JEA K-128: A novel Encryption Algorithm Using VHDL

JAMAL N. BANI SALAMEH
Computer Engineering Department
Mu'tah University
Mu'tah – Karak, P.O.Box (7)
Jordan
jbanisal@mutah.edu.jo

Abstract: - Data security is an important issue in computer networks and cryptographic algorithms are essential parts in network security. In this paper a new block- ciphering algorithm, JEA K-128 (for Jamal Encryption Algorithm with a Key of length 128 bits) is described. JEA K-128 is a symmetric block cipher suitable for hardware or software implementations. JEA K-128 has a 64-bit word size, 4-rounds, and 128-bit is the length of the secret key. New cryptographic features in our work include the use of many successive XORing for the plaintext with sub-keys, and the use of multiple multiplexers in sub-keys generator. The main goal of designing JEA K-128 algorithm is to reach the condition of having almost every bit of the ciphertext depend on every bit of the plaintext and every bit of the key as quickly as possible. Simulation study shows that JEA K-128 gives a strong Avalanche effect, when there is a change in one bit of the plaintext or one bit of the key; almost all bits in the ciphertext were changed. All code**s** for our algorithm were captured by using VHDL, with structured description logic. The reason for choosing VHDL is its suitability for hardware implementation. The design principles of JEA K-128 are given together with results and analyses to define the encryption algorithm precisely.

Key-words:- Data encryption, block cipher, cryptography, key generator, VHDL implementation.

1 Introduction

Symmetric-key block ciphers have long been used as a fundamental cryptographic element for providing information security. Although they are primarily designed for providing data confidentiality, their versatility allows them to serve as a main component in the construction of many cryptographic systems such as pseudorandom number generators, message authentication protocols, stream ciphers, and hash functions. There are many symmetric-key block ciphers which offer different levels of security, flexibility, and efficiency. Among the many symmetric-key block ciphers currently available, some (such as DES, RC5, CAST, Blowfish, FEAL, SAFER, and IDEA) have received the greatest practical interest [1-6].

Most symmetric-key block ciphers (such as DES, RC5, CAST, and Blowfish) are based on a "Feistel" network construct and a "round function". A Feistel cipher involves dividing the plaintext into two halves and repeatedly applying a round function to the data for some number of rounds, where in each round using the round function and a key, the left half is transformed based on the right half and then the right half is transformed based on the modified left half. The round function provides a basic encryption mechanism by composing several simple linear and nonlinear operations such as exclusive-or, substitution, permutation, and modular arithmetic [7, 8]. Different round functions provide different levels of security, efficiency, and flexibility. The strength of a Feistel cipher depends heavily on the degree of diffusion and non-linearity properties provided by the round function. Many ciphers (such as DES and CAST) base their round functions on a construct called a "substitution box" (s-box) as a source of diffusion and non-linearity. Some ciphers (such as RC5) use bit-wise data-dependent rotations and a few other ciphers (such as IDEA) use multiplication in their round functions for diffusion.

In this paper, we present a novel symmetric-key block cipher, called JEA K-128, with a block size of 64 bits and a key size of 128 bits. The motivation for this work is the need to design our own cryptographic algorithm plus sub-keys generator. This algorithm plus the key generator will form a kind of encryption/decryption box. This box could be connected to the output of the transmitter and to the input of the receiver. A secret key should be known by all parties that use communication system. Before the transmitter start sending he needs to use the secret key to run the generator to generate the sub-keys to be used in the encryption algorithm to encrypt all information that goes through the

encryption box. In the other side, the receiver needs to do same procedure: use the secret key, then run the key generator; generate the required sub-keys; use them in the decryption algorithm to reconfigure back the original plaintext. Each time the transmitter or the receiver got the same secret key and runs the generator, this should generate the same sub-keys that will be used in the encryption/decryption box. When they change the secret key they got a different group of sub-keys.

One of the main problems in using the symmetric cryptography is key management. In this case you have the choice weather to send the secret key to the other party by hand or send it through the channel (this is too risky) [7]. By using the proposed technique; there will be no problem to send the secret key through the channel. If an intruder got this key; he still needs to get the encryption/decryption box that holds the algorithm plus the key generation to break this communication system.

In this paper; besides the JEA K-128 algorithm we also introduce a novel technique for sub-keys generation.

Many pseudorandom generators were proposed in the literature that depends on some good statistical properties of the LFSR (left feedback shift register) sequences. The nearest work to the proposed design is Geffe's generator [9]. This key stream generator uses three LFSRs, combined in a nonlinear manner. Two of the LFSR's are inputs into a multiplexer, and the third LFSR controls the output of the multiplexer. Another scheme uses a multiplxer to combine two LFSRs is the Jennings generator [10]. The multiplxer, controller by the LFSR-1, selects 1-bit of LFSR-2 for each output bit. There is also a function that maps the output of LFSR-2 to the input of the multiplexer.

The sub-keys generator that is proposed in this work could be used in any communication system to generate pseudorandom sub-keys given the main secret key as the input for this generator. The size of main secret key as well as the size of the sub-key are variable and could be used with any length to suit any system. For the JEA K-128 algorithm we choose the size of the main key to have 128-bit and the size of the sub-keys have a length of 16-bit.

All codes for JEA K-128 algorithm was captured by VHDL [11, 12], with structured description logic. The reason for choosing VHDL is its suitability for hardware implementation. The VHDL codes were synthesized using the MAX+Plus II Simulator [13].

The rest of the paper is organized as follows: Section 2 describes the JEA K-128 cryptographic algorithm. Section 3 discusses the results and checks the performance of the algorithm. Finally, section 4 provides some concluding remarks.

2 The JEA K-128 Cryptographic Algorithm

JEA K-128 is a block ciphering algorithm; it operates on 64-bit plaintext blocks. It has four different rounds. The key is 128 bits long, which makes it practically immune to brute-force attacks.

Overview of JEA K-128

A 64-bit block of plaintext goes in one end of the algorithm and a 64-bit block of ciphertext comes out the other end. JEA K-128 is a symmetric algorithm; the same algorithm and key are used for both encryption and decryption. As with all the other block ciphers, JEA K-128 uses both confusion and diffusion. The design philosophy behind the algorithm is one of mixing XOR operations. These operations can be viewed as JEA K-128's S-box, and they are easily implemented in both hardware and software.

JEA K-128 was designed in accordance with Shannon's principles of confusion and diffusion for obtaining security in secret-key ciphers [14]. When a round subkeys are mixed with the plaintext within the round; this acts like a nonlinear combination with respect to the subsequent transformations in the nonlinear layer and in the linear layer. This gives the cipher the confusion required to make the statistics of the ciphertext depend in a complicated way on the statistics of the plaintext; provided that small changes diffuse quickly through the cipher. To guarantee this diffusion in JEA K-128, we spread the redundancy of the plaintext out over the ciphertext. A cryptanalyst looking for those redundancies will have a harder time finding them.

Detailed description of JEA K-128

In this section, we will introduce the JEA K-128 algorithm in some details. Fig. 1 shows a block diagram of JEA K-128 algorithm (encryption). The 64-bit input data is divided into four 16-bit blocks. These four blocks become the input to the first round of the algorithm. The 128-bit main key feeds the key generator which generates the required sub-keys. JEA K-128 algorithm needs 23 different 16-bit sub-keys that will be used in all rounds of the algorithm. More details about the key generator will be discussed later. JEA K-128 is based upon a basic

function, which is iterated four times. The first iteration (round) operates on four input 16-bit plaintext blocks and the successive iterations also operate on the 16-bit blocks that come from the previous iteration. In each round, the 16-bit plaintext blocks are XORed with the 16-bit sub-keys. There are multiple XOR operations in each round. After the last iteration, a final transform step produces the 64-bit cipher block.

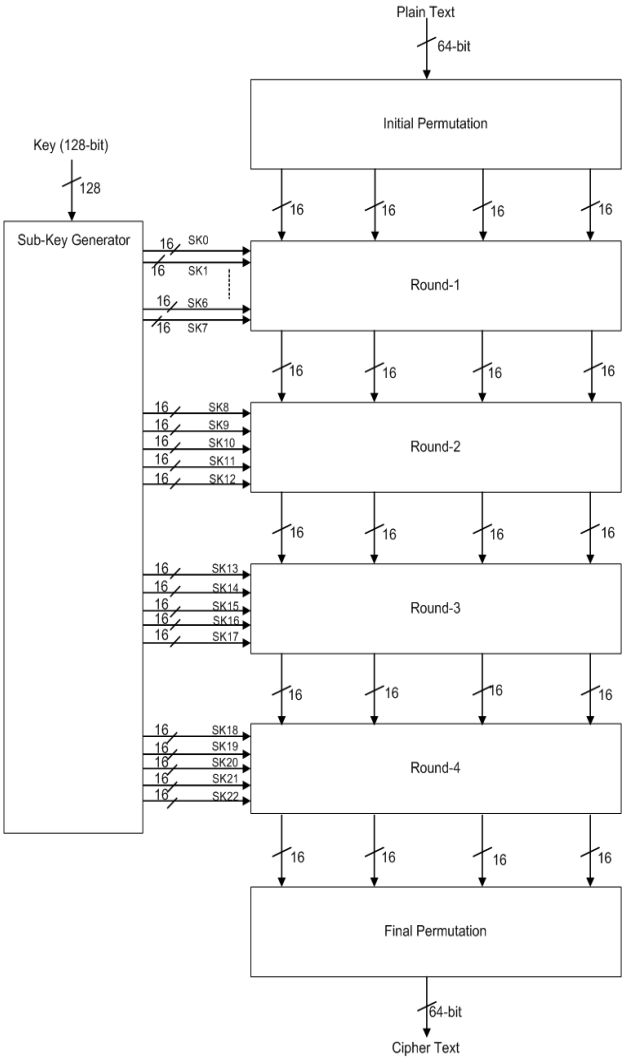

Figure 1: Block Diagram of JEA K-128 Algorithm (Encryption)

A schematic diagram (Encryption) for round 1 is shown in Fig. 2. The inputs for this round are four 16-bit blocks that were received from the initial permutation stage and eight 16-bit different sub-keys that were received from the main key generator. As we see in the figure each 16-bit block of text and each 16-bit sub-key are divided into four 4-bit blocks. All 4-bit text blocks are XORed many times with identical 4-bit blocks of the sub-keys. At the end of this round, the 4-bit block outputs are assembled again into 16-bit blocks of cipher text that form the inputs for round 2.

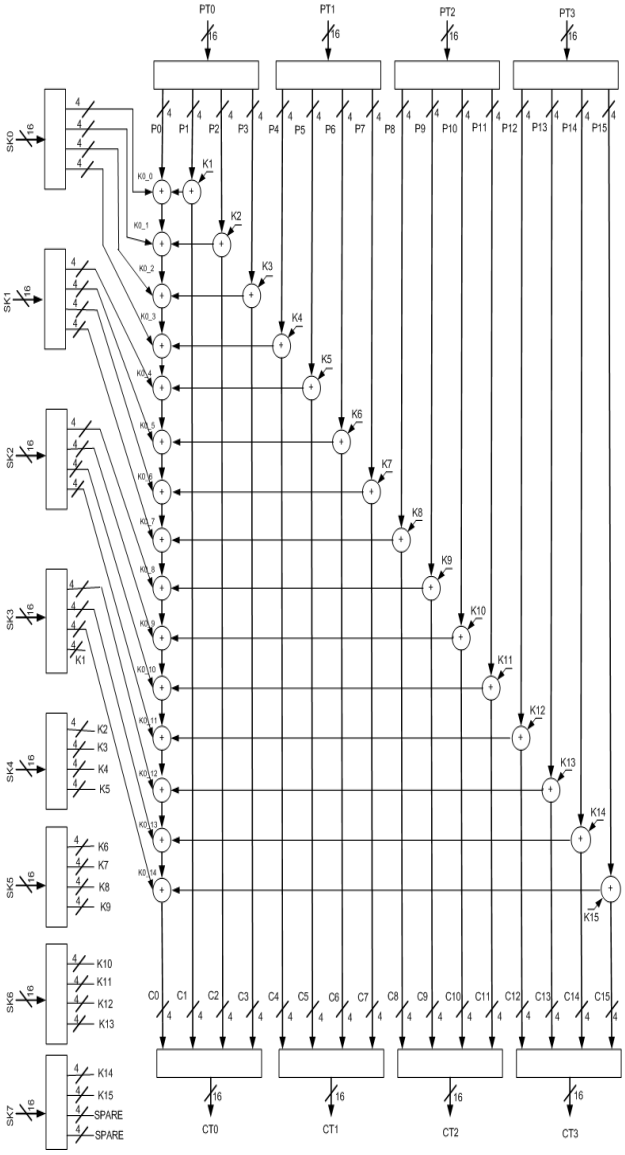

Figure 2: Round 1 Schematic Diagram (Encryption)

Fig. 3 shows a schematic diagram (encryption) for round 2. The inputs for round 2 are four 16-bit blocks that come from round 1 plus five 16-bit sub-keys that are generated by the key generator. Almost the same procedure as in round 1 occurs here in this round; all 4-bit text blocks are XORed many times with identical 4-bit blocks of the sub-keys. At the end of this round, the 4-bit block outputs are assembled again in a 16-bit blocks of cipher text that form the inputs for round 3.

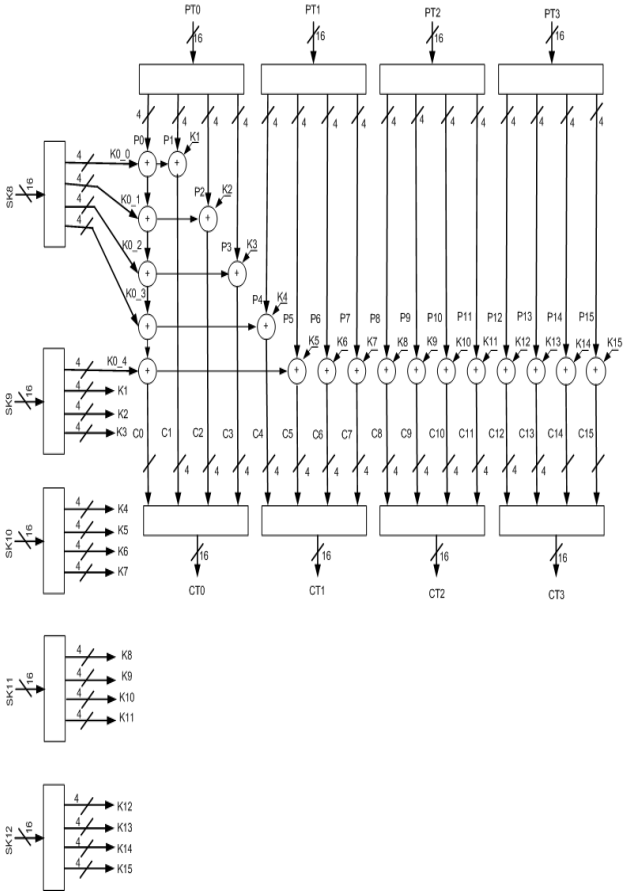

Figure 3: Round 2 Schematic Diagram (Encryption)

The same scenario is repeated in round 3 and round 4 with different mixing scheme between text-blocks and sub-keys. Fig. 4 shows a schematic diagram (encryption) for round 3 and Fig. 5 shows a schematic diagram (encryption) for round 4. Note that round 3 and round 4 also needs five 16-bit sub-keys each.

As a final note on the design process of the encoder; which could be seen in the schematics above; each round in our algorithm consists of a key-dependent permutation, a key and data-dependent substitution and all operations are EX-Ors on 4-bit words.

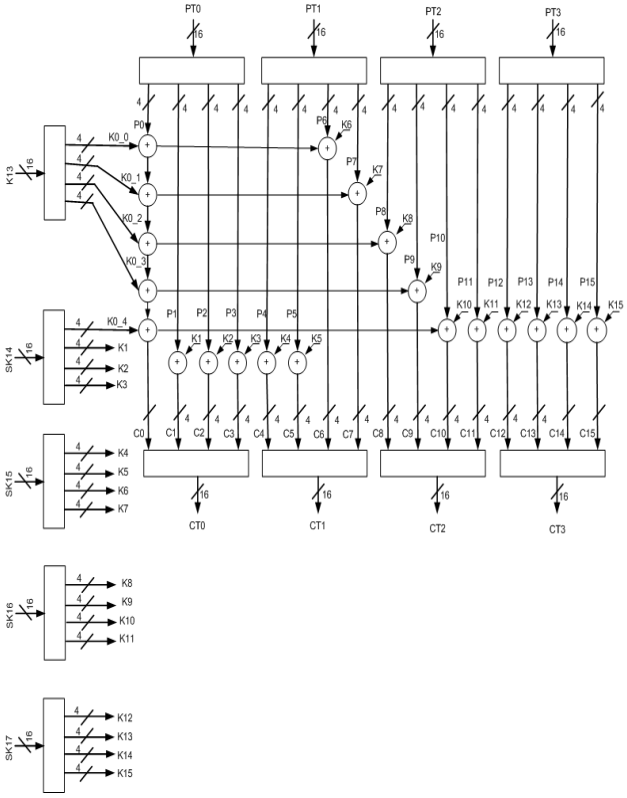

Figure 4: Round 3 Schematic Diagram (Encryption)

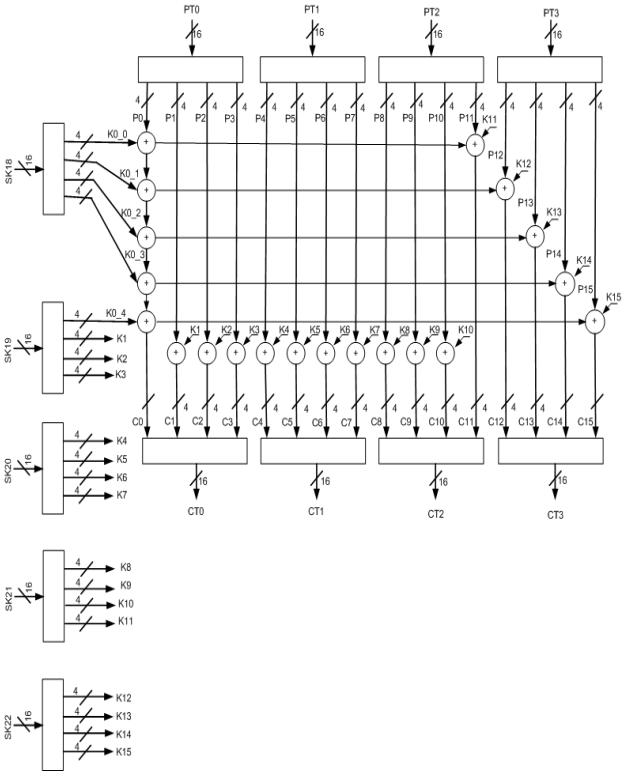

Figure 5: Round 4 Schematic Diagram (Encryption)

Decryption for JEA K-128 is relatively straightforward. Ironically, decryption works in the same algorithmic direction as encryption beginning with the ciphertext as input.

A block diagram for the decryption process is shown in Fig. 6. As we see in the figure, the same 128-bit secret key is used as input to the key generator, which generate the same sub-keys as in the encoder side. However, as expected, the sub-keys are used in reverse order. The 64-bit block of ciphertext goes in one end of the algorithm, and then the algorithm runs in the reverse direction, which reconfigures the 64-bit of plaintext at the end.

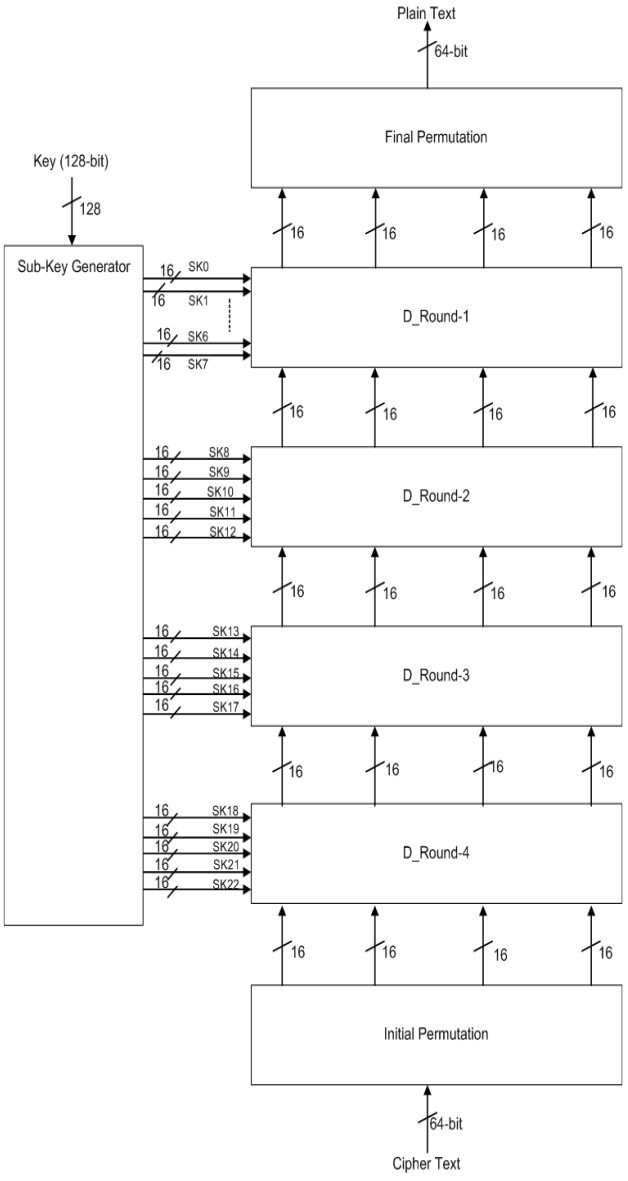

Figure 6: Block Diagram of JEA K-128 Algorithm (Decryption)

The schematic diagrams for all rounds in the decryption process are exactly the same as in the encryption process but in the reverse direction. Figure 7 shows a sample of this process: A schematic diagram for decryption in round 1.

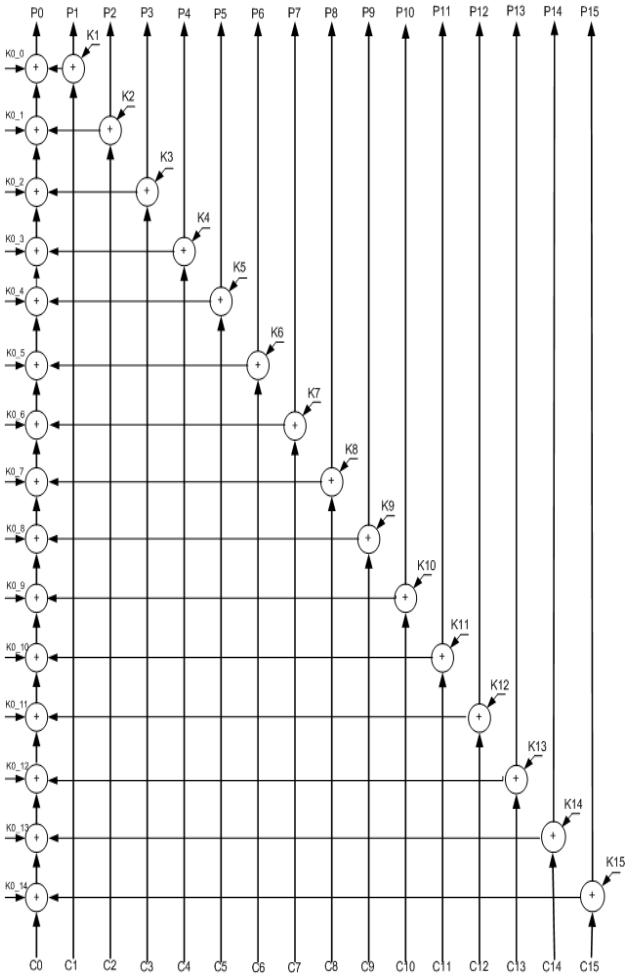

Figure 7: Round 1 Schematic Diagram (Decryption)

2.3 The Sub-keys Generation Process

JEA K-128 uses a large number of sub-keys. These sub-keys must be pre-computed before any data encryption or decryption. The key array consists of twenty three 16-bit sub-keys (Sk0, Sk1,…....., Sk22). The procedure for generating the sub-keys for JEA K-128 is indicated in the block diagram that is shown in Figure 8. This block diagram shows how the main key K (128-bit) is used to generate the 16-bit sub-keys that are required within the 4-rounds of JEA K-128 algorithm.

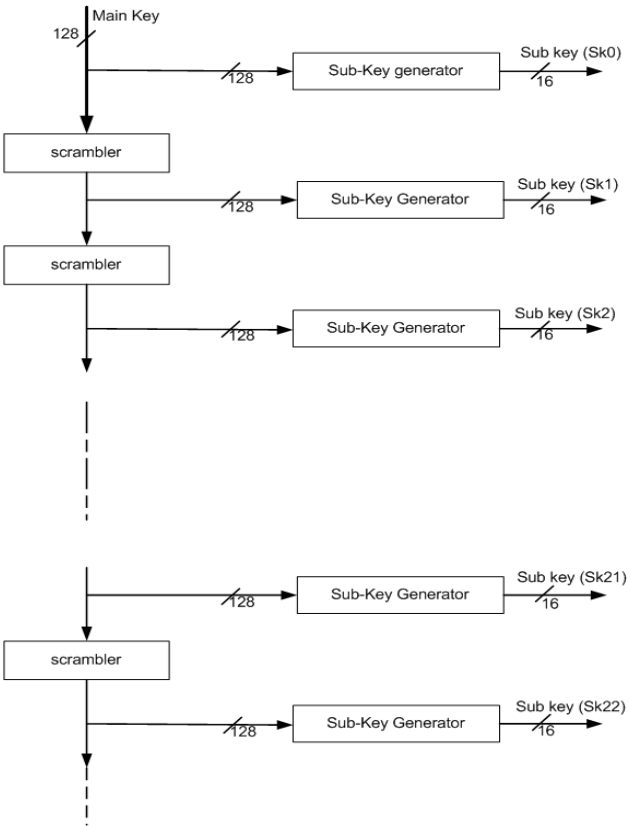

Figure 8: Sub-Key Generation Block Diagram

Note that in the generation process, the original key (K) is bit-wise rotated by 1-bit to the left between the process of generating a new sub-key. The rotation process is shown in Fig. 9 and it known as the scrambler in the previous block diagram.

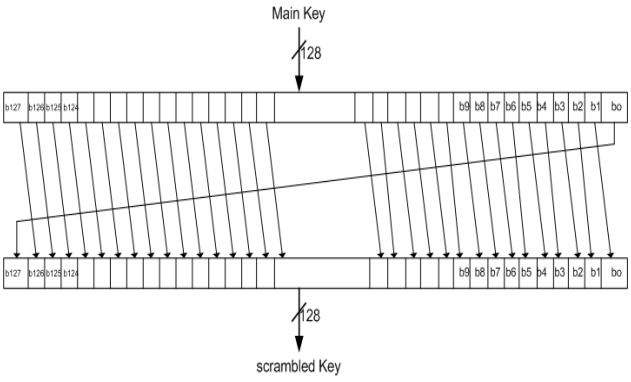

Figure 9: Scrambler

Figure 10 shows a schematic diagram for the sub-key generation. Here, we introduce a novel idea of using multiple multiplexers (MUX's) in choosing random sub-keys among the main key.

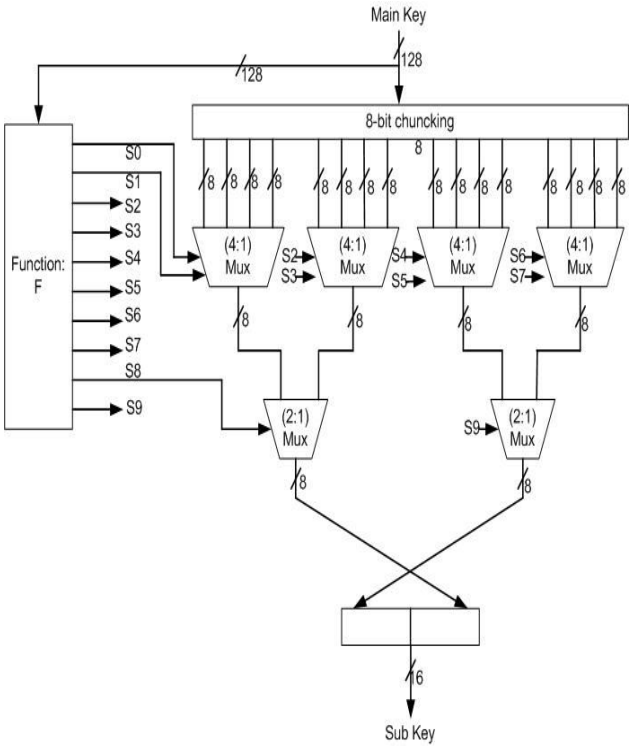

Figure 10: Sub-Key Generator Schematic Diagram

As we known that the MUX has multiple inputs and one output, one of the inputs will be active on the output depending on the selection lines. Our selection lines that are needed for the MUX's chosen upon a simple scheme from the main key; this scheme could be changed from time to time. As seen in the figure we need 10 selection lines (S0, S1, ….., S9) that will be used for choosing sub-keys.
Those selection lines were chosen according to a simple algorithm that is shown in Fig. 11.

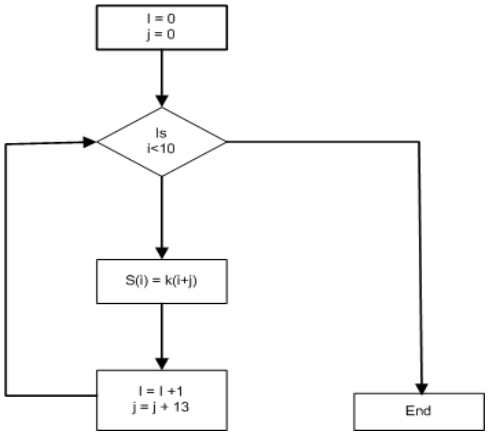

Figure 11: Algorithm to Choose the Selection Lines

The previous algorithm is considered as one of the main parts in Fig. 10 that is labeled (Function F). Each time we need to generate a sub-key, the function F should run to determine values for all selection lines.

3 Results and Discussion

The change in number of bits in the cipher text whenever there is a change in one bit of the plain text or one bit of key is called Avalanche effect [7, 15]. A desirable feature of any encryption algorithm is that a small change in either the plaintext or the key should produce a significant change in the ciphertext. If the changes are small, this might provide a way to reduce the size of the plaintext or key space to be searched and hence makes the cryptanalysis very easy. So, in order to say that any cryptographic algorithm is secure, it should exhibit strong avalanche effect**.**

JEA K-128 algorithm is designed to reach the condition of having almost every bit of the ciphertext depend on every bit of the plaintext and every bit of the key as quickly as possible. Here we provide some performance measurements for JEA K-128 encryption/decryption operations. Most of the experiments that was done for performance evaluation aimed to check the Avalanche effect.

The main goal of the first test is to make sure that the decoder is able to recover the original plain text. First of all, both the key and the data were set then we run the simulation; the result for the encoder is shown in Fig. 12.

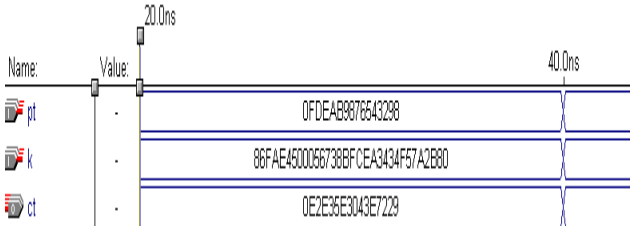

Figure 12: Data Setup for Encryption

After that, we provide the cipher text that we got out of the encoder along with the same key to the decoder. The decoder is able efficiently to recover the original plain text; this result is shown in Fig. 13.

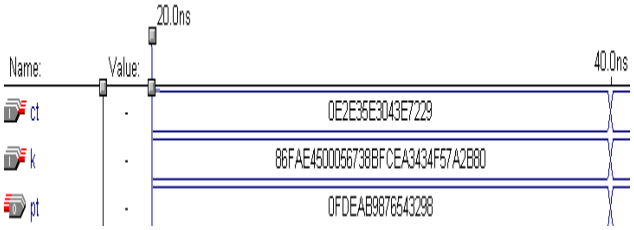

Figure 13: Data Setup for Decryption

In the second experiment we changed just 1-bit in the plaintext (bit-7) keeping the same key compared to experiment 1. After we run the simulator; we got the following result that is shown in Fig. 14.

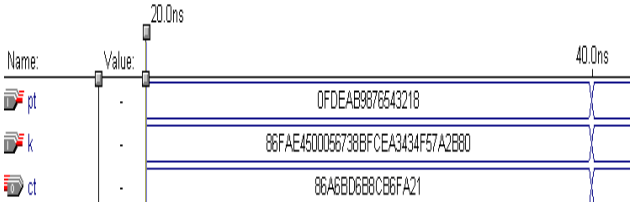

Figure 14: Data Setup for Encryption with 1-bit Change in the Plain Text

If we compared this result to the one in the first test; that is shown in Fig. 12; we notice that almost all bits in the cipher text changed. After that, we provide the cipher text that we got out of the encoder along with the same key to the decoder. The decoder is able efficiently to recover the original plain text; this result is shown in Fig. 15.

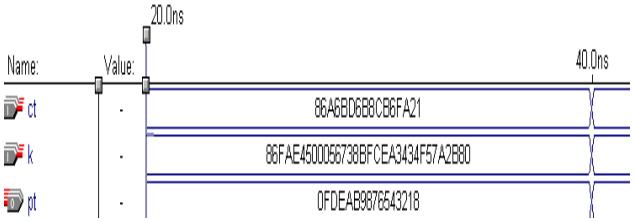

Figure 15: Data Setup for Decryption Changing 1-bit in the Plain Text

In the third experiment we changed just 1-bit in the key (bit-2) keeping the same plain text compared to experiment-2. After we run the simulator; we got the following result that is shown in Fig. 16. As we see from this figure changing 1-bit in the key affects in changing almost all bits in the cipher text; compared to Fig. 14.

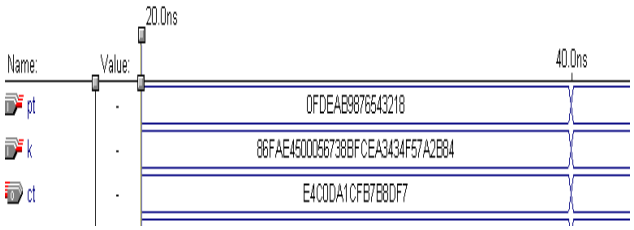

Figure 16: Data Setup for Encryption Changing 1-bit in the Key

After that, we provide the cipher text that we got out of the encoder along with the same key to the decoder. The decoder is able efficiently to recover the original plain text; this result is shown in Fig. 17.

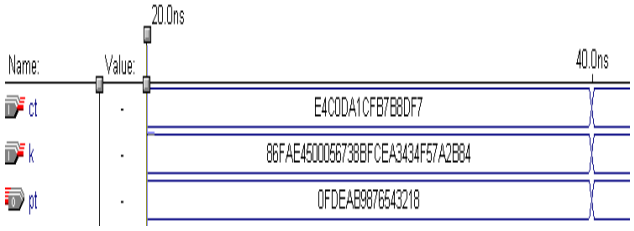

Figure 17: Data Setup for Decryption Changing 1-bit in the Key

We did one more test by changing just 1-bit in the key (bit-126) keeping the same plain text compared to the previous experiment. The result for the encoder is shown in Fig. 18. Also this experiment ensures that changing 1-bit in the key affects in changing almost all bits in the cipher text.

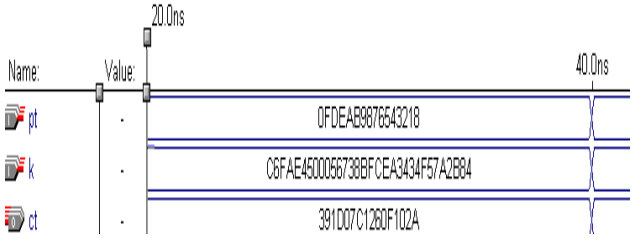

Figure 18: Data Setup for Encryption Changing 1-bit in the Key

The result for the decoder for this experiment is shown in Fig. 19. Also at this time the decoder is able to recover the original plaintext.

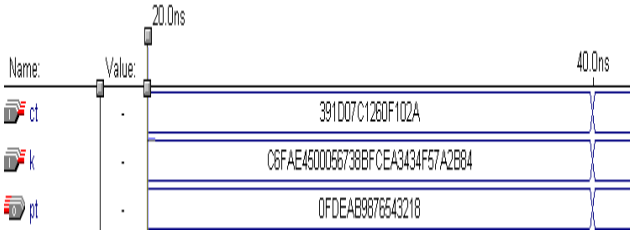

Figure 19: Data Setup for Decryption Changing 1-bit in the key

Figure 20 shows a test for generation the required sub-keys for our algorithm. As we see in this figure, the key generator gives random different sub-keys each time we provide a new secret key. The number of sub-keys and the size of each sub-key could be changed as needed. However, each time we provide a new main key we got totally different sub-keys.

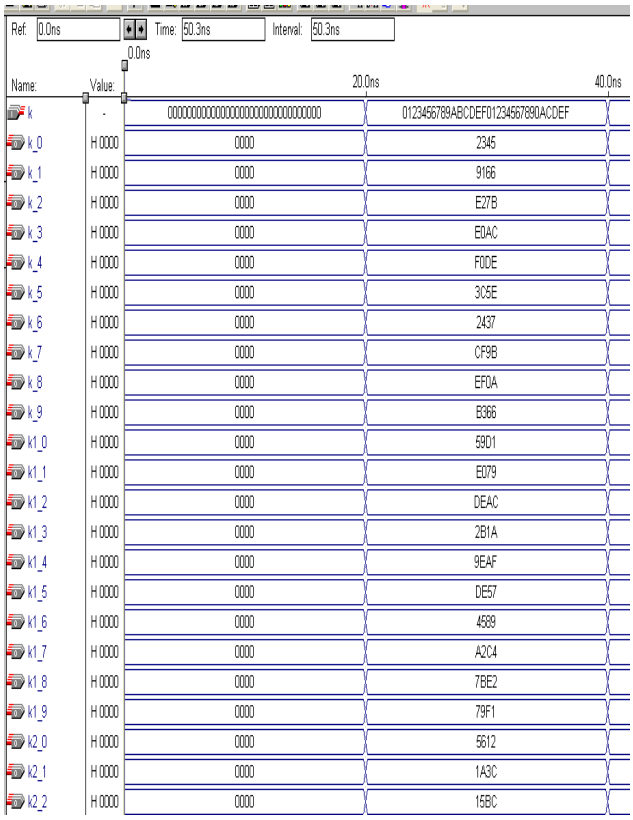

Figure 20: Sub-keys Generation

To make sure that the generator works as required, we provided another 128-bit key then we run the simulator; as seen in Fig. 21; the generator is able to give another pseudorandom different group of sub-keys.

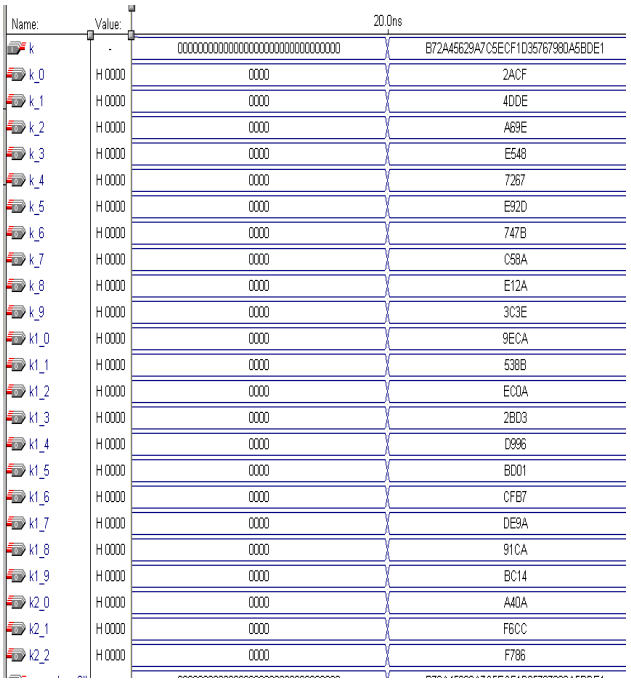

Figure 21. Sub-keys Generation-2

The same scenario is repeated when we provide the generator with another 128-bit key; it is able to generate a totally different group of sub-keys; this result in shown in Fig. 22.

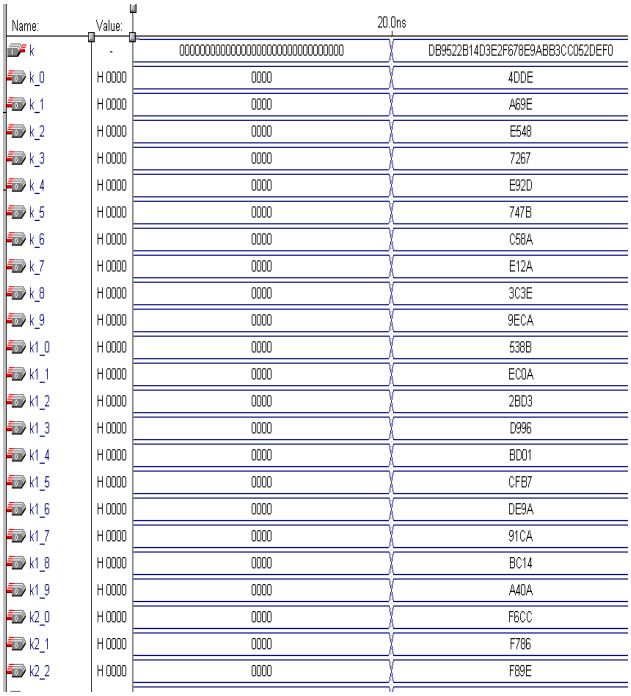

Figure 22. Sub-keys Generation-3

Sub-keys are independently chosen and their generation depends not only on the main key but also on the values for selection lines of all MUX's in the generator. The real goal in the design procedure for sub-keys generation is to a void man in the middle attacks. This generation process occurs just on both sides of the communication system; an attacker whose goal is to break the system needs not only the secret key but also the key generator which should be physically protected. There is no risk in sending the secret key through the channel, but we should encrypt that key before sending it.

4 Conclusions and Future Work

The number of rounds for thorough mixing and to ensure that every plaintext and every key bit affects almost every ciphertext bit proved to be 4-rounds. After 4-rounds the ciphertext was essentially a random function of every plaintext bit and every key bit. This is a good avalanche and sufficiently secure. That's the reason why we stopped after 4-rounds, but for more security and to prevent the algorithm from being attacked more rounds could be added. This will be done as a future work.

The JEA K-128 algorithm promises good mixing of the key and the plaintext for a scrambled ciphertext. The ciphertext is easy to be de-ciphered if the key is known. A brute force attack would take too long to break the system. Further work would include more thorough testing and analysis to get better S-boxes, and do some performance comparison with up-to-date block ciphering algorithms. Also, another future work will include hardware implementation for our work in a suitable FPGA.

References:

[1] R. L. Rivest, The RC5 Encryption Algorithm, *Dr. Dobb's Journal,* Vol. 20, No. 1, 1995, pp. 146-148.

[2] B. Schneier, The Blowfish Encryption Algorithm, *Dr. Dobb's Journal,* Vol. 19, No. 4, 1994, pp. 38- 40.

[3] National Bureau of Standards, *Data Encryption Standard*, FIPS PUB 46, 1977.

[4] J. L. Massey, SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm, *Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag,* 1994, pp.1-17.

[5] C. M. Adams, Constructing Symmetric Ciphers Using the CAST Design Procedure, *Design, Codes, and Cryptography,* Vol. 12, No. 3,1997, pp. 283-316.

[6] B. Schneier, The IDEA Encryption Algorithm, *Dr. Dobb's Journal,* Vol. 18, No. 13, 1990, pp. 50-56.

[7] B. Schneier, *Applied Cryptography,* John Wiley & Sons Inc, 1996.

[8] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography,* CRC Press, 1996.
[9] Geffe, P. R., How to protect data with ciphers that are really hard to break, *Electronics,* 1973, pp. 99-101.
[10] S.M. Jennings, Autocorrelation Function of the Multiplexed Sequence, *IEE Proceedings,* Vol. 131, No. 2, 1984, pp. 169-172.
[11] *IEEE Standard VHDL Language Reference Manual*, IEEE, 1993.
[12] Z. Navabi, *VHDL: Analysis and Modeling of Digital Systems,* McGraw-Hill, 1993.
[13] *Altera Max Plus II VHDL,* Altera Corporation. 1994.
[14] C.E. Shannon, Communication Theory of Secrecy Systems, *Bell System Tech. Jour.,* Vol. 28, 1949, pp. 656-715.
[15] A.G. Konheim, *Cryptograph: A Primer,* New York: John Wiley & Sons, 1981.