Design a Secure Hybrid Stream Cipher

Ali H. Kashmar[1, 2*], Eddie S. Ismail[1], Firdaus M. Hamzah[3], Haider F. Abdul Amir[4]

[1] School of Mathematical, Sciences Faculty of Science and Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor DE, Malaysia.
[2] College of Science, University of Baghdad, Baghdad, Iraq.
[3] Uint Pengajian Asas Kejuruteraan, Faculty Kejuruteraan and Alam Bina, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor DE, Malaysia.
[4] School of Science and Technology, Universiti Malaysia Sabah, 88400 Kota Kinabalu, Sabah, Malaysia.

* Corresponding author. Tel.: +60104361429; email: Kashmar992000@yahoo.dk
Manuscript submitted February 28, 2015; accepted June 5, 2015.
doi: 10.17706/ijapm.2015.5.3.153-166

Abstract: By employing the characteristics of the basic structure (keystream), Stream ciphers designers attempt to create algorithms that have advanced features from security and speed point of view. They take into consideration the state-of-the-art scientific and technical developments to design more advanced algorithm versions. This research proposes the design of a new efficient and secure stream cipher, named BloStream which proves to be more secure than conventional stream ciphers that commonly implement Exclusive-OR (XOR) for mixing. The proposed algorithm encompasses two major components. The first part involves the Pseudo Random Number Generator (PRNG), exhausting Rabbit algorithm. And the second part involves a nonlinear invertible round function (combiner), depending on Rijndael-like function algorithm, to perform the encryption/decryption processes. This new construction strengthens the weak XOR combiner. The proposed cipher is not only a random number generator but also a self-synchronizing stream cipher in such a way that the cipher text influences its internal functioning. The proposed algorithm utilizes 16-bytes secret key to encrypt the plaintext which is a multiple of 16-bytes up to 2^{64}bytes length. The evaluation of BloStream performance, in terms of implementation aspects and security properties as well as the statistical test for keystream and comparison with similar systems revealed that, BloStream was more efficient, faster, and securer than the conventional stream ciphers.

Key words: Stream ciphers, rabbit cipher, Rijndael-like function, combiner algorithm, PRNG.

1. Introduction

Stream ciphers are an important class of symmetric encryption algorithms. They encrypt individual characters or binary digits of a plaintext message one at a time, using an encryption transformation which varies with time. They are also more appropriate, and in some cases mandatory (e.g. in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Stream ciphers employ — in Shannon's terminology — confusion only [1], [2]. Their basic design philosophy is inspired by the Vernam (One-Time Pad) cipher, which encrypts by XO Ring the plaintext with a random key. The drawback of the Vernam cipher is that the keystream must possess a true random sequence, shared by the sender and the receiver, and it can only be used once [1]-[3]. A combiner is the heart of a stream cipher, which generally employs an 'additive' combiner such as XOR. Additive combiners have absolutely no strength at all; this means that, if an opponent somehow comes up with some plaintext which matches the cipher text, an additive combiner immediately reveals the confusion sequence. This allows the opponent to begin work on breaking the confusion sequence generator [4], [5].

An alternate approach to the design of a secure stream ciphers is to seek combining functions which can resist attack; such functions would act to conceal the pseudo-random sequence from analysis. Such cryptographic combining functions could be utilized to substitute the Vernam XOR combiner provided that they have an inverse. An improved combiner is intended to enhance the sophistication of cryptanalysis, making it more time consuming and expensive than simple combiners [6]. Dynamic substitution is a way to build a cryptographic combiner; it is not a complete cipher. It is deployed simply as a replacement for the weak XOR combiner, conventionally used in stream ciphers. The strength of dynamic substitution combiner can support the use of weaker but faster PRNG [5]. In today's world, this does not provide the required security. Also this does not represent a real blend (mixing together) between the plaintext and keystream. Instead, it is merely, a hiding process for the plaintext without affecting its actual bits directly. Therefore, if the confusion RNG is linear, with a small amount of state, the opponent has the capacity to try various sets of keystream values until the system is solved. But if the RNG has a large amount of state, selecting a set of correct random values from the larger set of possible keystream values (by many trials), then cryptanalysis will be very difficult.

Hence, to complicate the weak XOR combiner, this paper produces a new scheme, employing a hybrid concept between the block cipher round and stream cipher system. The proposed cipher scheme aims to utilize large block sizes in order to overcome attacks such as frequency analysis and cipher text/plaintext pairs. The new design is called BloStream algorithm, using key dependent S-boxes based on a Rijndael-like function [7]. The paper is followed in Section 2 by the investigation of the structure of BloStream. In Section 3 BloStream is compared with other similar ciphers. Security analyses with possible attacks are presented in Section 4. Randomness tests of the keystream bits are applied in Section 5. Conclusion is summed up in the last section.

2. BloStream

2.1. Algorithm Components

The motivation for the choice of the design in BloStream is summarized as follows:

According to the structure presented in Fig. 1, there are three basic parts in the proposed algorithm: 1) PRNG for keystream generation; 2) cipher feedback; and 3) combiner for encryption/decryption process. Each of these components is elaborated as follows.

2.1.1. PRNG

The PRNG in the BloStream algorithm is implemented depending on Rabbit algorithm, for the cryptanalysis of Rabbit does not reveal an attack better than exhaustive key search. The next advantage of Rabbit is its simplicity and small size which makes it suitable for implementations on processors with limited resources such as 8-bit processors. Furthermore, Rabbit was designed to be faster than commonly deployed ciphers (as illustrated in Table 1), justifying a key size of 128 bits for encrypting up to 2^64 blocks of plaintext; as such, it is suitable for both hardware and software implementation [8].

Table 1. Best Encryption Speed of Some Stream Ciphers on a Pentium IV

Algorithm name	Profile	Cycle/byte
Trivium	HW	8.10
Rabbit	SW & HW	9.46
Phelix	SW & HW	10.09
SOSEMANUK	SW	10.26
HC-256	SW	11.12
Dragon	SW	11.43

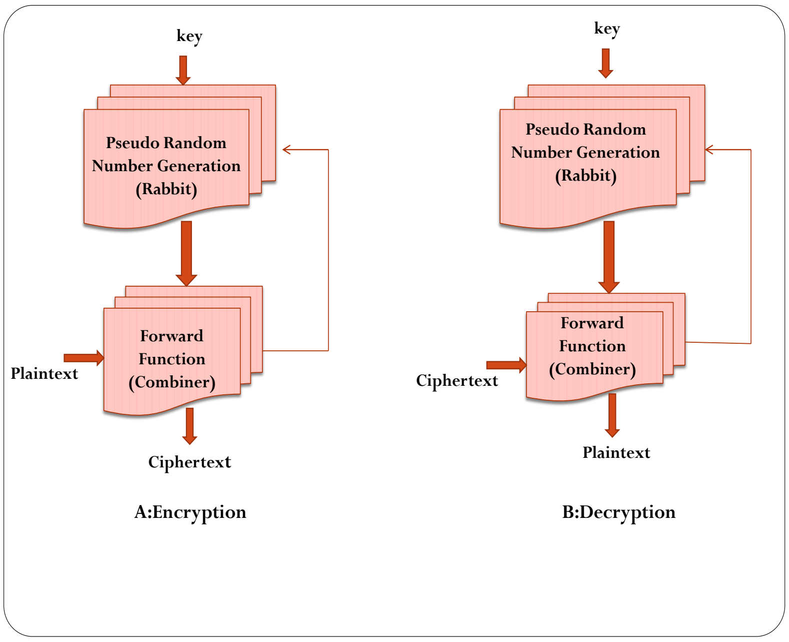
key
Pseudo Random Number Generation (Rabbit)
Forward Function (Combiner)
Plaintext
Ciphertext
A:Encryption
key
Pseudo Random Number Generation (Rabbit)
Forward Function (Combiner)
Ciphertext
Plaintext
B:Decryption

Fig. 1. Graphical illustration of BloStream algorithm.

Rabbit is characterized by a high performance in software with a measured encryption/decryption speed of 3.7 clock cycles per byte on a Pentium III processor [8]. The algorithm is initialized by expanding the 128-bit key into both the eight state variables and the eight counters in such a way that there is a one-to-one correspondence between the key and the initial state variables $X_{j,0}$, and the initial counter $C_{j,0}$, the key K [127...0] is divided into eight sub keys: K0=K[15...0], K1=K[31..16], ..., K7=K[127...112]. The state and counter variables are initialized from the sub keys. We use the following notation: $\oplus$ denotes logical XOR, & denotes logical AND, $\ll$and $\gg$denote left and right logical bit-wise shift, $⋘$ and $⋙$ denote left and right bit-wise rotation, and $\diamond$ denotes concatenation of two bit sequences. $A^{[g..h]}$ means bit number g through h of variable A. When numbering bits of variables, the least significant bit is denoted by 0. Hexadecimal numbers are prefixed by'0x'. Finally, we use integer notation for all variables and constants. The state and counter variables are initialized from the subkeys as follows:

$$x_{j,0} = \begin{cases} k_{(j+1 \bmod 8)} \diamond k_j & for\ j\ even \\ k_{(j+5 \bmod 8)} \diamond k_{(j+4\, mod\ 8)} & for\ j\ odd \end{cases}$$

$$c_{j,0} = \begin{cases} k_{(j+4 \bmod 8)} \diamond k_{(j+5 \bmod 8)} & for\ j\ even \\ k_j \diamond k_{(j+1 \bmod 8)} & for\ j\ odd \end{cases}$$

The system is iterated four times, according to the next-state function, to diminish correlations between bits in the key and bits in the internal state variables. Finally, the counter variables are re-initialized according to:

$$c_{j,4} = c_{j,4} \oplus x_{(j+4 \bmod 8),4} \qquad for\ all\ j$$

To prevent recovery of the key by inversion of the counter system. The core of the Rabbit algorithm is the iteration of the system defined by the following equations[9]:

$$x_{0,i+1} = g_{0,i} + (g_{7,i} <<< 16) + (g_{6,i} <<< 16),\ x_{1,i+1} = g_{1,i} + (g_{0,i} <<< 8) + g_{7,i}$$

$$x_{2,i+1} = g_{2,i} + (g_{1,i} <<< 16) + (g_{0,i} <<< 16)\ , \qquad x_{3,i+1} = g_{3,i} + (g_{2,i} <<< 8) + g_{1,i}$$

$$x_{4,i+1} = g_{4,i} + (g_{3,i} <<< 16) + (g_{2,i} <<< 16), \qquad x_{5,i+1} = g_{5,i} + (g_{4,i} <<< 8) + g_{3,i}$$

$$x_{6,i+1} = g_{6,i} + (g_{5,i} <<< 16) + (g_{4,i} <<< 16)\ , \qquad x_{7,i+1} = g_{7,i} + (g_{6,i} <<< 8) + g_{5,i}$$

$$g_{j,i} = \left((x_{j,i} + c_{j,i+1})^2 \oplus \left((x_{j,i} + c_{j,i+1})^2 >> 32 \right) \right) \mathrm{mod} 2^{32}$$

where all additions are modulo 2^{64}. This coupled system is illustrated in Fig. 2.

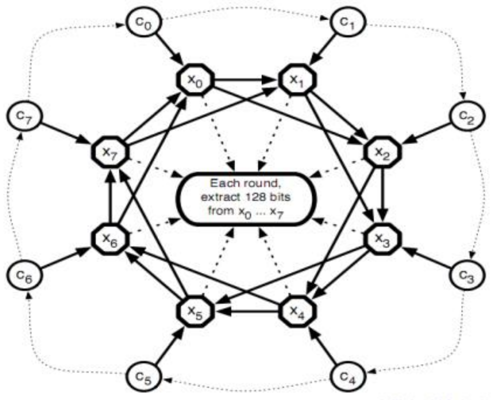

Fig. 2. Graphical illustration of the Rabbit algorithm.

Before the iteration the counters are incremented as illustrated in Fig. 2. The dynamics of the counters is defined in the following equations:

$$c_{0,i+1} = c_{0,i} + a_0 + \emptyset_{7,i} \ \mathrm{mod} 2^{32}, \ c_{1,i+1} = c_{1,i} + a_1 + \emptyset_{0,i+1} \mathrm{mod} 2^{32}$$

$$c_{2,i+1} = c_{2,i} + a_2 + \emptyset_{1,i+1} \mathrm{mod} 2^{32}, \ c_{3,i+1} = c_{3,i} + a_3 + \emptyset_{2,i+1} \mathrm{mod} 2^{32}$$

$$c_{4,i+1} = c_{4,i} + a_4 + \emptyset_{3,i+1} \ \mathrm{mod} 2^{32}, \ c_{5,i+1} = c_{5,i} + a_5 + \emptyset_{4,i+1} \mathrm{mod} 2^{32}$$

$$c_{6,i+1} = c_{6,i} + a_6 + \emptyset_{5,i+1} \mathrm{mod} 2^{32}, \ c_{7,i+1} = c_{7,i} + a_7 + \emptyset_{6,i+1} \mathrm{mod} 2^{32}$$

where the counter carry bit, $\emptyset_{j,i+1}$, is given by

$$\emptyset_{6,i+1} = \begin{cases} 1 \ if \ \ c_{0,i} + a_0 + \emptyset_{7,i} \ \geq \ 2^{32} \wedge j = 0 \\ 1 \ if \ \ c_{j,i} + a_j + \emptyset_{j-1,i+1} \ \geq \ 2^{32} \wedge j > 0 \\ 0 \qquad\qquad\qquad\qquad\qquad otherwise \end{cases}$$

Furthermore, the a_j constants are defined as:

$$a_0 = 0x4D34D34D, \ a_1 = 0xD34D34D3, \ a_2 = 0x34D34D34, \ a_3 = 0x4D34D34D$$

$$a_4 = 0xD34D34D3, \ a_5 = 0x34D34D34, \ a_6 = 0x4D34D34D, \ a_7 = 0xD34D34D3$$

After each iteration the output is extracted as follows:

$$S_i^{[15..0]} = x_{0,i}^{[15..0]} \oplus x_{5,i}^{[31..16]}, \ S_i^{[31..16]} = x_{0,i}^{[31..16]} \oplus x_{3,i}^{[15..0]}$$

$$S_i^{[47..32]} = x_{2,i}^{[15..0]} \oplus x_{7,i}^{[31..16]},\ S_i^{[63..48]} = x_{2,i}^{[15..0]} \oplus x_{5,i}^{[15..0]}$$

$$S_i^{[79..64]} = x_{4,i}^{[15..0]} \oplus x_{1,i}^{[31..16]},\ S_i^{[96..80]} = x_{4,i}^{[31..16]} \oplus x_{7,i}^{[31..16]}$$

$$S_i^{[111..96]} = x_{0,i}^{[15..0]} \oplus x_{3,i}^{[31..16]},\ S_i^{[127..112]} = x_{6,i}^{[31..16]} \oplus x_{1,i}^{[15..0]}$$

where S_i is the 128-bit keystream block at iteration i.

The extracted bits are XOR'ed with the plaintext/ciphertext to encrypt/decrypt:

$$C_i = P_i \oplus S_i$$

$$P_i = C_i \oplus S_i$$

where C_i and P_i denote the i^{th} ciphertext and plaintext blocks, respectively.

The chaotic scheme that is employed in the PRNG of the BloStream algorithm can achieve a higher level of complexity than classical binary systems due to its arithmetical properties. The deployed PRNG is a generator with a new type of design. It provides a strong non-linear mixing of the inner state between iterations. As opposed to almost all other designs currently available, it uses neither linear feedback shift registers nor S-boxes. The mechanism utilized in the PRNG of the proposed cipher is depicted in Fig. 3.

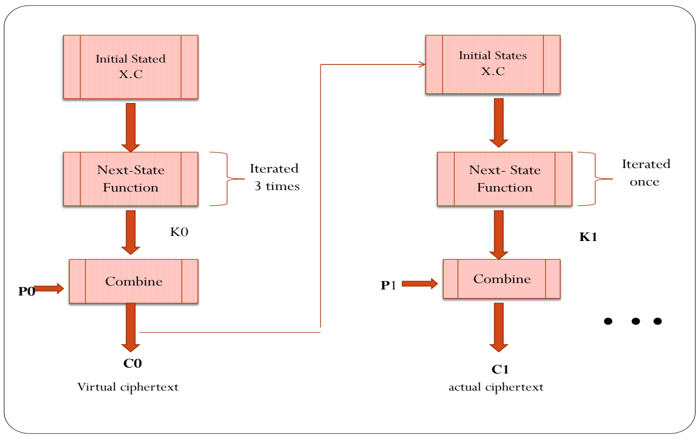

Fig. 3. Mechanism used in PRNG.

2.1.2. Cipher feedback

Stream ciphers never assume the use of a single key more than once. In some algorithms, Initial Vector (IV) is used as another input to ensure the ability to use the same key more than once. The proposed cipher, producing the cipher text C_0 that provides the possibility of deploying the same key for more encryptions as what the IV offers. This increases randomness in the internal states in addition to long distance that the plaintext passes through, when compared to IV. The time of this distance is considered at setup stage prior to encryption/decryption point, and the modern microprocessors overcome this problem. Fig. 4 illustrates algorithm feedback used in BloStream.

As it is illustrated in Fig. 4, the data value involved in the cipher text is feedback into the RNG internal state (in Rabbit cipher) by XO Ring with the four 32-bit counter variables $C_{j,i}$ of the internal state according to the even/odd condition of the fifth byte of the key stream. In this case, if the result of XO

Ring 5^{th} byte with 0 $\times$ 01 is 0, then $C_{j,i}[0]$, $C_{j,i}[2]$,$C_{j,i}[4]$,and $C_{j,i}[6]$ are modified with four bytes of cipher text, Else $C_{j,i}[1]$,$C_{j,i}[3]$,$C_{j,i}[5]$, and $C_{j,i}[7]$ are modified. The 4th, 8th, 12th and 16th bytes respectively are selected from the previous cipher text to modify $C_{j,i}$ counter variable and this will affect the new 16-bytes of the keystream generation in the next iteration which is utilized in the combiner function. The important advantage of the feedback is the capacity to share--or to contribute to — the round function in the encryption/decryption process and it cannot be isolated. In the proposed cipher, there is dependency between the feedback to the RNG and the output function (combiner) since the keystream being updated by the previous cipher text at the i^{th} iterations is immediately used as the new keystream. This dependency decreases the speed of encryption/decryption by about the time of modifying $C_{j,i}$ variables of the internal states. The proposed cipher implementation employs simple instructions which relatively constitute a small part of the overall computation. Also it does not deal with this dependency because its effect on the encryption speed is very limited on the Pentium 4 processor. The cipher text is feedback to the RNG instead of the combiner round function because the round function is used merely as an obscurity following XOR. In doing so, the trails of analyzing cipher text will be reduced. Also, the combined output cannot be used as contributor in the next processes.

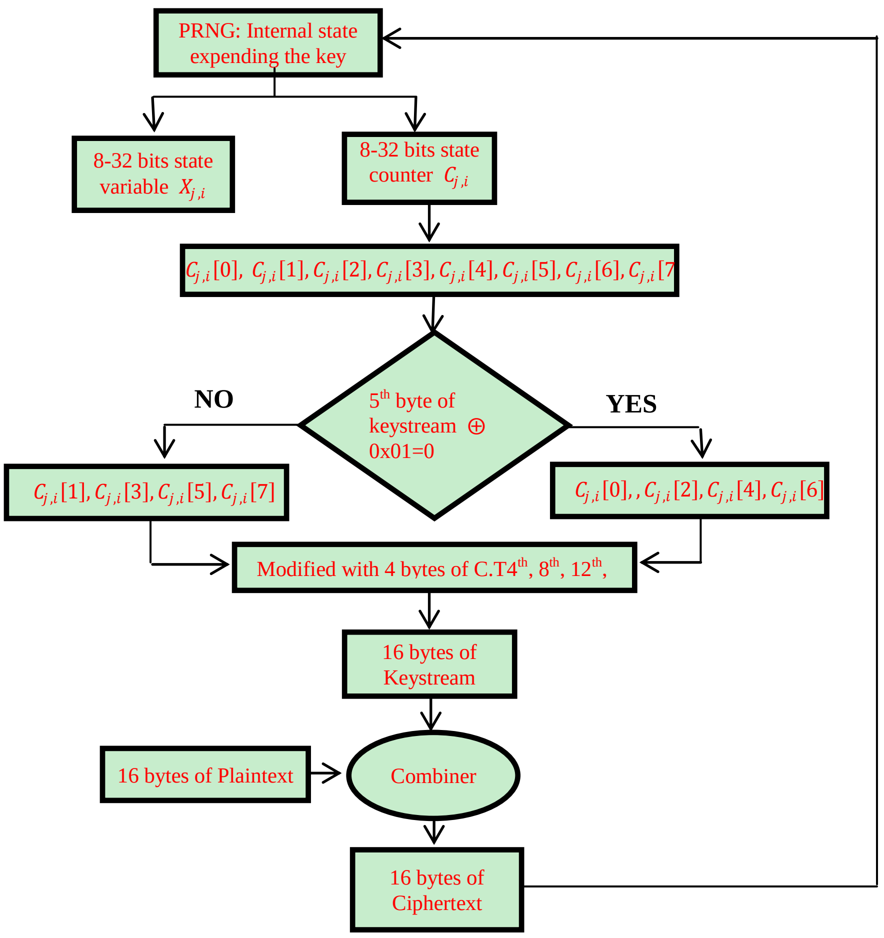
PRNG: Internal state expending the key
8-32 bits state variable $X_{j,i}$
8-32 bits state counter $C_{j,i}$
$C_{j,i}[0]$, $C_{j,i}[1]$, $C_{j,i}[2]$, $C_{j,i}[3]$, $C_{j,i}[4]$, $C_{j,i}[5]$, $C_{j,i}[6]$, $C_{j,i}[7$
NO
5th byte of keystream $\oplus$ 0x01=0
YES
$C_{j,i}[1]$, $C_{j,i}[3]$, $C_{j,i}[5]$, $C_{j,i}[7]$
$C_{j,i}[0]$,, $C_{j,i}[2]$, $C_{j,i}[4]$, $C_{j,i}[6]$
Modified with 4 bytes of C.T4th, 8th, 12th,
16 bytes of Keystream
16 bytes of Plaintext
Combiner
16 bytes of Ciphertext

Fig. 4. Cipher feedback algorithm.

2.1.3. Combiner

The Combiner in the proposed stream cipher is implemented, using Rijndael round like function because of Rijndael varied properties. First, Rijndael round is applied as a nonlinear invertible combiner, since it provides efficient implementation on 8-bit processors, typical for current smart cards, and on 32-bit processors, typical for PCs. Second, it achieves high speed and compactness on a wide range of platforms. Third, the simplicity and resistance against known attacks are the other merits of Rijndael [10]. There are four transformations in this combiner, Add Roundkey stream, Byte Substitution, Shift Rows, and Dynamic folding. The new combiner shares more than one character in the encryption/decryption process. A graphical description of the combiner (round function) is illustrated in Fig. 5.

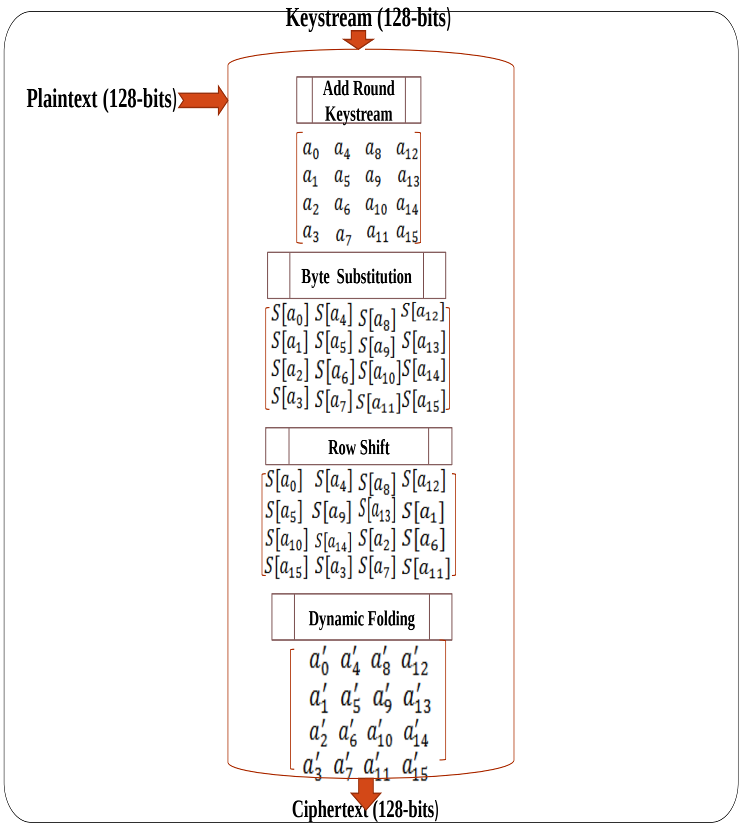

Fig. 5. The combiner of BloStream.

For the new combiner algorithm, the length of the cipher key is 128 bits. The input plaintext block is viewed as 4 × 4 matrixes of bytes called a state. Each round consists of four functions, which are listed below:

1) *Add Round Keystream Transformations*: this refers to the transformation in encryption and decryption in which a Round Key is added to the state, using an XOR operation. The length of a Round Key equals the size of the state (see Fig. 6).

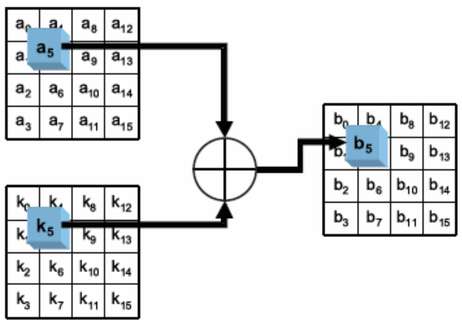

Fig. 6. Add round keystream transformation.

Byte Substitution Transformations: the signifies of the transformation in encryption that processes the state, employing a nonlinear byte substitution table (S-box) that operates on each of the state bytes independently (see Table 2).

Table 2. The Substitution Table-S-box [x, y] in Hexadecimal [11]

										Y							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
X	**7**	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	CA	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	08	D8
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	EA	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	48	BD	88	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

In decryption, the same transformations are implemented but with the use of the inverse substitution box. Fig. 7 illustration the byte substitution transformations.

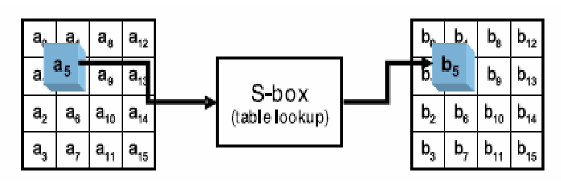

Fig. 7. Byte substitution transformation.

2) *Shift Rows Transformations*: this implies that the action of shifting rows is particularly simple, just performing left circular shifts of rows 1, 2 and 3, by amounts of 1, 2, and 3 bytes respectively. Row 0 is not changed. In the decryption process, the action of inverse shifting rows is particularly simple, just performing right circular of rows 1, 2, and 3, by amounts of 1, 2, and 3 bytes. Row 0 is not changed (see Figure 8).

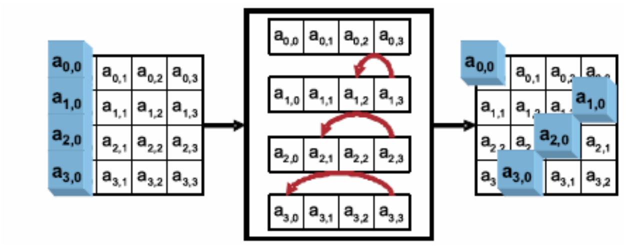

Fig. 8. Shift row transformation.

3) *Dynamic Folding Transformations*: this means that to perform the encryption and decryption process at the combiner, specific values are selected from the keystream generated by the PRNG. The direction for the new permutation and the position to start are extracted from those values to implement the dynamic folding as the following:

- The 1st byte of the keystream is AND'ed with 0 × 01, If the result = 0, the Horizontal Direction is right, Else, the Horizontal Direction is left
- The 2nd byte of the keystream is AND'ed with 0 × 01, If the result = 0, the Vertical Direction is up, Else, the Vertical Direction is down
- The 3rd byte of the keystream is AND'ed with 0 × 03, If the result = 0, the row is 0, if the result = 1, the row is 1, If the result = 2, the row is 2, If the result = 3, the row is 3
- The 4th byte of the keystream is AND'ed with 0x03, If the result = 0, the column is 0, If the result = 1, the column is 1, If the result = 2, the column is 2, If the result = 3, the column is 3.

Suppose the start point is (1, 2), after applied dynamic folding transformation, one of the four directions, illustrated in Fig. 9, is produced.

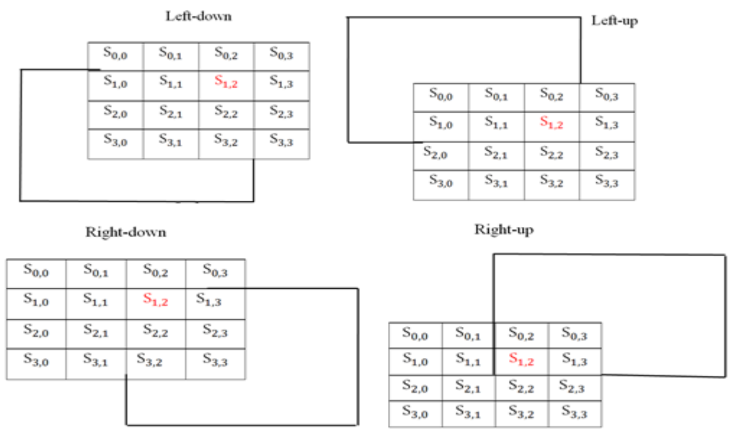
Left-down
Left-up
Right-down
Right-up
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$

Fig. 9. Dynamic folding four directions.

Considering the four stages of the proposed combiner, it can be argued that the new combiner has attained certain properties that make it more challenging against cryptanalysis. First, XOR is the initial operation; this ensures an absolute security to the plaintext. Moreover, the remaining transformations themselves will be a real complexity against the opponent. Second, the utilization of key dependent S-box in the combiner ensures large difficulties since S-box is unknown to some scope, it increases the number of probabilities faced by the attacker. Third, since XOR represents uniquely the first point of meeting between the plaintext with the confusion sequence, the adversary cannot analyze the other parts that transmit the effect of bits among each other. The selection of Rijndael round function seems a better choice for BloStream algorithm. Fourth, the probability that some items appear with the existence of dynamic folding in the proposed cipher combiner encompasses both rows and columns and involves mutating elements depending on a specific issue. Consequently, dynamic folding augments probabilities against the cryptanalysis. This means adding some key dependent operations to the main body of the combiner before outputting each block. There are 16 locations in the (4×4) state array to start the new permutation according to a specific direction. Instead of utilizing a fixed arrangement for the state, there are 64 (4 × 4) state arrays. Thus the opponent cannot specify the correct order easily and this will increase the difficulties of cryptanalysis.

2.2. Algorithm Inputs

There are four types of inputs in BloStream algorithm.

1) The key (K): it is the main key of the algorithm. This key is 16-bytes ($k_0, k_1, k_2, \dots, k_{15}$), which enters PRNG to produce the keystreams ($KS1, KS2, KS3, KS4$ ...). Each KS is 16 bytes

keystream$(ks_0, ks_1, \dots, ks_{15})$ in each iteration.

2) The plaintext (P1, P2, P3, P4 ...): it is the real message to be encrypted. Each P is 16- bytes $(p_0,\ p_1, \dots,\ p_{15})$ which enters the combiner.
3) Text1 (P0): it is used only once in the algorithm. It is 16-bytes text $(p_0,\ p_1, \dots,\ p_{15})$ utilized as a starting (virtual) plaintext which is input to the encryption/decryption function to produce the starting virtual cipher text C0, used in the feedback.
4) Text2 (C0): it is a virtual cipher text that is generated before actual encryption and decryption process, each C0 is 16 bytes $(c_0,\ c_1, \dots, c_{15})$ used in the feedback.

2.3. Algorithm Outputs

The cipher text (C1, C2, C3, C4 ...): it is the result of combining keystream with plaintext. Each C is 16-bytes $(c_0,\ c_1, \dots,\ c_{15})$ for each round.

2.4. Algorithm Stages

Fig. 10 and Fig. 11 illustrate the mechanism employed in the algorithm stages as described below:

2.4.1. Setup stage

This stage is performed by both the sender and receiver. The next state function defined in Rabbit algorithm is iterated four times. At the fourth iteration, 16-bytes (128-bit) are extracted to be assumed as virtual keystream KS0 which is input with the P0 to the combiner function to produce the cipher C0. Each KS0, P0, and C0 is known to both the encipherer and the decipherer. Then C0 is input to the PRNG to modify the $C_{j,i}$ counter variables of the internal states. At this point the modified key setup is completed. The proposed algorithm now is ready to get the actual keystream (KS1, KS2 ...) to be deployed in the encryption and decryption processes.

2.4.2. Encryption stage

Here, the first 16-bytes block of plaintext P1= $(p_0, p_1, \dots, p_{15})$ and 16-bytes keystream KS1= $(ks_0, ks_1, \dots, ks_{15})$are input to the combiner (forward round function) and the output is 16-bytes ciphertext C1= $(c_0, c_1, \dots, c_{15})$. But the encryption of the second 16-bytes block of plaintext P2 requires returning the previous block of cipher text C1 to the RNG. This pervious cipher text affects the next 16-bytes keystream KS2 generation. Then the second 16- bytes keystream KS2 and the second 16-bytes block of plaintext P2 enter the encryption function to produce the second 16-bytes block of cipher text C2. Also, the same mechanism is used to encrypt P3, P4... (see Fig. 10).

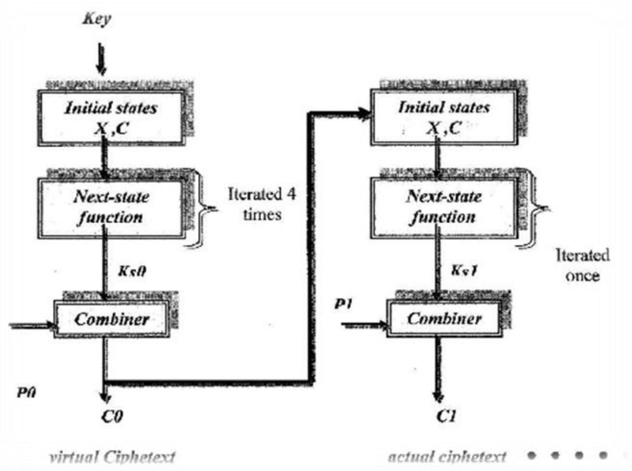

Fig. 10. Encryption stage.

2.4.3. Decryption stage

When the receiver obtains the cipher text C1, C0 is input to combiner (inverse function) with KSl to get P1 as the first plaintext block. To get P2, C1 is input to PRNG to get KS2 which is combined with C2 to produce P2. The same process is followed to decrypt C3, C4… (see Fig. 11).

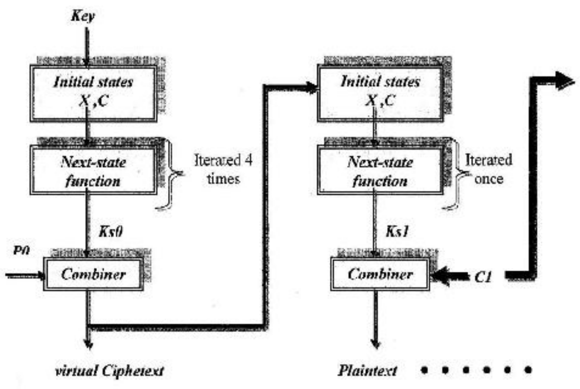

Fig. 11. Decryption stage.

3. Blostream vs. Others

In this section BloStream is compared with other similar stream ciphers such as Chameleon [12] and RC4 [13], especially with regard to performance and memory requirements. Chameleon is built from a new cryptographic primitive, called a Mutating S-box that shares some basic internal structures with RC4. In RC4 and Chameleon, there is an obvious symmetry between the state and the cycle that produces a clear biasing in the output, but the Initial Vector is employed to modify the state which is time consuming. In BloStream design, there is no symmetry between the state and the cycle. Two different algorithms are required (Rabbit PRNG and Rijndael-like round function) that avoid biases and make the cryptanalysis more complicated. Table 3 explains the speed measuring and memory requirement of the proposed cipher.

Table 3. BloStream Speed Measuring and the Required Memory

Cipher	Encryption Operation/byte	Decryption Operation/byte	Speed MB/S	Memory Requirements in Bytes
BloStream	4 operation	4 operation	66 MB/S	key setup: 68 Enc. / Dec. : 276

Table 4 summarizes the performance evaluation and memory requirements of RC4, Chameleon and BloStream in terms of key setup, encryption, decryption, and memory bytes.

Table 4. BloStream vs. RC4 and Chameleon

Cipher	Key Schedule without IV	Encryption 16-byte	Decryption 16-byte	Memory Requirements
RC4	3328 operation	768 operation	2560 operation	256 Bytes
Chameleon	3300 operation	80 operation	112 operation	513 Bytes
BloStream	1987 operation	64 operation	64 operation	344 Bytes

From the presented results, it is inferred that BloStream has better performance than RC4 and Chameleon stream ciphers. Moreover, the memory requirements are less in the proposed cipher.

4. Security Analysis with Possible Attacks

Several possible attacks are surveyed to examine the efficiency of Blostream.

1) *Brute-force attack*. With a key length of 128 bit, there are 2128 possible keys, which are approximately 3.4×10^{38} keys; moreover, Text 1 is changed with each communication. Thus, a brute-force attack appears impractical.
2) *Ciphertext only attack*. The plaintext data that is input to the combiner of BloStream algorithm is randomized, using the key stream sequence by XOR. Then this data is treated by many transformations of the round function that include SubBytes, Shift Rows, and dynamic folding. Then the letter frequency statistics will be concealed and the result is random-like output. Therefore, it is difficult to applied cipher text only attack.
3) *Known plaintext attack*. In the round function, each byte is XOR'ed with the key stream and then substituted, using keyed S-Box. The unknown values of the key stream and the shuffled S-Box make the attacker unable to determine the plaintext byte. Also, the dynamic folding, according to the key stream selected, changes dynamically in every iteration. This forms a more sophisticated state when deciphering the ciphertext byte.
4) *Differential attack*. Differential attack is invalid to apply to BloStream because the employed tables are 'keyed', i.e. initialized by a particular key. This implies that the attacker does not have a prior knowledge of a particular table arrangement.
5) *Distinguishing and Correlation attacks*. The nonlinear combiner of the proposed cipher has been defined to use an essential amount of inputs, e.g. virtual plaintext and the shuffled (or key dependant) S-box to perform strong transformations. In addition, the nonlinear equations of the PRNG make this kind of attack inapplicable.

5. Randomness Tests of the Keystream

Some statistical tests on Rabbit were performed; NIST Test Suite [14], the DIEHARD battery of tests [15] and the ENT test [16]. These tests were performed on the internal state as well as on the extracted output. Furthermore, the randomness property of the new PRNG is analyzed by performing the following statistical tests: Frequency test, Serial test, Poker test, Runs test, and Auto-correlation test [1]-[17]. Different key sizes were adopted in these tests. The keystream generated in the proposed PRNG successfully passed all five statistical tests for every run and no weaknesses were found in any case. Table 5 shows the results of applying the statistical equations.

Table 5. Statistical Test of PRNG of the Proposed Algorithm

Type of Tests	KEY1=256 bit	KEY2=384-bit	KEY3=512-bit	Pass Value
Frequency Test	0.5625	0.5104167	0.28125	must be≤ 3.841
Run Test	0.41151961	0.26455128	0.021514893	must be≤ 22.362
Poker Test	2.463904	4.283115	1.721505	must be≤ 14.067
Serial Test	0.67282104	0.66711426	1.9320679	must be≤ 5.991
Autocorrelation Test				
Shift 1	0.020383043	1.9317618E-4	0.42174298000	must be≤ 1.960
Shift 2	0.029912446	0.1895285500	0.24039537000	
Shift 3	0.346075680	0.1825514100	0.26257980000	
Shift 4	0.133855480	0.0850798800	1.12325324E-4	
Shift 5	0.158414360	0.3287709400	0.80725235000	
Shift 6	0.313954110	0.1980387100	1.32431070000	
Shift 7	0.005365565	0.0055902110	0.34034240000	
Shift 8	1.230112000	0.4823570300	0.55453010000	
Shift 9	0.078361400	0.1367103000	0.39982912000	
Shift 10	0.147380400	0.0019449207	0.15548588000	

6. Conclusion

The mechanism adopted in this research is a modem stream cipher, called Blostream, with a combining function which excels the weak classical concept of simple XOR combiner, transforming it into a stronger form which is suitable for computer cryptography. A complete description of the algorithm, an evaluation of its performance, security properties and implementation aspects were examined. Because of its simplicity in implementation and its small processer size, BloStream is flexible. Furthermore, the analysis demonstrated that the proposed cipher is fast and highly secure.

References

[1] Menezes, A. J., Oorschot, P., & Vanstone, S. (2006). *Hand book of Applied Cryptography.* Florida: CRC Press.

[2] Stamp, M. (2006). *Information Security: Principles & Practice*. New York: Wiley.

[3] Robshaw, M. J. B. (1995). Stream Ciphers, RSA Laboratories Technical Report. Retrieved from www.comms.scitech.susx.ac.uk/fft/crypto/stream_ciphers.pdf

[4] Ritter, T. (1996). The Penknife Cipher Design. Cryptogia. Retrieved September 10, 1996, from www.ciphersbyritter.com/PENDESN.HTM

[5] Ritter, T. (1997). Dynamic substitution in stream cipher cryptography: A reversible nonlinear combiner with internal state. Retrieved June 9, 1997, from www.ciphersbyitter.com/DYNSUB.HTM

[6] Ritter, T. (1990). Substitution cipher with pseudorandom shuffling: The dynamic substitution combiner. *Cryptologia*, *14(4)*, 289-303.

[7] Daemen, J. & Rijmen, V. (2002). *The Design of Rijndael: AES the Advanced Encryption Standard*. Berlin: Springer.

[8] Gurkaynak, F. K., Peter, L., Nico, B., Rene, B., Victoria, G., Marcel, M., Hubert, K., Norbert, F., & Wolfgang, F. (2006). *Hardware Evaluation of eSTREAM Candidates*. Zurich.

[9] Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J. & Zenner, E. (2003). *Rabbit Stream Cipher-Design & Analysis.* Denmark: Fast Software Encryption.

[10] Stallings, W. (2003). *Cryptography and Network Security: Principal and Practice*. New Jersey: Prentice-Hall.

[11] Glandman, B. (2003). *A Specification for Rijndael, the AES Algorithm*. Retrieved March 25, 2003, from http://ccc-classweb.ucsd. edu/ece111a

[12] Ross, A. & Charalampos, M. (1997). Chameleon: A new kind of stream cipher. *Fast Software Encryption, Lecture Notes in Computer Science*, *1267*, 107-113.

[13] McKague, M. E. (2005). *Design & Analysis of RC4-like Stream Ciphers*. PhD Thesis. Canada: University of Waterloo.

[14] NIST. (2001). A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. National Institute of Standards and Technology. Retrieved August 21, 2001, from http://csrc.nist.gov/rng

[15] Masaglia, G. (1996). A battery of tests for random number generators. Florida State University. Retrieved from http://stat.fsu.edu/ geo/diehard.html

[16] Walker, J. (1998). A pseudorandom number sequence test program. Retrieved May 21, 1998, from www.fourmilab.ch/random

[17] Beker, H., & Piper, F. (1982). *Cipher System: The Protection of Communication*. London: Northwood.

Ali H. Kashmar was born on January 3, 1969 in Iraq. He obtained his B.Sc. and M.Sc. degrees both in mathematics from Faculty of Sciences, University of Baghdad at Iraq in 1996 and 1999, respectively. He had been an active researcher and lecturer at many universities and institutions in Iraq. He is currently working as a lecturer in University of Baghdad, Iraq since 2006. He is also a member of International Association for Cryptologic Research (IACR, 2001-2002) He has been a full time Ph.D. student since 2012 in Universiti Kebangsaan Malaysia, Malaysia. His current research interests are mathematics, cryptography, statistics and information theory. He has a track record of fundamental research on these topics which is documented by numerous publications.

Eddie S. Ismail received B.Sc. & M.Sc degrees in mathematics from Universiti Kebangsaan Malaysia (UKM) in 1998 and 1999, and Ph. D. degree in cryptography from Universiti Sains, Malaysia (USM) in 2004. He is now an associate professor at UKM and his current research interests include digital signatures, cryptosystems, threshold systems, and stream cipher. He is also a member of International Association for Cryptologic Research (IACR, 20080348) and Malaysian Society for Cryptology Research (MSCR, 0062).

Firdaus M. Hamzah is a senior lecturer at the Unit of Fundamental Engineering Studies, Faculty of Engineering and Built Environment (FKAB), Universiti Kebangsaan Malaysia (UKM). His research interest includes hydrological, ecological, environmental and computational statistics, and engineering education. He is currently the head of Qualitative & Quantitative Reasoning Domain of Centre for Citra, UKM.

Haider F. Abdul Amir was born on June 4, 1969 in Iraq. He obtained his master and PhD degrees in engineering physics (with concentration on semiconductor devices, instrumentations and nuclear), from Faculty of Engineering, University of Gadjah Mada, Indonesia, Secondment of Osaka University Japan. He had his first degree in Faculty of Sciences, University of Baghdad at Iraq. He had been an active researcher and lecturer at many universities and institutions, in Iraq and Indonesia. He is currently working as an associate professor in Universiti Malaysia Sabah (UMS), Malaysia since 2006. His current research interests are nanotechnology in electronic, semiconductor materials, nuclear physics and green technology. He has a track record of fundamental research on these topics which is documented by numerous publications. He has been reviewer for numerous journals and indexed papers, such as Borneo Science, IEEE, Science Direct.