^1 = AND$,
$\otimes_1^2 = OR$.

Then the round functions will have the following:
In the first round f_1^1, f_1^2 will be as follows:

$$w_0 = w_0 \oplus ((w_2 \otimes_1^1 k_1^0) >>> Rot_1^1) = f_1^1$$
$$w_0 = w_0 \oplus ((\sim w_2 OR\ k_1^0) >>> Rot_1^1)$$
$$w_3 = w_3 \oplus ((w_1 \otimes_1^2 k_1^3) >>> Rot_1^2) = f_1^2$$
$$w_3 = w_3 \oplus ((w_1 AND\ k_1^3) >>> Rot_1^2)$$

In the second round f_2^1, f_2^2 will be as follows:

$$w_0 = w_0 \oplus ((w_2 \otimes_2^1 k_2^0) >>> Rot_2^1) = f_2^1$$
$$w_0 = w_0 \oplus ((w_2 AND\ k_2^0) >>> Rot_2^1)$$

$$w_3 = w_3 \oplus ((w_1 \otimes_2^2 k_2^3) >>> Rot_2^2) = f_2^2$$
$$w_3 = w_3 \oplus ((w_1 OR\ k_2^3) >>> Rot_2^2)$$

All rotations are right rotations (>>>), as previously shown in the algorithm.

Magdy Saeb:

Education: Ph.D., University of California, Irvine, School of Engineering, Electrical & Computer Engineering Dept. June, 1985. MSEE., University of California, Irvine, School of Engineering, Electrical & Computer Engineering Dept., June, 1981, BSEE., Cairo University, Cairo, School of Engineering, Electrical Engineering Dept., July, 1974.

Professional Experience: Professor, Arab Academy for Science, Tech. & Maritime Transport, School of Engineering , Computer Engineering (CE) Dept., Alexandria, Egypt. 1996-to-present
Head of Computer Engineering Dept., Taught graduate & undergraduate courses in Computer Networks, Parallel Processing, Computer Organization, Structure of Programming Languages, System Science, Pascal & C-programming, Computing Algorithms, Discrete Mathematics and Numerical Analysis.
MS. Thesis in CE advisor, BS. Projects advisor, coordinator of CE/CS MS. Program. Supervised MS. Thesis in Computer Network Reliability, NW Router Implementation, FPGA Implementation of Neural Nets, Visual Information Retrieval, Encryption Processors, Mobile Agent Security, Load Balancing Employing Mobile Agents, FPGA Implementations of Encryption and Steganography security techniques. He was with Kaiser Aerospace and Electronics, Irvine California, and The Atomic Energy Establishment, Anshas, Egypt.