A SUGGESTED SUPER SALSA STREAM CIPHER

Mohammed Salih Mahdi[1]
[1]*BIT Dept., Business Information College,*
University of Information Technology and Communications,
Baghdad, Iraq
Mohammed.salih@uoitc.edu.iq

Nidaa Flaih Hassan[2]
[2]*Computer science Department*
University of Technology
Baghdad, Iraq
110020@uotechnology.edu.iq

***Abstract*: Salsa (20) cipher is speedier than AES cipher and its offered superior security. Salsa (8) and Salsa (12) are specified for apps wherever the grade of security is less necessary than speed. The concept of this research is to suggest super salsa keystream utilizing various volumes matrices size (array (4, 4), array (4, 8), array (4, 16)) are used to increase the complexity of key stream and make it more reluctant to linear and differential attacks. Furthermore, in each iteration, the diffusion of generated keystream will increase due the effect of changing the volume acting for one element of the array is not fixed. The generated keys of the suggested Super SALSA keystream are depicted as simple operations and a high hardiness randomly keystream by exceeding the five benchmark tests. Likewise, it's presenting a situation of equilibrium between complexity and speed for Salsa (8, 12 and 20).**

Keywords: Stream cipher, Keystream Generator Salsa (20), Super salsa.

I. INTRODUCTION

Ciphering procedures are mostly categorized into two forms: The 1st form is a block cipher indicates the procedure of the cipher by splitting each original data into sequential blocks and every block is encrypted by utilizing identical key [1], [2]. The 2nd form is a stream cipher indicates the procedure of the cipher by utilizing XOR function between the original data and key random series for getting the cipher data. The cipher and decipher procedures of stream cipher can be exhibited in the subsequent equations respectively [3].

$$C[s] = O[s] \oplus K[s] \qquad \text{(eq.1)}$$

$$O[s] = C[s] \oplus K[s] \qquad \text{(eq.2)}$$

Where $\oplus$ indicates mod by 2, C [s] is indicates the cipher data bits, K [s] is indicates the key random series bits, O [s] is indicates the original data bits and S is 1 bit at a same time. Focusing on eq. 1 and eq. 2, the cipher and decipher together, required to fit to use identical seed key to produce the identical keystream series K [s] as shown in figure (1) [3]. The alteration of keystream series will not let any assign a guide to the adversary to break the cipher data by providing a style of keystream would cannot be recurring [4], [5].

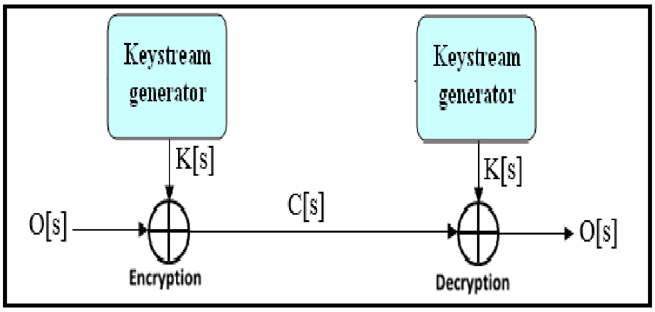

Fig1: Stream Cipher [3]

The stream ciphers are mostly categorized into two areas: software environment and hardware environment. It is an obvious, the robustness of the stream ciphers according to the keystream provider and can be evaluated in expressions of complexity, correlation and randomness [6]. A stream cipher is the most significant encryption procedures in pioneer scope: real time apps, military side, wireless sensor network, strategic regions, Bluetooth and mobile communications, etc. Because pioneer scopes have vast resource utilization and bounded cooperating with bandwidth [6]. In hardware, a stream cipher is mostly quicker than a block cipher, in addition, Stream cipher more suitable when memory is limited. Likewise, it has pros such as no error propagation and less complexity [7]. Moreover, Stream cipher is specified with expeditiously processing than block ciphers [8],[9].

1. Salsa (20)

Salsa (20) is stream cipher utilized counter mode for encryption procedure. The initial seed of Salsa (20) is an array (4, 4) with 512 bits as illustrated in figure (2).

constant1	key1	key2	key3
key4	constant2	nonce1	nonce2
counter1	counter2	constant3	key5
key6	key7	key8	constant4

=

v0	v1	v2	v3
v4	v5	v6	v7
v8	v9	v10	v11
v12	v13	v14	v15

Fig 2: Array of Salsa (20) distribution

The basic operations in salsa (20) are "edition, XOR and rotation" as shown in figure (3) which are applied on an Array of Salsa (20) for 10 rounds. Each round, the Array of Salsa (20) is changed a twice, so it's called Salsa (20). At the end of the salsa (20), addition operation is utilized between the final adjust of Array of Salsa (20) and the initial seed of Array of Salsa (20).

v[4] ⊕= (v[0] ⊞ v[12])<<<7	v[1] ⊕= (v[0] ⊞ v[3])<<<7
v[9] ⊕= (v[5] ⊞ v[1])<<<7	v[6] ⊕= (v[5] ⊞ v[4])<<<7
v[14] ⊕= (v[10] ⊞ v[6])<<<7	v[11] ⊕= (v[10] ⊞ v[9])<<<7
v[3] ⊕= (v[15] ⊞ v[11])<<<7	v[12] ⊕= (v[15] ⊞ v[14])<<<7
v[8] ⊕= (v[4] ⊞ v[0])<<<9	v[2] ⊕= (v[1] ⊞ v[0])<<<9
v[13] ⊕= (v[9] ⊞ v[5])<<<9	v[7] ⊕= (v[6] ⊞ v[5])<<<9
v[2] ⊕= (v[14] ⊞ v[10])<<<9	v[8] ⊕= (v[11] ⊞ v[10])<<<9
v[7] ⊕= (v[3] ⊞ v[15])<<<9	v[13] ⊕= (v[12] ⊞ v[15])<<<9
v[12] ⊕= (v[8] ⊞ v[4])<<<13	v[3] ⊕= (v[2] ⊞ v[1])<<<13
v[1] ⊕= (v[13] ⊞ v[9])<<<13	v[4] ⊕= (v[7] ⊞ v[6])<<<13
v[6] ⊕= (v[2] ⊞ v[14])<<<13	v[9] ⊕= (v[8] ⊞ v[11])<<<13
v[11] ⊕= (v[7] ⊞ v[3])<<<13	v[14] ⊕= (v[13] ⊞ v[12])<<<13
v[0] ⊕= (v[12] ⊞ v[8])<<<18	v[0] ⊕= (v[3] ⊞ v[2])<<<18
v[5] ⊕= (v[1] ⊞ v[13])<<<18	v[5] ⊕= (v[4] ⊞ v[7])<<<18
v[10] ⊕= (v[6] ⊞ v[2])<<<18	v[10] ⊕= (v[9] ⊞ v[8])<<<18
v[15] ⊕= (v[11] ⊞ v[7])<<<18	v[15] ⊕= (v[14] ⊞ v[3])<<<18
(a)	(b)

Fig3: Operations of Salsa (20): a. First Changing. b. Second Changing [10]

In each round of Salsa (20), ninety-six word operation has done, i.e. Forty-eight word operation for first changing followed by forty-eight word operation for second changing. Forty-eight word operation is calculated by multiplying sixteen word operation with three operations (addition, XOR and rotation). For 10 rounds, the total of operations is nine hundred and sixty word operations. At the end of the salsa (20), nine hundred and sixty word operations plus sixteen word operation conclude hundred and seventy-two word operations in total for one encryption [11]

II. Related Work

Many different papers are utilized to develop salsa (20), In [12], developed a modern procedure of salsa(20) by gaining swifter diffusion than the basic Salsa(20) according to chaos theory. Utmost of the experiences illustrated a modern procedure of two iterations is swifter than the basic four iterations, however it exhibits same diffusion grade. In [13], the array (4, 4) of Salsa (20) with 512 bits is altered to array (3, 3) of Salsa (20) with 576 bits, i.e. Each location in the array of Salsa (20) is utilized 64 bit words and in each iteration, changing their locations by applying nine operations, it led to more diffuse than the basic Salsa (20). Many different papers utilized to broken salsa (20), In [14] Crowley is reported attack on Salsa(20/5) by utilizing three iterations differential with alleged 2^{165} attempts and his award one thousand dollars . The adversary tries forwards on a short known original difference upon bias bit three iterations later, and tries two iterations backwards from a result in accordance with estimating one hundred and sixty pertinent key bit. In [15], reported attack on Salsa(20/6) and swifter attack on Salsa(20/5) by utilizing four iteration differential with alleged 2^{177} attempts. The adversary tries forwards on a short known original difference upon bias bit four iterations later, and tries two iterations backwards from a result in accordance with estimating one hundred and sixty pertinent key bit .In [16], reported attack on Salsa(20/7) and swifter attack on Salsa(20/6) by utilize four iteration differential with alleged 2^{190} attempts. The adversary tries forwards on a short known original difference upon bias bit four iterations later, and tries three iterations backwards from a result in accordance with estimating one hundred and seventy-one pertinent key bit. In[17], reported attack on Salsa(20/8) with alleged 2^{249} attempts and swifter attack on Salsa(20/7) with alleged 2^{153} attempts. The adversary tries forwards on a short known original difference upon bias bit four iterations later, and tries four iterations backwards from a result in accordance with estimating two hundred and twenty-eight pertinent key bit.

Concerning , salsa (20) that is debated at the top, the contribution of this work is manifesting super salsa keystream generator , the array (4, 4) of Salsa (20) with 512 bits is altered to array (4, b) of three versions of Salsa (8/12/20) with 512 bits for presenting a situation of equilibrium between complexity and speed. Furthermore, in each iteration, the diffusion of generating the keystream will increase due the effect of changing the volume acting for one element of the array (4, b), i.e. The represented volume of each element in the array (4, b) is not fixed.

III. A Suggested Super SALSA Stream Cipher

For encrypted data, the essential basics of the salsa (20) have been analyzed to establish the suggested Super SALSA keystream generator. Three versions of the Super SALSA keystream generator are suggested, these versions (8,12,20) are utilized array with (4, b) size instead of utilizing an array with (4, 4) size as interpreted in algorithm (1), algorithm (2) and algorithm (3). The (4, b) size of array is ultimately summarized into:

- b= 4 → the size of the array is (4 by 4), each item in this array is acted by 32 bits, the array consists of key (k1 to k 8), nonce (n 1, n 2) and constant (ct1 to ct4) and counter (cr1, cr2) as illustrated in figure (4.a).
- b = 8 → the size of the array is (4 by 8), each item in this array is acted by 16 bits, the array is consist of key (k 1 to k 16), nonce (n1 to n4) and constant (ct1 to ct8) and counter (cr1 to cr4) as illustrated in figure (4.b).
- b= 16→ the size of the array is (4 by 16), each item in this array is acted by 8 bits and the array is consist of key (k1 to k32), nonce (n1 to n8) and constant (ct1 to ct16) and counter (cr1 to cr8) as illustrated in figure (4.c).

In each iteration, the parameter (b) is selected according to the super key. The super key contains set of b parameters with versions (8,12,20) and it's generated randomly, securely among session members.

Algorithm (1): Suggested Super SALSA keystream Generator

Input: version-salsa =8,12;20, superkey=1,2;3 with length of version-salsa, 512 bit (seed key, nonce, counter and constants) stored in a matrix v

Output: 512 bit (keystream)

Begin
Step1: iteration=1
Step2: s=superkey (iteration)
    if s ==1 then
        Goto Salsa-function with matrix v(4,4)
    else if s ==2 then
        Goto Salsa-function with matrix v(4,8)
    else
        Goto Salsa-function with matrix v(4,16)
Step3: iteration = iteration +1
Step4: if (iteration less than or equal to version-salsa) Goto step2
Step5: store the update matrix v in keystream matrix as key
End

ct1	k1	k2	k3
k4	ct2	n1	n2
cr1	cr2	ct3	k5
k6	k7	k8	ct4

(a)

ct1	k1	k2	k3	ct5	k9	k10	k11
k4	ct2	n1	n2	k12	ct6	n3	n4
cr1	cr2	ct3	k5	cr3	cr4	ct7	k13
k6	k7	k8	ct4	k14	k15	k16	ct8

(b)

ct1	k1	k2	k3	ct5	k9	k10	k11	ct9	k17	k18	k19	ct13	k25	k26	k27
k4	ct2	n1	n2	k12	ct6	n3	n4	k20	ct10	n5	n6	k28	ct14	n7	n8
cr1	cr2	ct3	k5	cr3	cr4	ct7	k13	cr5	cr6	ct11	k21	cr7	cr8	ct15	k29
k6	k7	k8	ct4	k14	k15	k16	ct8	k22	k23	k24	ct12	k30	k31	k32	ct16

(c)

Fig (4): Arrays of Super Salsa Distribution, a: Super Salsa Array (4, 4), b: Super Salsa Array (4, 8) , c: Super Salsa Array (4, 16).

Algorithm (2): Salsa-function

Input: b= 4 ,8;16 , matrix v(4,b)

Output: Goto Salsa-quarter function with right positions

```
Begin
Step1:
    if b==4 then
        Salsa-quarter function (v 0, v 4, v 8, v 12)
        Salsa-quarter function (v 5, v 9, v 13, v 1)
        Salsa-quarter function (v 10, v 14,v2, v 6)
        Salsa-quarter function (v 15, v 3, v 7, v 11)
        Salsa-quarter function (v 0, v 1, v 2, v 3)
        Salsa-quarter function (v 5, v 6, v 7,v 4)
        Salsa-quarter function (v 10, v 11, v 8, v 9)
        Salsa-quarter function (v 15, v 12, v 13, v 14)
    else if b ==8 then
        Salsa-quarter function (v 0, v 4, v 8, v 12)
        Salsa-quarter function (v 5, v 9, v 13, v 1)
        Salsa-quarter function (v 10, v 14,v2, v 6)
        Salsa-quarter function (v 15, v 3, v 7, v 11)
        Salsa-quarter function (v 0, v 1, v 2, v 3)
        Salsa-quarter function (v 5, v 6, v 7,v 4)
        Salsa-quarter function (v 10, v 11, v 8, v 9)
        Salsa-quarter function (v 15, v 12, v 13, v 14)
        Salsa-quarter function (v 16, v 20, v 24, v 28)
        Salsa-quarter function (v 21, v 25, v 29, v 17)
        Salsa-quarter function (v 26, v 30,vl 8, v 22)
        Salsa-quarter function (v 31, v 19, v 23, v 27)
        Salsa-quarter function (v 16, v 17, v 18, v 19)
        Salsa-quarter function (v 21, v 22, v 23,v 20)
        Salsa-quarter function (v 26, v 27, v 24, v 25)
        Salsa-quarter function (v 31, v 28, v 29, v 30)
    else
        Salsa-quarter function (v 0, v 4, v 8, v 12)
        Salsa-quarter function (v 5, v 9, v 13, v 1)
        Salsa-quarter function (v 10, v 14,v2, v 6)
        Salsa-quarter function (v 15, v 3, v 7, v 11)
        Salsa-quarter function (v 0, v 1, v 2, v 3)
        Salsa-quarter function (v 5, v 6, v 7,v 4)
        Salsa-quarter function (v 10, v 11, v 8, v 9)
        Salsa-quarter function (v 15, v 12, v 13, v 14)
        Salsa-quarter function (v 16, v 20, v 24, v 28)
        Salsa-quarter function (v 21, v 25, v 29, v 17)
        Salsa-quarter function (v 26, v 30,vl8 , v 22)
        Salsa-quarter function (v 31, v 19, v 23, v 27)
        Salsa-quarter function (v 16, v 17, v 18, v 19)
        Salsa-quarter function (v 21, v 22, v 23,v 20)
        Salsa-quarter function (v 26, v 27, v 24, v 25)
        Salsa-quarter function (v 31, v 28, v 29, v 30)
        Salsa-quarter function (v 32, v 36, v 40, v 44)
        Salsa-quarter function (v 37, v 41, v 45, v 33)
        Salsa-quarter function (v 42, v 46,v34 , v 38)
        Salsa-quarter function (v 47, v 35, v 39, v 43)
        Salsa-quarter function (v 32, v 33, v 34, v 35)
        Salsa-quarter function (v 37, v 38, v39,v 36)
        Salsa-quarter function (v 42, v 43, v40, v 41)
        Salsa-quarter function (v 47, v 44, v 45, v 46)
        Salsa-quarter function (v 48, v 52, v 56, v 60)
        Salsa-quarter function (v 53, v 57, v 61, v 49)
        Salsa-quarter function (v 58, v 62,v50 , v 54)
        Salsa-quarter function (v 63, v 51, v 55, v 59)
        Salsa-quarter function (v 48, v 49, v 50, v 51)
        Salsa-quarter function (v 53, v 54, v 55,v 52)
        Salsa-quarter function (v 58, v 59, v 56, v 57)
        Salsa-quarter function (v 63, v 60, v 61, v 62)
End
```

Algorithm (3): Salsa-quarter function

Input: a1, b1, c1, d1

Output: Update a1, b1, c1, d1

Begin

Step1:

$z1 = b1 \oplus ((a1 + d1) \lll 7)$

$z2 = c1 \oplus ((x1 + z1) \lll 9)$

$z3 = d1 \oplus ((z1 + z2) \lll 13)$

$z0 = a1 \oplus ((z2 + z3) \lll 18)$

Step2:

a1=z0
b1=z1
c1=z2
d1=z3

End

IV. Results of Suggested Keystream Generator

The suggested Super SALSA keystream generator has been analyzed and implemented by utilizing C++. The level of randomness of the Super SALSA keystream generated has been estimated by checking the five benchmark tests as interpreted in Table (1).

Table (1): Benchmarked 5-Tests Equations and Information [16]

5 - benchmark tests Equations	Information on 5 - benchmark tests
$T1 = \frac{(M0 - M1)^2}{M}$	M0: number of 0's in keystream. M1: number of 1's in keystream. M: total size of keystream.
$T2 = \frac{4}{M-1}((M11)^2 + (M00)^2 + (M01)^2 + (M10)^2) - \frac{2}{M}(M1^2 + M0^2) + 1$	M11: number of 11's in keystream. M00: number of 00's in keystream. M01: number of 01's in keystream. M10: number of 10's in keystream
$P = \frac{M}{N}$ $\quad \frac{M}{N} \geq (5 * 2^N)$ $T3 = \frac{2^N}{P}\left(\sum_{j=1}^{2^N} M_j^2\right) - P$	M_j: number of appearance of the j^{th} of length N.
$P_j = \frac{M - j + 3}{2^{(j+2)}}$ $T4 = \left(\sum_{j=1}^{N} \frac{(B_j - P_j)^2}{P_j}\right) + \left(\sum_{j=1}^{N} \frac{(G_j - P_j)^2}{P_j}\right)$	N :maximum j for which $P_j \geq 5$. B_j: Amount of blocks (subsequences runs of 1's) of length j in M. G_j: amount of gabs(subsequences runs of 0's) of length j in M.
$A(k) = \sum_{j=0}^{M-k-1} (S_j + S_{j+k}) \bmod 2$ $T5 = \frac{2\left(A(k) - \frac{(M-k)}{2}\right)}{\sqrt{(M-k)}}$	$k : 1 \leq k \leq [m/2]$

The following steps illustrated the generated keystream of the suggested Super SALSA keystream according to Algorithm (1), Algorithm (2) and Algorithm (3).

- S_1: Suppose the version-salsa is 8, the array of superkey is [3,1,1,3,2,2,1,3] and the seed of keystream is "43cb80a0","530fa20d","b7b05c9f","78bf2735","4d063cf3","d2151f5a","544f1190","183be031","75480d5c","6c51a262","a47b7d1a","dc0fc344","bf615c38","63ab9b04","2ab5ba53","b8532ed1" storing in matrix v which is represented by 128 hexa-number that is equal to 512 bits.
- S_2: According to superkey, at index 0 of superkey array is 3 in iteration 1. So, matrix v (4,4) will be converted to v v(4,16) each item in this array is acted by 2 hexa number (8 bits) ,the update matrix v "43","cb","80","a0","53","0f" ,"a2","0d",………. "b8","53", "2e","d1" according to Algorithm (2)
- S_3:Going to salsa-quarter function with b=16 ,so v(4,16) converts to v(64) .At first update v4,v8,v12 and v0 by implementing the three procedure mod $2^{8,16;32}$ of suggested Super SALSA keystream e.g. v4="53", v8="b7", v12="78" and v0="43". According to Algorithm (3) , So, getting the update v4 by firstly, v[0] + v[12]→ "43" + "78"="bb". Secondly, rotate by 7 → "bb">>>7="dd" then finally v4="53"

⊕ "dd" mod 28→ update v4=" 8e" and so on form each item in matrix v.

- S_4: Next to iteration 2, According to superkey, at index 1of superkey array is 1. So, will be converted to v(4,4) each item in this array is acted by 8 hexa number (32 bits) and so on.
- S_5: check fitness function when finishing all iteration, already, the generated keystream is passing the five standard criteria. So the final result in binary form:

"10001010100111110000100000101011010100110000110
1
0011010001111111100110100001000001100000000000010
1011001010000101110011101110000101011010010010100
0110011011000000001110111010110001001100001011000
0
0111010000011010000100100001101111000111011010101
1000111111000001001010001001101100100110110000101
0010101011110111111001111110111110001101110001111
0010001111100001000100101010010001110010110000011
0000000100011111001111010011100101010100010101011
1000011110110001101101101011110101011111001001001
1000110000000000000001100"

The above generated keys sample of the suggested Super SALSA keystream are depicted as simple operations and a high hardiness randomly keystream by exceeding the five benchmark tests as interpreted in Table (2).

Table (2): Five Benchmark Tests Performance

5 - benchmark Tests	Test Value	Threshold	Test value< Threshold
Frequency T1	2.82	3.841	pass
Serial T2	3.698	5.991	pass
Poker T3	8	24.995	pass
Runs T4	4.61	12.591	pass
Autocorrelation T5	0.336	1.96	pass

Many of superkey of Super SALSA keystream generator has been taken in is compared with standard salsa version (8),(12)and (20) according to time consuming and complexity applies on 512 bits in milliseconds as showing in Table (3) and Table (4). So, as output, the Suggested super Salsa version of 8, 12 and 20 is greater than Standard Salsa particularly in complexity., however super Salsa need more time don't exceed 60 milliseconds that indicated its good indicator when applies on real time apps or any apps.

Table (3): time consuming and complexity of Standard Salsa

Criteria	Standard Salsa		
versions	8	12	20
time	20	25	29
complexity	Low	medium	High

Table (4): time consuming and complexity of Suggested super Salsa

Criteria	Standard Salsa		
versions	8	12	20
time	30	42	55
complexity	medium	high	High

V. Conclusion

This paper suggests Super Salsa Keystream Generator with robust construction, according to utilizing an array with (4, b) size instead of utilizing an array with (4, 4) volume of salsa (8, 12, 20). Presenting a situation of equilibrium between complexity and speed. Also, the diffusion of generating the keystream will increase due the effect of changing the volume acting for one element of the array (4, b) in each iteration. When b is equal to 4 that means the size of each element is equal to 32 bits, when b is equal to 8 that means the size of each element is equal to 16 bits, while, b is equal to 16 that means the size of each element is equal to 8 bits. Furthermore, utilizing various volumes of size of matrices of super salsa is guide to increase complexity of key stream and make it more reluctant to linear and differential attacks. Its need for 2^{512} Probable keys to break super salsa, which is guided to not utilize brute-force attacking due to its unwieldy procedure in this situation. In addition, the randomness of the Super SALSA keystream has successfully exceeded the five benchmark tests.

References

[1] T. W. Cusick, C. Ding, and A. R. Renvall, Stream ciphers and number theory, vol. 66. Elsevier, 2004.

[2] E. Bach and J. O. Shallit, Algorithmic Number Theory: Efficient Algorithms, vol. 1. MIT press, 1996.

[3] Y. Minglin and M. Junshuang, "Stream ciphers on wireless sensor networks," in Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on, 2011, vol. 3, pp. 358–361.

[4] B. Schneier, Applied cryptography: protocols, algorithms, and source code in C. john wiley & sons, 2007.

[5] M. Hell, T. Johansson, A. Maximov, and W. Meier, "A stream cipher proposal: Grain-128," in Information Theory, 2006 IEEE International Symposium on, 2006, pp. 1614–1618.

[6] M. Shin, J. Ma, A. Mishra, and W. A. Arbaugh, "Wireless network security and interworking," Proc. IEEE, vol. 94, no. 2, pp. 455–466, 2006.

[7] A. Jolfaei and A. Mirghadri, "Survey: image encryption using Salsa20," Int. J. Comput. Sci. Issues, vol. 7, no. 5, pp. 213–220, 2010.

[8] H. Lee and K. Chen, "Pingpong-128, a new stream cipher for ubiquitous application," in Convergence Information Technology, 2007. International Conference on, 2007, pp. 1893–1899.

[9] T. Good and M. Benaissa, "Hardware results for selected stream cipher candidates," State Art Stream Ciphers, vol. 7, pp. 191–204, 2007.

[10] D. J. Bernstein, "The Salsa20 family of stream ciphers," in New stream cipher designs, Springer, 2008, pp. 84–97.

[11] D. Priemuth-Schmid and A. Biryukov, "Slid pairs in Salsa20 and Trivium," in International Conference on Cryptology in India, 2008, pp. 1–14.

[12] M. Almazrooie, A. Samsudin, and M. M. Singh, "Improving the diffusion of the stream cipher salsa20 by employing a chaotic logistic map," J. Inf. Process. Syst., vol. 11, no. 2, pp. 310–324, 2015.

[13] A. Al-Saleh, M. Al-Ahmmad, A. Issa, and A. Al-Foudery, "Double-A -- A Salsa20 Like: The Design," 2015 4th Int. Conf. Adv. Comput. Sci. Appl. Technol., pp. 24–29, 2015.

[14] C. Paul, "Truncated differential cryptanalysis of five rounds of Salsa20," Stream Ciphers Revisit., 2006.

[15] S. Fischer, W. Meier, C. Berbain, Jean-Fran, C. Biasse, and M. J. B. Robshaw, "Non-randomness in eSTREAM candidates Salsa20 and TSC-4," In INDOCRYPT, pp. 2–16, 2006.

[16] Y. Tsunoo, T. Saito, H. Kubo, T. Suzaki, and H. Nakashima, "Differential cryptanalysis of Salsa20/8," SASC, State Art Stream Ciphers, 2007.

[17] J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger, "New features of Latin dances: analysis of Salsa, ChaCha, and Rumba," in International Workshop on Fast Software Encryption, 2008, pp. 470–488.

AUTHOR PROFILE

MOHAMMED SALIH MAHDI IS CURRENTLY A PHD CANDIDATE. HIS BSC DEGREE IN HIDING DATA IN 2010 AND HIS MSC DEGREE IN A SECURITY OF CLOUD COMPUTING IN 2012 FROM COMPUTER SCIENCE DEPT., AT UNIVERSITY OF TECHNOLOGY, IRAQ, BAGHDAD. CURRENTLY. LECTURER IN BUSINESS INFORMATION COLLEGE, UNIVERSITY OF INFORMATION TECHNOLOGY AND COMMUNICATIONS. HIS RESEARCH INTERESTS INCLUDE DATA MINING, ARTIFICIAL INTELLIGENT, COMPUTER SECURITY, IMAGE PROCESSING, DATA COMPRESSION, HEALTHCARE, MOBILE APPLICATION, CLOUD COMPUTING, INTERNET OF THINGS, INTERNET OF EVERYTHING.

ASSIST. PROF .DR. NIDAA F. HASSAN RECEIVED THEMSC. AND PHD. IN COMPUTER SCIENCE FROM UNIVERSITY OF TECHNOLOGY, IRAQ, 1996 AND 2005 RESPECTIVELY. SHE HAS AROUND 21 YEARS OF TEACHING EXPERIENCE. HER AREAS OF INTEREST'S COMPUTER SECURITY AND IMAGE PROCESSING