The Stone Cipher-192 (SC-192): A Metamorphic Cipher

Magdy Saeb

Computer Engineering Department,
Arab Academy for Science, Tech. & Maritime Transport
Alexandria, EGYPT
(On-Leave), Malaysian Institute of Microelectronic Systems MIMOS
Kuala Lumpur, MALAYSIA
mail@magdysaeb.net

Abstract: *The Stone Cipher-192 is a metamorphic 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 one-way function. However for added security, the user key is first encrypted using the cipher encryption function with agreed-upon initial values. The generated table is used in a special configuration to considerably increase the substitution addressing space. Accordingly, we call this table the S-orb. Four bit-balanced operations are pseudo-randomly selected to generate the sequence of operations constituting the cipher. These operations are: XOR, INV, ROR, NOP for bitwise xor, invert, rotate right and no operation respectively. The resulting key stream is used to generate the bits required to select these operations. We show that the proposed cipher furnishes concepts of key-dependent pseudo random sequence of operations that even the cipher designer cannot predict in advance. In this approach, the sub-keys act as program instructions not merely as a data source. 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 metamorphism 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.*

Keywords: metamorphic, polymorphic, cipher, cryptography, filters, hash.

1. Introduction

A metamorphic reaction takes place in a rock when various minerals go from amphibolites facies to some color schist facies. Some of the minerals such as quartz may not take place in this reaction. The process in its essence follows certain rules; however the end result provides a pseudo random distribution of the minerals in the rock or stone. The metamorphic natural process results in thousands or even millions of different shapes of the rock or stone. This process has inspired us to design and implement a new metamorphic cipher that we call "Stone Cipher-192". The internal sub-keys are generated using a combination of the encryption function itself and a 192-bit specially-designed one-way function. The idea of this cipher is to use four low level operations that are all bit-balanced to encrypt the plaintext bit stream based on the expanded stream of the user key. The key stream is used to select the operation; thus providing a random however recoverable sequence of such operations. A bit-balanced operation provides an output that has the same number of ones and zeroes. These operations are XOR, INV, ROR and NOP. Respectively, these are, xoring a key bit with a plaintext bit, inverting a plaintext bit, exchanging one plaintext bit with another one in a given plaintext word using a rotation right operation and producing the plaintext bit without any change. In fact, these four operations are the only bit-balanced logic operations. In the next few sections, we discuss the design rationale, the structure of the cipher, the one-way function employed to generate the sub-keys, the software and hardware implementations of the cipher, a comparison with a polymorphic cipher and a discussion of its security against known and some probable cryptanalysis attacks. Finally, we provide a summary of results and our conclusions.

2. Design Rationale

It is a long-familiar fact that all ciphers, including block and stream ciphers, are emulating a one-time pad OTP. However, for provable security, the key bits have to be used only once for each encrypted plaintext bit. Obviously, with present day technology this is not a practical solution. Alternatively, one resorts to computational complexity security. In this case, the key bits will be used more than once. Unfortunately, this will provide the cipher cryptanalyst with the means to launch feasible statistical attacks. To overcome these known attacks, we propose an improvement in the nonlinearity-associated filtering of the plaintext bits. This can be achieved in various ways as shown in [1]; however, the process can be further simplified and become appreciably faster and more riotously-secure if we parallelize all operations employed. We will establish that the proposed configuration can be further parallelized to enormously improve its security and throughput. One can imagine the algorithm as a pseudo random sequence of operations that are totally key-dependent. Accordingly, we presuppose that most known attacks will be very difficult to launch since there are no statistical clues left to the attacker. The algorithm utilized is randomly selected. Even the cipher designer has no clear idea what is the sequence of bitwise

operations would be. The encryption low-level operations are selected to be bit-balanced. That is, they do not provide any bias to the number of zeroes or ones in the output cipher. The result of such an approach will be the creation of an immense number of wrong messages that conceal the only correct one. Therefore, the cryptanalyst is left with the sole option of attacking the key itself. However, if the sub-keys are generated based on a cascade of the same encryption function and a one-way hash, then we conceive that these attacks will be unmanageable to launch. We are producing an unexampled key-dependent encryption algorithm. In this case, the least high-priced kept secret is the key. The proposed system is malleable and resilient if unknowingly disclosed. This theme does not dispute Kerckhoffs' principle [2] or Shannon's maxim since the "enemy knows the system". However, it provides a degree of security against statistical attacks [3] that, we believe, cannot be attained with conventional ciphers [4], [5], [6], [7], [8],[9].

3. The Structure of the Cipher

The conceptual block diagram of the proposed cipher is shown below in Figure 1. It is constructed of two basic functions; the encryption function and the sub-key generation one-way hash function. The pseudo random number generator is built using the same encryption function and the one-way hash function in cascade. Two large numbers (a, b) are used to iteratively generate the sub-keys. The details of the substitution box or what we call the S-orb can be found in [1]. The user key is first encrypted then the encrypted key is used to generate the sub-keys.

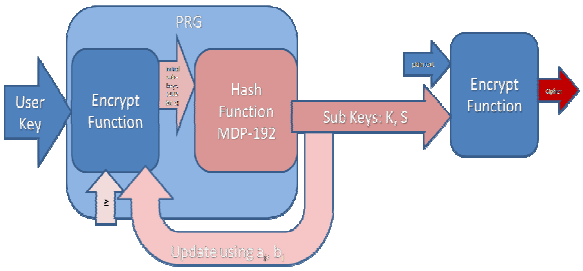

Figure 1. The structure of the cipher

The encryption function or the cipher engine is built using four low-level operations. These are XOR, INV, ROR and NOP. Table 1 demonstrates the details of each one of these operations.

Table 1: The basic cipher engine (encryption function) operations

Mnemonic	*Operation*	*Select Operation code*
XOR	$C_i = K_i \oplus P_i$	00
INV	$C_i = \neg(P_i)$	01
ROR	$P_i \leftarrow P_j$	10
NOP	$C_i = P_i$	11

The basic crypto logic unit (CLU) is shown in Figure 2. All operations are at the bit level. The unit is to be repeated a number of times depending on the required word or block size. The rotation operation, referred to by the circular arrow, is performed using multiplexers as shown in Figure 3. In the software version these multiplexers are replaced by "case" or "switch" statement. This CLU is used as the encryptor or the decryptor. This can be easily verified, if we investigate the truth table shown in Appendix A. In this table, if we change the output cipher bit to become an input plain text bit, the new output will be the same as the old plain text bit. Obviously, this is a feature of the applied functions namely XOR, INV or NOP. The only exception is in the case of ROR, the decryptor will use ROL.

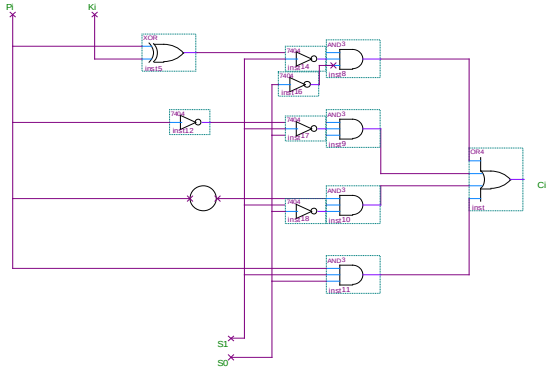

Figure 2. The basic crypto logic unit

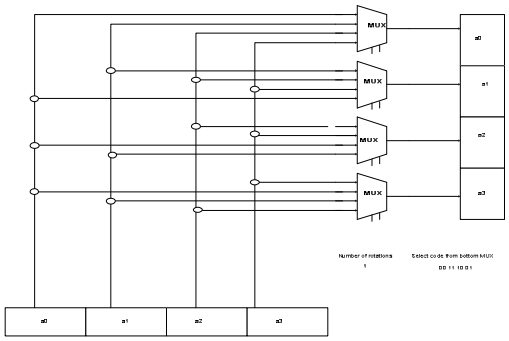

Figure 3. The rotation operation (ROTR) implementation using multiplexers

The operation selection bits (S_1 S_0) can be chosen from any two sub-key consecutive bits; as shown in Figure 4. The same applies for the rotation selection bits (S'_1 S'_0).

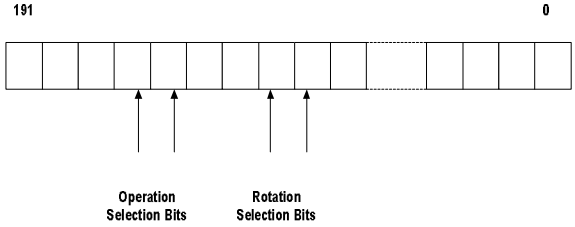

Figure 4. The proposed key format where the location of the selection bits is shown

4. The One-way Hash Function

Cryptographic one-way 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 [10], Rivest MD-5 [11], SHA-1 and RIPEMD [12]. 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 5, provides a 192-bit long hash [13] that utilizes six variables for the round function.

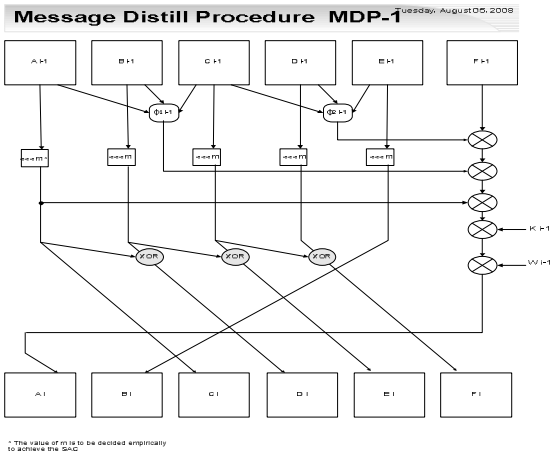

Figure 5. Operation of MDP-192 one-way function [13]

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[14] and MD5. However, the main improvements included in MDP-192[13] 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 [13]:

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.

5. The Pseudo Random Number Generator (PRG)

The combination of the encryption function and the one-way hash function is used to generate the sub-keys. The cipher designer has to select which one should precede the other. Based on the work by Maurer and Massey [15] where they have proved that a cascade of two ciphers is as strong as its first cipher. Therefore, we have adjudicated to start with the encryption function. The one-way hash function is then used recursively to generate the sub-keys based on two large numbers that are derived from the user key. In this case, the encryption function requires some initial agreed-upon vector value (IV), [16], [17], [18] to complete the encryption process. This IV can be regarded as a long-term key or even a group-key that can be changed on a regular basis or when a member leaves the group. The combination of the encryption function and the one-way function are used as the required pseudo random number generator PRG. It is worth pointing out that the design of the cipher intentionally allows the change of the one-way hash if successfully attacked.

6. The Algorithm

The algorithm can be formally described as shown in the next few lines.

Algorithm: STONEMETAMORPHIC
INPUT: Plain text message P, User Key K, Block Size B
OUTPUT: Cipher Text C
Algorithm body:
Begin
Begin key schedule

1. Read user key;
2. Encrypt user key by calling encrypt function and using the initial agreed-upon values as the random input to this function;
3. Read the values of the large numbers a and b from the encrypted key;
4. Generate a sub-key by calling the hash one-way function and using the constants a, b;
5. Store the generated value of the subkey;
6. Repeat steps 5 and 6 to generate the required number of subkeys;
End key schedule;
Begin Encryption
7. Read a block B of the message P into the message cache;
8. Use the next generated 192-bit key to bit-wise encrypt the plain text bits by calling the encrypt function;
9. If message cache is not empty, Goto step 8;
10. Else if message cache is empty:
If message not finished
10.1 Load next block into message cache;
10.2 Goto 8;
Else if message is finished then halt;
End Encryption;
End Algorithm.
Function ENCRYPT
Begin
1. Read next message bit;
2. Read next key bit from sub-key;
3. Read selection bits from sub-key;
4. Read rotation selection bits from sub-key;
5. Use selection & rotation bits to select and perform operation: XOR, INV, ROR, NOP;
6. Perform the encryption operation using plaintext bit and sub-key bit to get a cipher bit;
7. Store the resulting cipher bit;
End;

As seen from the above formal description of the algorithm, it simply consists of a series of pseudo random calls of the encryption function. However, each call will trigger a different bitwise operation. The simplicity of this algorithm readily lends itself to parallelism. This parallelism can be achieved using today's superscalar multi-threading capabilities or multiple data paths on a specialized hardware such as FPGA with their contemporary vast gate count.

7. Software Implementation

The pseudo C-function [19] that represents such a table is given by:

```
encrypt (plain-text-bit, key-bit, selection-bit0, selection-bit1, rot-bit)
{
        a1= plain-text-bit ^ key-bit;
        e1= a1 & (~selection-bit0) & (~selection-bit1);
        b1= ~ plain-text-bit;
        f1= b1 & (selection-bit0) & (~selection-bit1);
        g1= rot-bit & (~selection-bit0) & (selection-bit1);
        h1= plain-text-bit & (selection-bit0) & (selection-bit1);
        cipher-bit = e1|f1|g1|h1;
        return (cipher-bit);

}
```

8. Hardware Implementation

The hardware version of the CLU, previously shown in Figure 2, is FPGA-implemented. We have used Altera Quartus II 6.1 Web Edition, [20]. The average delay per byte was found to be 4.33 cycles per byte. Straightaway, if we use four CLUs in-parallel, this delay will be approximately equal to one cycle per byte. This proposed parallel configuration is shown in Figure 6.

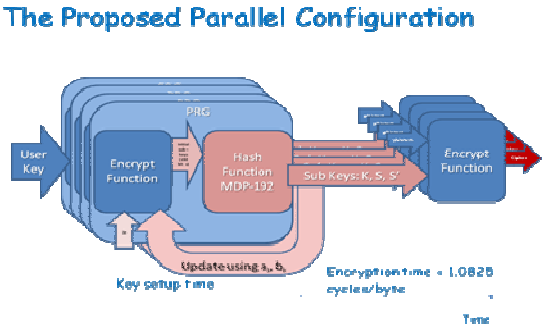

Figure 6. The proposed parallel configuration

A representative code of the Verilog file used to FPGA-implement the CLU is given by:

```
module metamorph (p1,k1,s0,s1,p2,c1);
        input p1,k1,s0,s1,p2;
        output c1;
        xor(a1,p1,k1);
        and(e1,a1,~s0,~s1);
        assign b1= ~p1;
        and(f1,b1,s0,~s1);
        and(g1,p2,~s0,s1);
        and(h1,p1,s0,s1);
        or(c1,e1,f1,g1,h1);
endmodule
```

9. Comparison with Chameleon Polymorphic Cipher

As seen from the given analysis and results, one can summarize the various characteristics of this cipher, when compared to Chameleon Polymorphic Cipher [Saeb09], as follows:

Table 2: A comparison between Stone Metamorphic Cipher and Chameleon Polymorphic Cipher

Cipher Characteristic	Chameleon-192 Polymorphic Cipher	Stone-192 Metamorphic Cipher
User key size	Variable	Variable
Sub-keys	192-bit K, S(K)	192-bit K, S(K), S'(K)
Estimated maximum delay per byte	10 cycles/byte	6 cycles/byte
Estimated average delay per byte	9.1 cycles/byte	4.3 cycles/byte
PRG (Sub-key Generation)	One-way Function	One-way cascaded with the Encryption Function
Structural	Sequential: Sel-1, ROT, Sel-0	Concurrent: XOR, ROT, INV, NOP
Number of rounds	Variable (key-dependent with minimum equal to 5 rounds)	Variable (key-dependent with minimum equal to 8 rounds)
Algorithm Template	Yes (key changes operation parameters)	No (key selects operations)
Parallelizable	Yes (some sequential operations)	Yes (operations are selected concurrently)
Security	Secure	Improved Security (pseudo random sequence of operations and more secure PRG)

10. Security Analysis

One claims that differential cryptanalysis, linear cryptanalysis, Interpolation attack, partial key guessing attacks, and side-channel attacks, barely apply in this metamorphic cipher. The pseudo random selection of operations provides the metamorphic nature of the cipher. This, in turn, hides most statistical 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 completely different forms (meta-forms) of the cipher. Even the cipher designer cannot predict in advance what these forms are. It can be easily shown that the probability of guessing the correct sequence of operations is of the order of $\frac{1}{2^{2wN}}$, where w is the word size and N is the number of rounds. That is for, say, a word size of 8 bits, the probability of guessing this word only is $\frac{1}{2^{16N}}$. For a block size of 64 bits, this probability is $\frac{1}{2^{128N}}$. Consequently, statistical analysis is not adequate to link the plain text to the cipher text. With different user keys, we end up with a different "morph" of the cipher; therefore, it is totally infeasible to launch attacks by varying keys or parts of the key. The only option left to the cryptanalyst is to attack the key itself. To thwart this type of attacks, we have used the encryption function as a first stage in a cascade of the encryption function and the one-way function. Regarding the key collision probability, it was shown in section 4 that the key collision probability is negligible when a 192-bit hash is applied. Moreover, the cryptanalyst has a negligible 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. Depending on the word or the block size required, 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. Undeniably, using the same encryption process and sub-keys for each block is a disadvantage from a security point of view. Still, this is exactly the same issue with block ciphers in general. The advantage obtained from such a configuration, similarly to block ciphers, is saving memory and communication bandwidth on the chip and the channel levels. The pseudo random selection of operations and the key-dependent number of rotations provide a barrier against pattern leakage and block replay attacks. These attacks are quite frequent in multi-media applications. Using ECB mode, when encrypting images with conventional ciphers, a great deal of the structure of the original image is preserved [3]. This contributes to the problem of block replay. However, the selective operations allow the cipher to encrypt images with no traces of the original image. This is a major advantage of the Stone Metamorphic Cipher bit-level operations when applied to multimedia files.

11. Summary & Conclusions

We have presented a metamorphic cipher that is altogether key-dependent. The four bit-balanced operations are pseudo-randomly selected. Known statistical attacks are barely applicable to crypt-analyze this type of ciphers. The proposed simple structure, based on the crypto logic unit CLU, can be easily parallelized using multi-threading superscalar processors or FPGA-based hardware implementations. This presented CLU can be viewed as a nonlinearity-associated filtering of the data and key streams. The PRG, constructed from a cascade of the encryption function and the one-way hash function, provides the required security against known key attacks. On the other hand, it easily allows the replacement of the hash function if successfully attacked. The cipher is well-adapted for use in multi-media applications. We trust that this approach will pave the way for key-driven encryption rather than simply using the key for sub-key generation.

Appendix A: The truth table of the CLU

P_i	K_i	$S'_1 S'_0 \rightarrow P_j$	S_1	S_0	OP	C_i
0	0	0	0	0	**XOR**	**0**
0	0	0	0	1	**INV**	**1**
0	0	0	1	0	**ROR**	**0**
0	0	0	1	1	**NOP**	**0**
0	0	1	0	0	**XOR**	**0**
0	0	1	0	1	**INV**	**1**
0	0	1	1	0	**ROR**	**1**
0	0	1	1	1	**NOP**	**0**
0	1	0	0	0	**XOR**	**1**
0	1	0	0	1	**INV**	**1**
0	1	0	1	0	**ROR**	**0**
0	1	0	1	1	**NOP**	**0**
0	1	1	0	0	**XOR**	**1**
0	1	1	0	1	**INV**	**1**
0	1	1	1	0	**ROR**	**1**
0	1	1	1	1	**NOP**	**0**
1	0	0	0	0	**XOR**	**1**
1	0	0	0	1	**INV**	**0**
1	0	0	1	0	**ROR**	**0**
1	0	0	1	1	**NOP**	**1**
1	0	1	0	0	**XOR**	**1**
1	0	1	0	1	**INV**	**0**
1	0	1	1	0	**ROR**	**1**
1	0	1	1	1	**NOP**	**1**
1	1	0	0	0	**XOR**	**0**
1	1	0	0	1	**INV**	**0**
1	1	0	1	0	**ROR**	**0**
1	1	0	1	1	**NOP**	**1**
1	1	1	0	0	**XOR**	**0**
1	1	1	0	1	**INV**	**0**
1	1	1	1	0	**ROR**	**1**
1	1	1	1	1	**NOP**	**1**

References

[1] Magdy Saeb, "The Chameleon Cipher-192: A Polymorphic Cipher," SECRYPT2009, International Conference on Security & Cryptography, Milan, Italy; 7-10 July, 2009.

[2] Auguste Kerckhoffs, "La cryptographie militaire," Journal des sciences militaire, vol. IX, pp. 5-83, Jan. 1883, pp.161-191, Feb. 1883.

[3] Swenson, C., Modern Cryptanalysis; Techniques for Advanced Code Breaking, Wiley Pub. Inc., 2008.

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

[5] Massey, J. L., "On Probabilistic Encipherment," IEEE. Information Theory Workshop, Bellagio, Italy, 1987.

[6] Massey, J.L., "Some Applications of Source Coding in Cryptography," European Transactions on Telecommunications, vol. 5, No. 4, pp.7/421-15/429, 1994.

[7] Rogaway, P., Coppersmith, D., "A Software-oriented Encryption Algorithm," Fast Software Encryption Cambridge Security workshop Proceedings, Springer-Verlag, pages 56-63, 1994.

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

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

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

[11] Rivest, R.L., "The MD5 Message Digest Algorithm," RFC 1321, 1992.

[12] Hans Dobbertin, Antoon Bosselaers, Bart Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," Fast Software Encryption, LNCS 1039, Springer-Verlag, pages 71–82, 1996.

[13] Magdy Saeb, "Design & Implementation of the Message Digest Procedures MDP-192 and MDP-384," ICCCIS2009, International Conference on Cryptography, Coding & Information Security, Paris, June 24-26, 2009.

[14] Rivest, R.L., "The MD4 Message Digest Algorithm," RFC 1186, 1990.

[15] Ueli Maurer, James Massey, "Cascade Ciphers: The Importance of Being First," Journal of Cryptography, vol. 6, no. 1, pp. 55-61, 1993.

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

[17] Erik Zenner, On Cryptographic Properties of LFSR-based Pseudorandom Generators, Ph.D. Dissertation, University of Mannheim, Germany, 2004.

[18] Erik Zenner, "Why IV Setup for Stream Ciphers is Difficult," Dagstuhl Seminar Proceedings 07021, Symmetric Cryptography, March14, 2007.

[19] Michael Welschenbach, Cryptography in C and C++, Apress, 2005.

[20] S. Brown, Z. Vranesic, Fundamental of Digital Logic with Verilog Design, McGraw-Hill International Edition, 2008.

Author Profile

Magdy Saeb received the BSEE. School of Engineering, Cairo University, in 1974; the MSEE. and Ph.D. in Electrical & Computer Engineering, 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. Currently, he is a professor in the Department of Computer Engineering, Arab Academy for Science, Technology & Maritime Transport, Alexandria, Egypt, (on leave) to Malaysian Institute of Microelectronic Systems (MIMOS), Kuala Lumpur, Malaysia. His current research interests include Cryptography, FPGA Implementations of Cryptography and Steganography Data Security Techniques, Encryption Processors, Computer Network Reliability, Mobile Agent Security. www.magdysaeb.net.