The Chameleon Cipher-192 (CC-192):

A Polymorphic Cipher

Magdy Saeb

Arab Academy for Science, Tech. & Maritime Transport,

Computer Engineering Dept., Alexandria, Egypt

On-leave to: Malaysian Institute of Microelectronic Systems (MIMOS Bhd.),

Kuala Lumpur-57000, Malaysia

E-mail: mail@magdysaeb.net

Keywords: Polymorphic cipher, homophonic, random walks, key-driven, cryptanalysis, hash function.

Abstract:

The Chameleon Cipher-192 is a polymorphic cipher that utilizes a variable word size and variable-size user's key. In the preprocessing stage, the user key is extended into a larger table or bit-level S-box using a specially developed hash-function. The generated table is used in a special configuration to substantially increase the substitution addressing space. Accordingly, we call this table the S-orb. We show that the proposed cipher provides concepts of key-dependent number of rotations, key-dependent number of rounds and key-dependent addresses of substitution tables. Moreover, the parameters used to generate the different S-orb words are likewise key-dependent. We establish that the self-modifying proposed cipher, based on the aforementioned key-dependencies, provides an algorithm polymorphism and adequate security with a simple parallelizable structure. The ideas incorporated in the development of this cipher may pave the way for key-driven encryption rather than merely using the key for sub-key generation. The cipher is adaptable to both hardware and software implementations. Potential applications include voice and image encryption.

1 INTRODUCTION

A process is ergodic if and only if its' time averages' over a single realization of the process converge in mean square to the corresponding 'ensemble averages' over many realizations. As an example, suppose the process is x = k + f (t) + e where k is unknown, f (t) is nonlinear and e is a white noise error. Then any sample of x for a known t gives information about k and that is enough information to make predictions at remote times in the future that are just as good as predictions at nearby times. In this case one identifies such a process as a "not ergodic" process. Using this definition, we call a cipher, when represented by a stochastic process, "ergodic" if sampling its cipher text does not give enough information about its key to make predictions regarding its plain text at subsequent times [Gray08]. In this work, we apply this principle to design a polymorphic cipher [Baja01] that is based on a specially developed hash function and ergodic substitutions to provide the required diffusion and confusion with aperiodic behavior. The polymorphic nature of the cipher results from the dependency of some design parameters on the user key. The truly random behavior of the white noise error can be approximated by specific functions in the cipher structure.

1.1 CC-192 in General

We aim to design a polymorphic secure cipher that can be efficiently implemented in both software and hardware. The evolution of superscalar 64-bit word processors and the expanding use of smart cards provide the incentive for designing ciphers that are flexible and better suited for these varying architectures. CC-192 is a word-based cipher with variable word and key sizes. The key stream and the number of rounds are both key-dependent thus eliminating the possibility of trap door functions. The proposed ergodic process is also key-dependent emulating a faulty compass. These key-dependencies provide the foundation from which this polymorphic cipher acquired its name. Furthermore, these substitutions provide the required aperiodic random walks. We have used the concept of a faulty compass rather than chaotic maps since these chaotic systems usually suffer from unpredictable reproducibility problems. In summary, we use a group of transformations leading to an enhanced homophonic substitution [Penz94], [Gunth88], [Mass87], [Mass94] in which the mapping of characters varies depending on the sequence of bits in the message text. In executing the method, encryption keys are first generated. Then, enhanced homophonic substitution is performed. Finally, a poly-alphabetic substitution is performed on the data. This involves using bit-wise XOR between the partially ciphered data and the generated keys. The operation can be viewed as a linear masking operation. The high security of this proposed cipher is a result of the polymorphic key-dependent operations. The proposed cipher, implemented using C#, performs data encryption at about 26 cycles per byte using eight threads and 16 or 32-bit word size. Key setup consumes about 116 cycles per byte. This is achieved employing multithreading capabilities of modern superscalar processors using Intel Core2 Duo CPU E6550 @ 2.33 GHz, 4 GB RAM, 32-bit operating system. Various tests were performed and passed with no indication of deviation from random behavior. The security of the proposed cipher, based on algorithm polymorphism and a variable size S-orb, is acceptable for a large number of today's data security requirements. This will be established in detail in sections 3,5, and 10.

1.2 Organization

This article is organized as follows: in section 2, we provide a summary of the design objectives of CC-192. In section 3, the ideas of a polymorphic cipher and a brief mathematical background are presented. Section 4 provides a discussion of the cipher basic building blocks. These are the cipher structure, the S-orb, the hash function employed, the whitening and the key scheduling process. In section 5, we provide our design rationale. The details of the algorithm are described in section 6. A section on the key generation procedure is also provided. The statistical tests, discussion of trap doors, cipher security, applications and performance are discussed in detail in sections 8, and 9 respectively. Finally, we give a summary and our conclusions. The appendix provides some details of the hash function utilized.

1.3 Notation

We use the following notation: $\oplus$denotes logical XOR, $\wedge$ denotes logical AND, $\vee$ denotes logical OR, << and >> denote left and right logical bit-wise shift, <<< and >>> denote left and right bit-wise rotation, || denotes concatenation, and Hexadecimal numbers are prefixed by "0x". We apply integer notation for all variables and constants.

2 CC-192 DESIGN OBJECTIVES

The objectives taken into consideration while designing this cipher include:

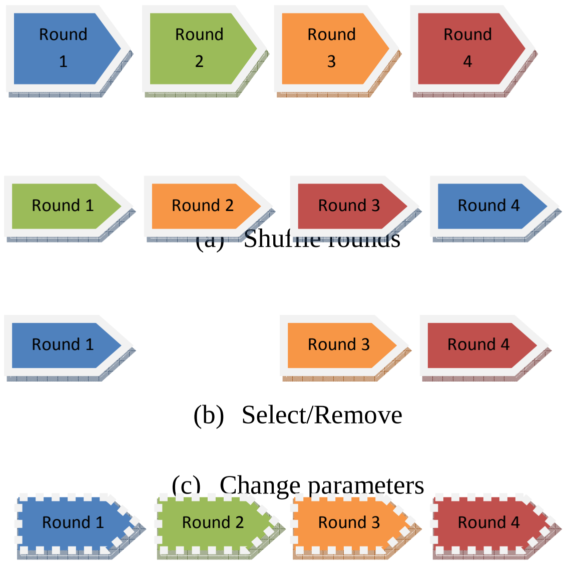
- The design of a key-driven, polymorphic highly secure cipher.
- Applicability to software with the proper utilization of today's superscalar processor architectures.
- Applicability to hardware with a design of a simple parallelizable cipher for FPGA-based applications.
- Flexible design; accepts keys and data blocks of different lengths and provide variable size S-orb depending on changing security requirements.
- Variable, key-dependent, number of rounds.
- Key setup time is kept to a minimum using a specially designed hash function.
- Simple construction and simple round function with minimum internal looping.

3 POLYMORPHIC STRUCTURE

For a true polymorphic cipher design, we propose three constructs:

- Shuffle
- Select/Remove
- Change parameters

One can visualize this approach as re-programming the cipher depending on an instruction set (number and function of various blocks). The micro-program instructions are actually stored in the user key. The larger the key size, the more "instructions" one can store. In conventional ciphers, the attacker uses the algorithm and the cipher to find a constant which is the key. However, in the proposed approach, the attacker has no substantial idea of the form of the algorithm since it is totally key-dependent. The attacker has to use the cipher to figure the algorithm construction first and then use the discovered algorithm and the cipher to find the key. The key is considered the memory of the system that contains not only the data segment, as in conventional ciphers, but also the program segment. If Alice sends the key to Bob, through a secure channel, she is actually sending both the structure of the algorithm and the data part used to expand the key. To approach the problem quantitatively, we provide the following discussion: In the Shuffle construct, we use the user's key to re-arrange the order of the operations. This idea was clearly used in the eight-block "Pyramids" block cipher [Hass05]. This technique, when applied to an n-operation structure, provides (n!) different algorithms. Each one of these algorithms has to be individually investigated by the attacker. On the other hand, if we change the parameters of the different operations with values depending on the user's key, one arrives at selection probabilities that correspond to one-out-of k cases. For example, if we assume that we can perform a variable number of rotations that depends on the register size utilized, say 32-bit register, then the probability of choosing the correct case is $1/2^5$. The same rationale applies to a varying number of rounds; say from 1 to 8 with a probability of choosing the correct one equal to $1/2^3$. For the correct bit-wise substitution, with a number of different cases equal to, say, 128 cases or addresses, the probability is $1/2^7$. To choose the correct values of the integers p_i and $q_{i,}$ used to update the next S-orb word, the attacker has to choose the correct values with a probability of $1 / 2^{32}$. Therefore, for an attacker to attain the correct probability he or she would have to try 2^{47} cases with a success probability of approximately 7.105 x 10^{-15}. Now to attack the hash function, acting as PRNG, using the birthday paradox, the success probability is given by $1 / 2^{96}$ using a 192-bit hash function. Thus, the overall probability of a successful attack on the cipher is $1/ 2^{143}$ or 89.68 x 10^{-45} which is smaller than a brute force attack using a 128-bit key. Future ciphers may embrace both of the two basic constructs; shuffle and select for highly secured applications. For the second construct "Remove", using the key one can reduce the number of operations; say L operations, from a maximum given number (n). Therefore, the attacker has to investigate a number of algorithms equal to (n! + (n-L)!). This basic notion of reprogrammable or polymorphic cipher is shown in Figure 1.



(a) Shuffle rounds

(b) Select/Remove

(c) Change parameters

Figure 1: The conceptual diagram of the constructs to realize a polymorphic cipher. Different colors represent different operations.

Now to compute the probability of a successful attack on a general polymorphic cipher, one starts with the probability of figuring out the algorithm utilized from shuffled n blocks. This probability is given by:

$$P_{shuffle} = 1/ n! \qquad (3.1)$$

Within each block there are m operations each operation i requires k_i parameters. Therefore, to select the correct parameter to operation i= 1, 2, … m

P_i = Pr {correctly selecting the parameter for operation i} = $1/k_i$

Then the probability to correctly select all parameters for all operations is given by:

$$P_{select} = \prod_{i=1}^{m} P_i = \prod_{i=1}^{m} 1/k_i \qquad (3.2)$$

Assuming all k_i are equal to, say k, then equation 4.2 takes the form:

$$P_{select} = \prod_{i=1}^{m} 1/k = 1/k^m \qquad (3.3)$$

Then the overall probability of finding the correct algorithm, allowing removal of certain blocks is given by:

$$P = \frac{1}{k^m\,(n!+(n-L)!)} \qquad (3.4)$$

For example, take m=5, k=4, n=8, L=2, then P will be equal to 2.37954 x 10^{-8}. However, the actual probability for a practical cipher will be much smaller than this value since the number of operations per block and the number of parameters will be larger than the previously given values. In Pyramids we have changed the order of operations. On the other hand, in Chameleon Cipher, we neither change the number of blocks nor their order; we only change the parameters of various operations and the number of rounds. Security here is founded on the notion of producing potentially large number of forms of a polymorphic algorithm. This is in contrast to conventional ciphers where it is implicitly assumed that the cipher machine is not reprogrammable. In these ciphers, the common wisdom dictates that there is no need to develop algorithms that can "Rewire the Enigma Machine". However, one can consider the user key as the system memory where both the user data and cipher re-programmability parameters are stored. In designing a cipher, the basic aim is to provide aperiodic or, in reality, a very long average period of the key stream. This can be partially achieved by ensuring a large internal state of the cipher. An ergodic process, as defined before, when sampling a cipher text does not give enough information about its key to make predictions regarding its plain text at subsequent times. We show that if the internal state of the cipher, represented by the S-orb address space, is made large enough such that the ergodic process can be correctly approximated.

4 CC-192 BUILDING BLOCKS

There are two distinct phases of performing this algorithm; the initialization of the S-orb and the encryption phase.

4.1 Initialization

The S-orb initialization of this cipher is performed "off-line" using the following recursive equation:

$$h_i = h\,(p_i\,.\,h_{i-1} + q_i) \qquad (4.1)$$

Where h_i is the hash function of the S-orb word (i). The total number of words of the S-orb (m) varies depending on the available memory and degree of security required. This value is taken equal to 6 resulting in an S-orb of six 192-bit words. The process is initialized with h_0 = h (k), where k is the user key, and p_i and q_i are two large secret integer numbers. These two numbers can be also obtained from the user key. The initial vector of the hash function (IV) is not necessarily to be kept secret. We use an assigned field in the round keys or S-orb words to determine the location of the center of what we call the "x-blocks". The contents of each block are used to perform the required substitution additions. The next step is to divide the plain text 192-bit block into six 32-bit words, 12 16-bit words or 24 eight-bit words. The same procedure is applied to different round keys. Now, we are able to perform the selective XOR operation, as shown in detail in the block diagram, in order to realize the required homophonic substitution. The next step is to perform a number of rotations to the partially ciphered words where this number is determined by a five-bit secret field of the round key. Finally, to perform the poly-alphabetic substitutions, we use the xor operation between the resulting partially ciphered word and the round key. The operation is repeated an additional number of rounds depending on the value obtained from the original user key. Other details are shown in the block diagram of Figure 2. This diagram illustrates the two basic operations utilized; initialization of the S-orb and the encryption phases. Figure 3.a illustrates some conceptual format details of the user and round keys. The substitution x-block is shown in at the lower side of Figure 3.b.

4.2 Key Schedule

A cipher, for a given security level, may require a relatively large number of round keys. Therefore, the S-orb number of words is intentionally left open to the user to increase the internal state of the cipher for added security. The user key can be varied from one bit to virtually any size key since it will be hashed using MDP-192 into a 192-bit set of round keys depending on the size of the S-orb.

Using a preprocessing phase of xoring the plain text with the user key and a post processing of xoring the cipher with the user key adds appreciably to the security of the algorithm as shown by Merkle [Merk90].

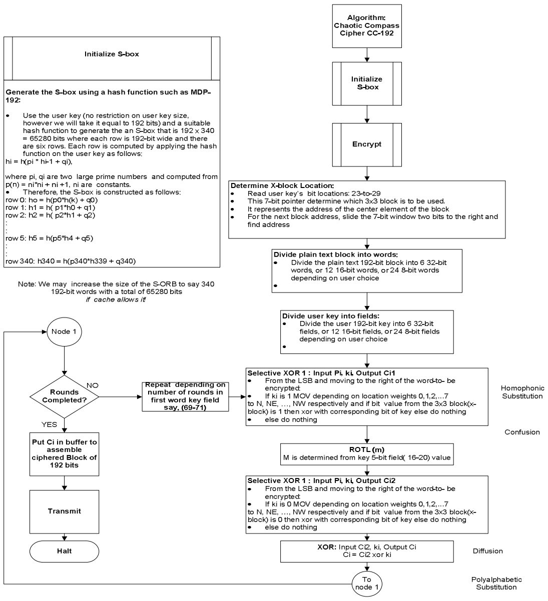

Algorithm: Chaotic Compass Cipher CC-192
Initialize S-box
Encrypt
Initialize S-box
Generate the S-box using a hash function such as MDP-192:
• Use the user key (no restriction on user key size, however we will take it equal to 192 bits) and a suitable hash function to generate the an S-box that is 192 x 340 = 65280 bits where each row is 192-bit wide and there are six rows. Each row is computed by applying the hash function on the user key as follows:
hi = h(pi * hi-1 + qi),
where pi, qi are two large prime numbers and computed from p(n) = ni*ni + ni +1, ni are constants.
• Therefore, the S-box is constructed as follows:
row 0: ho = h(p0*h(k) + q0)
row 1: h1 = h(p1*h0 + q1)
row 2: h2 = h(p2*h1 + q2)
:
:
row 5: h5 = h(p5*h4 + q5)
:
:
row 340: h340 = h(p340*h339 + q340)
Note: We may increase the size of the S-ORB to say 340 192-bit words with a total of 65280 bits
if cache allows it!
Determine X-block Location:
• Read user key's bit locations: 23-to-29
• This 7-bit pointer determine which 3x3 block is to be used.
• It represents the address of the center element of the block
• For the next block address, slide the 7-bit window two bits to the right and find address
Divide plain text block into words:
• Divide the plain text 192-bit block into 6 32-bit words, or 12 16-bit words, or 24 8-bit words depending on user choice
•
Divide user key into fields:
• Divide the user 192-bit key into 6 32-bit fields, or 12 16-bit fields, or 24 8-bit fields depending on user choice
•
Node 1
Rounds Completed?
NO
YES
Repeat depending on number of rounds in first word key field say, (69-71)
Selective XOR 1 : Input Pi, ki, Output Ci1
• From the LSB and moving to the right of the word-to- be encrypted:
• If ki is 1 MOV depending on location weights 0,1,2,...7 to N, NE, ..., NW respectively and if bit value from the 3x3 block(x-block) is 1 then xor with corresponding bit of key else do nothing
• else do nothing
Homophonic Substitution
Confusion
ROTL (m)
M is determined from key 5-bit field(16-20) value
Selective XOR 1 : Input Pi, ki, Output Ci2
• From the LSB and moving to the right of the word-to- be encrypted:
• If ki is 0 MOV depending on location weights 0,1,2,...7 to N, NE, ..., NW respectively and if bit value from the 3x3 block(x-block) is 0 then xor with corresponding bit of key else do nothing
• else do nothing
XOR: Input Ci2, ki, Output Ci
Ci = Ci2 xor ki
Diffusion
Put Ci in buffer to assemble ciphered Block of 192 bits
Transmit
Halt
To node 1
Polyalphabetic Substitution

Figure 2: The Chameleon Cipher

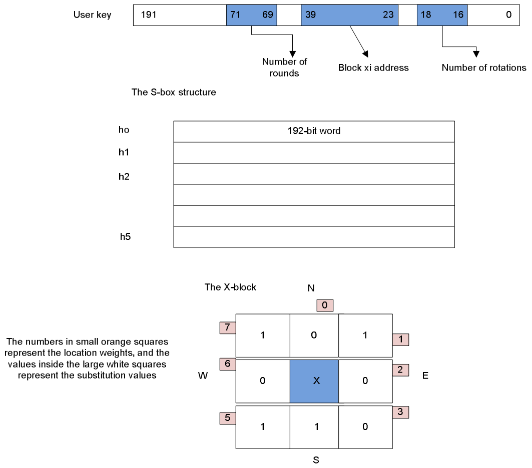

User key
191
71
69
39
23
18
16
0
Number of rounds
Block xi address
Number of rotations
The S-box structure
ho
h1
h2
h5
192-bit word
The X-block
N
0
7
1
6
2
5
3
1
0
1
0
X
0
1
1
0
W
E
S
The numbers in small orange squares represent the location weights, and the values inside the large white squares represent the substitution values

Figure 3: 3.a: (from top to bottom) The key format, the construction of the S-orb and 3.b: the X-blocks

4.3 Key setup

There are alternatives available to the cipher designer to build the S-box. These alternatives are:

1. Fixed S-box such as in DES [ANSI81], AES [Daem98].
2. Cipher-generated such as in BlowFish [Schn94].
3. SHA [FIPS95], [Brue05], hash function-generated such as in SEAL [Roga94].
4. Based on a specially-designed hash function

The first alternative may seriously compromise security. In the case of DES, there were a lot of conjectures that it contains trap doors. The second alternative consumes a large amount of key setup time. The third alternative is susceptible to attacks since it is based on a well-studied hash function. In our design, we have chosen the fourth alternative and designed our own hash function with features that can stand present and some future attacks. Moreover, the performance on modern superscalar processors of this hash and accordingly the key setup time were optimized and verified.

5 DESIGN RATIONALE

In the design of this cipher, we follow the general construction, suggested by T. Ritter [Ritt07]. In this basic structure, we utilize the ideas put forward by Ritter regarding the exchange of two message symbols. The shown transposition provides the required mathematical "permutation" of the message contents. However, this type of transposition notoriously has weaknesses when performed on the character level, since every character of the plain text is still visible in the cipher text. This allows for a chance for rearranging or "anagram' the cipher to find the plain text that makes sense. Nevertheless, if this permutation is performed on the bit-level, a large number of "Homophones" can be created. All but one is the required message. The concept behind this technique is rather simple. One starts by collecting data in blocks where the number of ones is almost equal to the number of zeros. Then the bits of these blocks are shuffled using a keyed pseudorandom number generator. We call this PRNG the "S-orb". If the sequence reuse is minimized, then one correctly obtains scrambled words or cipher text. However, it requires a PRNG with a relatively large internal state. This, in a sense, partially neutralizes the ability of the attacker to identify which permutation has occurred. The size of the S-orb was left variable to allow applicability to platforms with limited memory. Using conventional substitution tables may leak an infinitesimal fraction of these tables. This may lead to the exploitation of these ciphering tables. However, dynamic transposition provides an unbiased basic ciphering operation. Many different permutations will produce the exact same cipher from the same plain text. Thus, even known-plaintext does not expose the exact ciphering transformation. This is a form of balanced, nonlinear aggregation of the confusion sequence and data [Ritt07]. On the other hand, bit-permutation does consume a substantial execution time. However, in modern superscalar processors, this cost is increasingly becoming quite endurable. For simpler fine-grained processors, and FPGA-based implementations, one can always resort to parallelism or multiple similar data paths to compensate for this unavoidable increase in execution time. The proposed simple straight forward structure with minimal internal sequential looping makes this algorithm a good candidate for this type of parallelism. The second adopted principle in the design of this cipher is based on "algorithm polymorphism". The algorithm, depending on certain values embedded in the user key, transforms (morphs) into different forms. As shown in section 3, the attacker has a very small probability of discovering the correct form of the algorithm. In Chameleon Cipher, we only use parameter changes.

The design of the round function is kept simple and straight forward. It is based on two fast operations substitute and add (SUBSADD), and XOR. These two operations are supported in most modern processors. The rotation operation, while it is relatively slow, we found it essential for correct data scrambling and elimination of key leakage. The bit-wise substitutions are time- consuming. However, with proper utilization of modern superscalar processors, the associated delays are kept to a minimum. Even with such simple round function, it is well-known that increasing the number of rounds will provide the required security. This simple round function when iterated through a key-dependent number of rounds that is greater than or equal to a prescribed minimum number of rounds provides the required security. This approach contradicts conventional designs where the designers use strong round functions and less number of iterations. We view the performance as the overall execution time not the number of rounds. There is no internal looping per round. This feature provides the basis for parallelization on multi-thread superscalar processors. At the same time, the cipher can be easily implemented on FPGA using similar multi-data paths, as mentioned before, for improved performance.

5.1 The hash function MDP-192

Cryptographic hash functions or message digest have numerous applications in data security. The recent crypto-analysis attacks on existing hash functions have provided the motivation for improving the structure of such functions. The design of the proposed hash is based on the principles provided by Merkle's work [Merk79], Rivest MD-5 [Rive92], SHA-1 and RIPEMD [Hans96]. However, a large number of modifications and improvements are implemented to enable this hash to resist present and some probable future crypto-analysis attacks. The procedure, shown in Figure 4, provides a 192-bit long hash that utilizes six variables for the round function. A 1024-bit block size, with cascaded xor operations and deliberate asymmetry in the design structure, is used to provide higher security with negligible increase in execution time. The design of new hashes should follow, we believe, an evolutionary rather than a revolutionary paradigm. Consequently, changes to the original structure are kept to a minimum to utilize the confidence previously gained with SHA-1 and its predecessors MD4 [Rive90] and MD5. However, the main improvements included in MDP-1 are: The increased size of the hash; that is 192 bits compared to 128 and 160 bits for the MD-5 and SHA-1 schemes. The security bits have been increased from 64 and 80 to 96 bits. The message block size is increased to 1024 bits providing faster execution times. The message words in the different rounds are not only permuted but computed by xor and addition with the previous message words. This renders it harder for local changes to be confined to a few bits. In other words, individual message bits influence the computations at a large number of places. This, in turn, provides faster avalanche effect and added security. Moreover, adding two nonlinear functions and one of the variables to compute another variable, not only eliminates the possibility of certain attacks but also provides faster data diffusion. The fifth improvement is based on processing the message blocks employing six variables rather than four or five variables. This contributes to better security and faster avalanche effect. We have introduced a deliberate asymmetry in the procedure structure to impede potential and some future attacks. The xor and addition operations do not cause appreciable execution delays for today's processors. Nevertheless, the number of rotation operations, in each branch, has been optimized to provide fast avalanche with minimum overall execution delays. To verify the security of this hash function, we discuss the following simple theorem:

Theorem 5.1:

Let h be an m-bit to n-bit hash function where m >= n input keys k_1, k_2 to h.

Then h (k_1) = h (k_2) with probability equal to:

$$2^{-m} + 2^{-n} – 2^{-m-n}$$

Proof:

If $k_1 = k_2$, then h (k_1) = h (k_2).

However, if $k_1 \neq k_2$, then h(k_1) = h(k_2) with probability 2^{-n}.

$k_1 = k_2$ with probability 2^{-m} and $k_1 \neq k_2$ with probability $1-2^{-m}$.

Then the probability that h (k_1) = h(k_2) is given by:

$$\Pr \{h (k_1) = h (k_2)\} = 2^{-m} + (1 - 2^{-m}).\ 2^{-n}$$

As an example, assume two 192-bit different keys x_1, x_2 then

$$\Pr \{h(x_1) = h(x_2)\} = 2.\ 2^{-192} – 2^{-384}$$

$$= 2^{-191} (1 - 2^{-193}) \approx 3.186 \times 10^{-58}$$

This is a negligible probability of collision of two different keys.

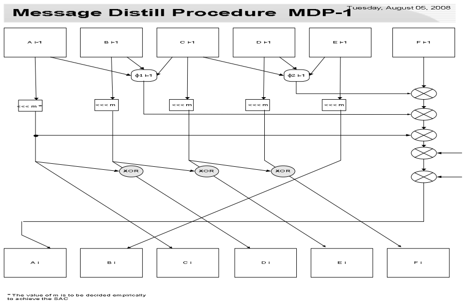


Figure 4: The hash function (MDP-192) used to generate the S-orb

5.2 The S-orb

As shown in section 4.1, the S-orb is constructed using an iterated application of MDP-192 on the round keys multiplied by and added to two large numbers. The iteration is initiated using the user's key. However, to increase the addressing space, we use the resulting table by folding it vertically and diagonally as shown in Figure 5. The resulting spherical configuration is what we have referred to as the S-orb. The programming effort involved is justifiable when one takes into consideration the potential increase in the number of addressable x-blocks.

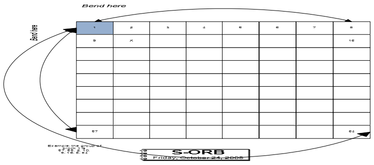


Figure 5: The conceptual S-orb

The generation of the S-orb based on the user key using a hash function eliminates the possibility of trap door functions. In addition, the number of words of the S-orb can be increased for added security and increasing the internal cipher period. This set of two large numbers is used to update the iterative hash calculation and is kept secret. This set is also user key-dependent. If we use, say six-word 192-bit per word S-box, then the number of x-blocks will be 128 blocks. However, if we use the S-orb configuration, then this number is increased to 1152 since each element can serve as the center of the x-block. Accordingly with such simple transformation of the S-box to an S-orb, the internal state of the cipher has been enormously increased. This may prove to be an important feature of the cipher for devices with limited memory.

6 THE ALGORITHM

In the next few lines we provide a formal description of the algorithm round structure.

Algorithm Chameleon-Cipher

[*Given a plain text message P, key K, the aim of the algorithm is to encrypt the plain text into a cipher text C and decrypt it again. To achieve this the algorithm utilizes a specially developed hash function to generate the key stream, and a dynamic transposition to permute the plain text, and finally modulo two addition to scramble a varying-size data unit*]

Encrypt:

Input: Plain text P, key K **Output:** Cipher C, word-size

Algorithm body:

Initialize the S-orb

Input: n is a positive integer ε $\mathbf{Z}^+$ equal to number of words of the S-orb, p_i, q_i are pairs of large positive integer numbers ε $\mathbf{Z}^+$ required to update the iterative application of the hash function.

Output: A 192-bit n-word table utilized as a pseudo random number generator PRNG called the S-orb.

```
        {Initialize S-orb body :}
i: = 0;
$h_0 := p_0. h (K) + q_0$;
{Hash the user key using MDP-192}
While i <= n
        $h_{i+1} := p_{i+1}. h_i (k) + q_{i+1}$;
        Save in S-orb file;
End while;
End Initialize;
 {Encrypt}
{P[m] = m blocks in P file}
Divide the Plaintext file P into m-1 192-bit blocks;
Append last block if necessary;
Read max-number of rounds from user key; {Input max-number of rounds from user key from assigned secret location in user key}
If max-number of rounds < 4 then max-number of rounds: = 4;
For round = 1 to max-number of rounds
While (P[m] ≠ EOF)        {EOF: End Of File}
        j := 0;
        While j ≠ n
        Read kw[j] of S file;
Using the round key kw[j], read value of integer given by bit location 23-to-29;        {This address represents the address of the center element of the block}
For the next block address, slide the 7-bit window two bits to the right and find new block address;
Divide the plain text 192-bit block into six 32-bit words, or twelve 16-bit words, or twenty four 8-bit words depending on user word-size;
From the LSB and moving to the right of the word-to- be encrypted: {Input: P[m], kw[j](round key), Output: $C_{i1}$}
If   $k_i$ =1 then move depending on location weights 0,1,2,...7 to N, NE, …, NW respectively then xor with corresponding bit of round-key kw[j];
Else do nothing;
ROTL (r); {r is determined from key 5-bit field (16-20) value, output $C_{i2}$}
{Input: $C_{i2}$, kw[j](round key), Output: $C_{i3}$}
If   $k_i$ =0 then move depending on location weights 0,1,2,...7 to N, NE, …, NW respectively then xor with corresponding bit of round-key kw[j];
Else do nothing;
{Input: $C_{i3}$, kw[j]$_i$ (round key), Output: $C_i$}
Ci = Ci3 xor kw[j];
Save $C_i$ in output file
End while;
Next round;
End Algorithm.
```

The substitution operation, using the x-blocks, explained above, provides homophonic substitutions that considerably improve the security of the cipher. In addition it provides the means to overcome the potential problem of block replay. The cipher is a binary-additive cipher that emulates a one-time-pad. The final xor operation between the partially ciphered text and the round key provides the polyalphabetic substitution and masking required for security. Inside the encryption process, the round keys effectively act as pointers in the homophonic substitutions without directly be part of the computations. This contributes to added security to the cipher. Testing of the cipher shows no bias to either ones or zeros and an average hamming distance of 3.8 for each byte encrypted. However, this value can be substantially increased with the increase of the minimum number of rounds. Testing of the cipher conforms to the Strict Avalanche Criteria (SAC) as required by New European Schemes for Signal Integrity and Encryption (NESSIE). The results are summarized and discussed in section 12. Contrary to conventional ciphers, the round function is kept simple and, in general, security is obtained through a relative increase of the number of rounds. This number of rounds can be large to ensure security. However, we adopted the idea of key-dependent number of rounds as long as it is greater than four rounds. This way, the security is increased twofold; by having an adequate number of rounds and at the same time hiding this number, in most cases, from the attacker.

7 KEY GENERATION

In this section, we provide a recap on some of the concepts that have already been presented regarding key-generation. Using the MDP-192 hash function recursively, one is able to generate the required PRNG. Most of the CPU time used in key-setup is actually consumed by this hash. The tests on this hash have shown that the average throughput, using Intel Core2 Duo CPU E6550 @ 2.33 GHz, 4 GB RAM, 32-bit operating system, is approximately 161.4 Mbps. That is 115.6 cycles per byte. If the key size is 128 bits, then we require around 1849.6 cycles for key setup. The recursive use of the hash function, as shown in equation 4.1, requires the multiplication of the hash by a large integer number p_i. There are a number of methods to achieve this object [Wels05]. We have adopted a modified version of Karatsuba Algorithm [Kara62], to perform this task. The homophonic selective substitutions were performed using 1152 x-blocks. No bias to the zeros or to the ones was completely and absolutely observed during the design phase. This was later verified based on the tests performed on the cipher.

8 STATISTICAL TESTS

The essential part of any cryptographic primitive is to generate a truly pseudo random sequence. A necessary but not a sufficient condition is to verify that there is no bias in the number of zeros or ones in the resulting cipher. This simple fact was repeatedly verified for various types of encrypted text, graphics or audio files. The results of these tests are shown in section 12. The strict avalanche criteria test is performed by changing one bit of the key and noting the change in the resulting cipher. As expected, and as required by NESSIE, the number of bits that have changed in the cipher is greater than or equal to 50%. The tests proposed by National Institute of Standards and Technology (NIST) and recommended by (NESSIE) were performed on the cipher. These tests are shown in the following list:

Frequency (Mono-bit) Test, Frequency within a block, Runs Test, Longest Run of ones in a block, Binary Matrix Rank Test, Discrete, Fourier Transform (spectral) Test, Overlapping Template Matching Test, Non-overlapping Template Matching Test, Maurer’s Universal Test, Lempel-Ziv Compression Test, Linear Complexity Test, Serial Test, Approximate Entropy Test, Cumulative Sums, Random Excursions Test, Random Excursions Variant Test. All of these tests were passed with no indication of deviation from random behavior. However, these tests are necessary but not a sufficient condition for a viable cipher. The simple cipher structure, the key-dependent number of rotations, the key-dependent addresses of the various x-blocks, the key-dependent number of rounds and above all the key-dependent S-orb all of these design parameters help eliminate the possibility of trap doors. In addition, the trap door has to endure the proposed variable number of rounds. The idea of a universal hidden key, in a sense, emulates a public key cryptography which is definitely not the case in this cipher [Schn98].

9 SECURITY & PERFORMANCE

The security features of this cipher are implicitly discussed in the sections covering polymorphic structure and design rationale. However, one claims that differential cryptanalysis, linear cryptanalysis, Interpolation attack, partial key guessing attacks, and side-channel attacks, hardly apply in this proposed cipher. The homophonic selective random substitutions and the polymorphic nature of the cipher, we believe, hide most traces that can be utilized to launch these attacks. Each key has its own unique “weaknesses” that will affect the new form of the algorithm utilized. Thus, different keys will produce different forms of the cipher. Accordingly, statistical analysis is not sufficient to link the plain text to the cipher text. With different inputs (user keys), we end up with a different “morph” of the cipher, therefore, it is totally infeasible to launch attacks by varying keys or part of the keys. Regarding the Key collision probability, it was shown in section 5.1 that the key collision probability is negligible when a 192-bit hash is applied. Moreover, we have proven that the attacker has a very small probability of guessing the correct form of the algorithm utilized. As was previously discussed, the simple structure of the proposed cipher provides a foundation for efficient software and hardware-based implementation. It is relatively easy to parallelize the data path either using multi-threading on a superscalar processor or by cloning this path on the FPGA material. The cryptographic selective substitution and the variable number of rotations provide a secure barrier against pattern leakage and block replay attacks in multi-media applications. Using ECB mode, when encrypting images with conventional ciphers, a lot of the structure of the original image will be preserved [Swen08]. This may lead to the problem of block replay. The selective substitution operation allows the cipher to encrypt images with no traces of the original image. This is a major advantage of the homophonic substitutions. In Figure 6, shown below, we encrypt part of the image to verify that there are no visible leaked structures from the original image. Based on the results shown in Figure 7, we provide a summary of the tests performed on the cipher in Table 1. The first and second ciphers, resulting from a one-bit change in key, were XORed to verify the SAC. The hamming distance is computed to assert that approximately one half the bits were changed as a result of the encryption process.

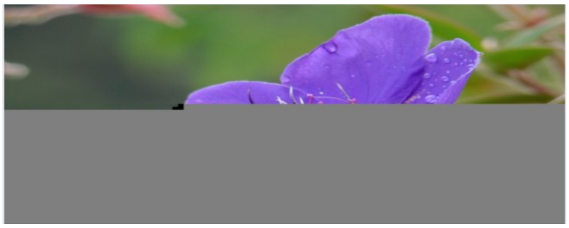


Figure 6 : A partially encrypted image, showing no structure leakage from the original.

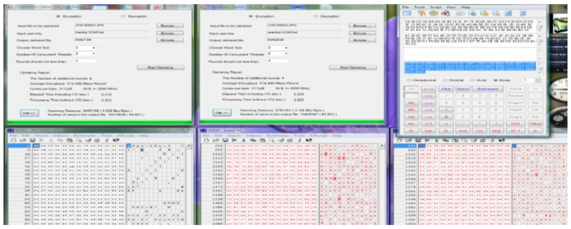


Figure 7: A screen shot showing the tests which results are summarized in table 1

The word size and the number of threads were changed to check for execution time, throughput and the number of cycles required to encrypt one byte. The tests were performed using Intel Core2 Duo CPU E6550 @ 2.33 GHz, 4 GB RAM, 32-bit operating system.

Table 1: Tests to verify SAC and no bias to zeros or ones in the cipher file

File Name & Size	DSC923, 2.49 MB	DSC923, 2.49 MB	DSC923, 2.49 MB
Keys	Frankly 1234, Erankly 1234	Frankly 1234, Erankly 1234	Frankly 1234, Erankly 1234
Cipher1	Beta1	Beta1	Beta2
Cipher2	Beta2	Beta2	Beta2
Cipher1 XOR Cipher 2	1’s = 49.9994%	1’s = 50.238%	1’s = 50.230%
Word size (bits)	8	16	32
Threads	8	8	8
Rounds	4+1	4+1	4+1
Throughput (Mbps/round)	514.486	610.764	610.764

Cycles per byte	31	26	26
System Frequency (MHz)	2.000	2.000	2.000
Execution time (sec), I/O included	0.219	0.187	0.187
Execution time (sec), I/O not included	0.203	0.171	0.171
Hamming distance (bits per byte changed)	3.636	3.778	3.804
Number of zeroes in encrypted file	49.84%	49.95%	49.95%

As seen from Table 1, the SAC is satisfied since the number of one's resulting from XOR two ciphers, encrypted with one-bit difference in key used (Frankly12345 versus Erankly12345), is about 50%. We have changed the number of threads and found the number of cycles per byte to be in the range 26-31 cycles per byte depending on the CPU usage. Using only one thread, the throughput is 304.491 Mbps, the cycles per byte is 52, the execution time with I/O included is 0.499 sec, and the execution time without I/O is 0.343 sec. As expected, using multithreading improves the performance appreciably. This is result of utilizing the parallelism in modern superscalar processors. We have not compromised the security for improved performance. We rather made full use of contemporary superior processors' performance.

10 SUMMARY & CONCLUSIONS

In this work, we have presented a polymorphic cipher, the rationale of its design and the general constructs required to build such a cipher. Throughout the process of implementing the ideas behind constructing such a cipher, we were able to demonstrate the following:

Design of a polymorphic secure cipher that can be efficiently implemented in both software and hardware. Contrary to conventional ciphers where it is implicitly assumed that the cipher machine is not reprogrammable, the proposed polymorphic cipher utilizes the user key to change the parameters of its operations. We have proposed three constructs in a general polymorphic cipher; Shuffle, Select/Remove and Change parameters as key-dependent operations. One considers the user key as the system memory where both the user key data and cipher re-programmability parameters are stored. The proposed cipher is a word-based cipher with variable word and key sizes. The bit-level S-orb replaces the conventional S-box leading to a noticeable increase in addressing space and added security. The key stream and the number of rounds are both key-dependent; thus eliminating the possibility of trap door functions. The generated S-orb is key-dependent using a specially-developed hash- function. The large integer numbers used in generating the different S-orb words are also key-dependent. The ergodic process, on which the cipher is based, is also key initiated emulating a faulty compass. These key-dependencies provided the foundation from which this polymorphic cipher acquired its name. Furthermore, these substitutions provide the required aperiodic random walks. We have used the concept of a faulty compass rather than chaotic maps since these chaotic systems usually suffer from unpredictable reproducibility problems. We have used selective additions leading to an enhanced homophonic substitution. In these homophonic bit-level substitutions, the mapping of characters varies depending on the sequence of bits in the message text. Inside the encryption process, the round keys act initially as pointers in the homophonic substitutions without directly being part of the computations. This contributes to added security. Finally, a poly-alphabetic substitution is performed on the data. This involves using bit-wise XOR between the partially ciphered data and the generated keys. The operation can be viewed as a linear masking operation. The high security of this cipher is a direct consequence of the polymorphic key-dependent design of the cipher operations' parameters.

The paradigm of polymorphic encryption provides the required security with relatively simple round function constructs. The security of the cipher was not compromised for an increase in its speed. We have preserved the pseudo-random permutations using robust bit-wise homophonic substitutions. In addition, we have utilized the capabilities of contemporary processors' superior performance to achieve acceptable execution speeds.

REFERENCES

[ANSI81] ANSI X3.92, (1981). American National Standard for Data Encryption Algorithm (DEA). *American National Standards Institute.*

[Baja01] Kostadin Bajalcaliev, May (2001). Quasi Functions and Polymorphic Encryption. http://eon.pmf.ukim.edu.mk/~kbajalc

[Brue05] Aiden A. Bruen, Mario A. Forcinito, (2005). *Cryptography, Information Theory and Error Correction,* Wiley-Inter-science.

[Daem98] Daemen and V. Rijmen, (1998). AES Proposal: Rijndael. *First AES conference*, California, US.

[FIPS95] Federal Information Processing Standard Publication, , April 17, (1995). Specifications for Secure Hash Standard. *FIPS PUB 180-1*, http://www.itl.nist.gov/fipspubs/fip180-1.htm

[Gray08] Gray, R.M., (2008). *Probability, Random Processes, and Ergodic Properties.* Springer Verlag, New York.

[Gunth88] Gunther, C., (1988). A Universal algorithm for homophonic coding. *Advances in Cryptology*, Eurocrypt 88, LNCS No. 330, pages 405-441, Springer-Verlag.

[Hans96] Hans Dobbertin, Antoon Bosselaers, Bart Preneel, (1996). RIPEMD-160: A Strengthened Version of RIPEMD. *Fast Software Encryption, LNCS 1039*, Springer-Verlag, pp. 71–82.

[Hass05] Hussein A. AlHassan, Magdy Saeb, Hassan D. Hamed, (2005). The Pyramids Block Cipher. *International Journal of Network Security*, Vol. 1, No., 1, pages 52-60.

[Kara62] Karatsuba A. and Yu Ofman, (1962). Multiplication of Many-Digital Numbers by Automatic Computers. *Proceedings of the USSR Academy of Sciences*, 145, pages 293-294.

[Merk79] Ralph C. Merkle, June, (1979). Secrecy, Authentication and Public Key Systems, *Ph.D. Dissertation*, Stanford University.

[Merk91] Merkle, R.C., (1991). Fast Software Encryption Functions. *Advances in Cryptology-CRYPTO '90 Proceedings*, pages.476-501, Springer Verlag.

[Mass87] Massey, J. L., (1987). On Probabilistic Encipherment. *IEEE Information Theory Workshop,* Bellagio, Italy.

[Mass94] Massey, J.L., (1994). Some Applications of Source Coding in Cryptography. *European transactions on Telecommunications*, Vol. 5, No. 4, pp.7/421-15/429.

[Penz94] Penzhorn, W.T., (1994). A fast homophonic coding algorithm based on arithmetic coding. *Fast Software Encryption, second International Workshop*, Leuven, Belgium, Lecture Notes in Computer Science1008, pages 329-346.

[Ritt07] Discussions by Terry Ritter, et al., 2007. http://www.ciphersbyritter.com/LEARNING.HTM.

[Rive90] Rivest, R.L., (1990).. The MD4 Message Digest Algorithm. *RFC 1186.*

[Rive92] Rivest, R.L., (1992). The MD5 Message Digest Algorithm. *RFC 1321.*

[Roga94] Rogaway, P., Coppersmith, D., (1994). A software-oriented Encryption Algorithm. *Fast Software Encryption Cambridge Security workshop Proceedings,* Springer-Verlag, pages 56-63.

[Schn94] Bruce Schneier, (1994). Description of a New variable-Length key, 64-bit Block Cipher (Blowfish). *Fast Software Encryption, Cambridge Security Workshop Proceedings,* Springer-Verlag, pages 191-204.

[Schn98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, , (1998). Twofish: A 128-bit Block Cipher. *First AES conference,* California, US.

[Swen08] Swenson, C., (2008). *Modern Cryptanalysis; Techniques for advanced Code Breaking,* Wiley Pub. Inc.

[Wels05] Michael Welschenbach, (2005). *Cryptography in C and C++*, Apress.

Magdy Saeb received the BSEE. In Electrical Engineering, School of Engineering from Cairo University, in 1974; the MSEE. and Ph.D. in Electrical & Computer Engineering, School of Engineering from University of California, Irvine, in 1981 and 1985, respectively. He was with Kaiser Aerospace and Electronics, Irvine California, and The Atomic Energy Establishment, Anshas, Egypt. He is currently a professor in the Department of Computer Engineering, Arab Academy for Science, Technology & Maritime Transport, School of Engineering, Alexandria, Egypt., (on leave) to MIMOS, Malaysia. His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography security techniques, Encryption Processors, Computer Network Reliability, NW Router Implementation, FPGA Implementation of Neural Nets, Visual Information Retrieval, Mobile Agent Security, Load Balancing Employing Mobile Agents.