Using Modified Genetic Algorithm to Replace AES Key Expansion Algorithms

Abdullah Abdali Rashed
Saba university Research, Studies and Training Center
Sana, Yemen.
abdullahRashed@yahoo.com

Abstract
The main criterion of the ciphering key is that it will be expanded to schedule key in ciphering phase. In addition, the schedule key is used to encrypt the plain text block. However it is preferred to have an inverse that will be used to decrypt the ciphered data block. This paper presents a new simple approach to carry out the key expansion process; it first takes the cipher key and expands it to construct the schedule key. The proposed approach is simple and fasts as it is based on Modified Genetic Algorithm (MGA).

Key Words: AES Algorithm, Genetic Algorithms, Ciphering, Key Expansion.

1. Introduction

Key schedule algorithm process is expanding short cipher key (128 bits) into large set of keys (1408 bits), called round keys (10). Schedule key is very important phase in ciphering algorithms, as a strong schedule key means a strong cipher that would be more resistant to various forms of attacks, such as differential and linear cryptanalysis (5).
Carter, Dawsony and Nielseny (5) classified AES candidates according to key schedules as the authors thought that key expansion is very important as strong schedule key means stronger algorithm against both linear and differential cryptanalysis. They recommended that the schedule key of AES candidates should be upgraded.

Genetic Algorithms (Gas) have played a strong role in data security systems, Yaseen et.al. (15) used genetic algorithm for the cryptanalysis, Spillman and et al. used GA to cryptanalysis a simple substitution (14), Mathews used GA in transposition ciphers (9) and Spillman used GA in knapsack based systems (13), Bagnall used GA to crack difficult systems such as block cipher (Data Encryption Standard DES) (3), Grundlingh et. al. used GA to attack mono-alphabetic substitution but their approach not seemed effective against transposition (8). Bagnall et al. used Genetic Algorithm as cryptanalysis of a three rotor machine using genetic algorithm and their results showed that an unknown three rotor machine can be cryptanalysed with about 4000 letters of ciphertext (4),

Dimovski et al. (7) presented an automated attack on the polyalphabetic substitution cipher whereas Rashed (12) used Limited Genetic Algorithm (LGA) to generate a pool of cipher keys and schedule keys that will be used in ciphering and deciphering AES processes. However it was suggested having a pool of AES keys and a schedule key would be taken from this pool then the ciphered block and index of the start location (1).

It is useful to avoid the normal key scheduling process, and specify the cipher keys (which should be random and independent) directly (2). In this research the genetic algorithm process will be modified to assist in the process of generating ciphering and scheduling keys.

2. Basic Idea of GAs (12)

Genetic algorithms consist of three phases as following:

(I) *Reproduction Operation:* The old string is carried through into a new population depending on the performance index values. The fitness values are calculated for each candidate string using a fitness function, which depends on a goal for optimization problems. According to the fitness values, string with larger fitness values give rise to a larger number of copies in the next generation.

(II) *Crossover operation:* The strings are randomly mated using the crossover operation. Each pair of candidate strings will undergo crossover with the probability cross. This operation provides randomized information exchange among the strings.

(III) *Mutation operation:* Mutation is simply an occasional random alteration of the value of a string position. In a binary code, this involves changing a 1 to 0 and vice versa. The sequence of successive stages of genetic algorithms is shown in figure (1).

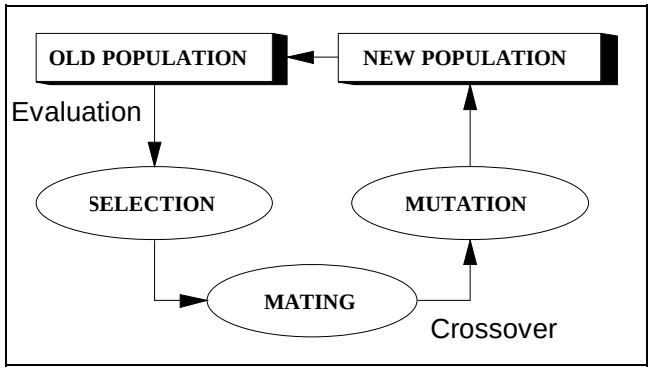


Figure (1) sequence of genetic algorithm

3. Key Expansion in AES (6 and 11)

The Expanded Key is a linear array of cipher key (4-byte words) and is denoted by W[Nb*(Nr+1)]. The first **Nk** words contain the Cipher Key. The key expansion function depends on the value of **Nk**: there are two versions of Nk as Nk may be equal to or below 6 (Nk=4 or 6), or Nk above 6 (Nk=8) for **Nk** $\leq$ 6, we have the following algorithm:

```
Function keyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
Begin
  For i = 0 to Nk step by 1
     W[i] = (Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3])
   End for
```

```
 For i=Nk to Nb (Nr + 1) step by 1
   Begin
     temp = W[i - 1];
     if   i   mod  Nk = 0
        temp = subBytes(rotBytes(temp)) ^ rcon[i / Nk]
    W[i] = W[i - Nk] ^ temp
   End if
 End for
End  keyExpansion
```

subBytes(state) is a method that substitutes each byte of applying the AES (Rijndael) S-box to the byte at the corresponding position in the input word returns a 4-byte word. The function rotBytes(W) returns a word in which the bytes are a cyclic permutation of those in its input such that the input word (a,b,c,d) produces the output word (b,c,d,a).

The first **Nk** words are filled with the cipher key. Every following word W[i] is equal to the EXOR of the previous word W[i-1] and the word **Nk** positions earlier W[i-Nk].

For words in positions that are a multiple of **Nk**, a transformation is applied to W[i-1] prior to the EXOR and a round constant is EXORed. This transformation consists of a cyclic shift of the bytes in a word (rotByte), followed by the application of a table lookup to all four bytes of the word (subBytes).

For **Nk** > 6, we have:

```
Function keyExpansion(byte Key[4*Nk] word W[Nb*(Nr+1)])
Begin
 For i = 0 to Nk step by 1
    W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3])
   End for
 For i = Nk  to Nb  (Nr + 1) step by 1
  Begin
   temp = W[i - 1]
   if  i mod Nk = 0
         temp = subBytes(rotByte(temp)) ^ rCon[i / Nk]
   end if
     else if (i mod  Nk = 4)
        temp = SubBytes(temp)
      end else if
    W[i] = W[i - Nk] ^ temp
   End for
End  keyExpansion
```

The round constants are independent of **Nk** and defined by:

rCon[i] = (RC[i],‘00’,‘00’,‘00’) with RC[i] representing an element in GF(2^8) with a value of x(i - 1) so that:
RC[1] = **1** (i.e. ‘01’)
RC[i] = **x** (i.e. ‘02’) ·(RC[i-1]) = x**(i-1)**

Round Key Selection
Round key (i) is given by the round key buffer words W[Nb*i] to W[Nb*(i+1)]

The cipher
The cipher AES consists of

- An initial Round Key addition;
- **Nr**-1 Rounds;
- A final round.

The algorithm is:

```
Input:  State, CipherKey
Output: cipheredBlock
Function AES
Begin
keyExpansion(CipherKey,ExpandedKey) ;
addRoundKey(State,ExpandedKey);
 for  i=1 to Nr-1  step by 1
     Round(State,ExpandedKey + Nb*i) ;
    cipheredBlock= finalRound(State,ExpandedKey + Nb*Nr);
  end for
end AES
```

The key expansion can be done on in advance and AES can be specified in terms of the Expanded Key.

```
Input: State, ExpandedKey
Output: cipheredBlock
Function AES
Begin
addRoundKey(State,ExpandedKey)
   for  i=1 to Nr-1  step by 1
    round(State,ExpandedKey + Nb*i)
    cipheredBlock= finalRound(State,ExpandedKey + Nb*Nr)
end for
end AES
```

Note: There are no restrictions on the selection of the Cipher Key. Whereas expanded key must be expanded from cipher key.

4. Proposed Algorithm

To date there has been no reported research using GAs in any form or shape within a key expansion. The work introduced in this paper will show the use of the first modified GAs in key expansion algorithm. The modified GAs will only use some of the conventional GAs process; this will include random initialization of the first population of cipher keys, and then apply the process of random cross over and mutation to produce further sets of cipher keys. In this research all cipher keys will be considered acceptable and hence, there will be no need to search for a best cipher key, since all keys will be used to cipher data in this system. The proposed algorithm in this case will only be used to generate cipher keys and it will not involve any search or optimization techniques.

4.1 Algorithm MGA Key Expansion

Input: Nk
Output: schedule key with length (Nr+1)Nb
Begin
Initialization: generate a cipher key (Nk words) in hexadecimal format.
Begin
Generate Child 1 and Child 2 by Calling MGA algorithm (cipher key)
Generate Child 3 and Child 4 by Calling MGA algorithm (Child 1)
Generate Child 5 and Child 6 by Calling MGA algorithm (Child 2)
Generate Child 7 and Child 8 by Calling MGA algorithm (Child 3)
Generate Child 9 and Child 10 by Calling MGA algorithm (Child 4)
If Nk= 6 then
Generate Child 11 and Child 12 by Calling MGA algorithm (Child 5)
End if
If Nk= 8 then
Generate Child 13 and Child 14 by Calling MGA algorithm (Child6)
End if

The crossover point would be in the range [2...6] and should not be duplicated as the crossover point of the child should be different to the parents.

End

4.2 Function MGA

Input: cipher key with 16 bytes

Output: two children considered as round keys

Begin

Convert all elements of the cipher key into binary String

Generate random crossover point using equation 1

Recombine individuals

Mate individuals as follows:

Generate two random points, r1 and r2

Key1 = $leftkey_{r2}$ with $rightKey_{r1}$

Key2 = $rightKey_{r1}$ with $leftkey_{r2}$

Add the new population to the old ones

End GACipherkey

For the second round: The crossover point should not be the same as the previous round as it might produce the parent element again. To prevent repeating the parent value (at grand child level), the algorithm have to follow up the following equation (1).

If old Crossover point= new Crossover point then

New crossover point = i+i mod 5 equation (1)

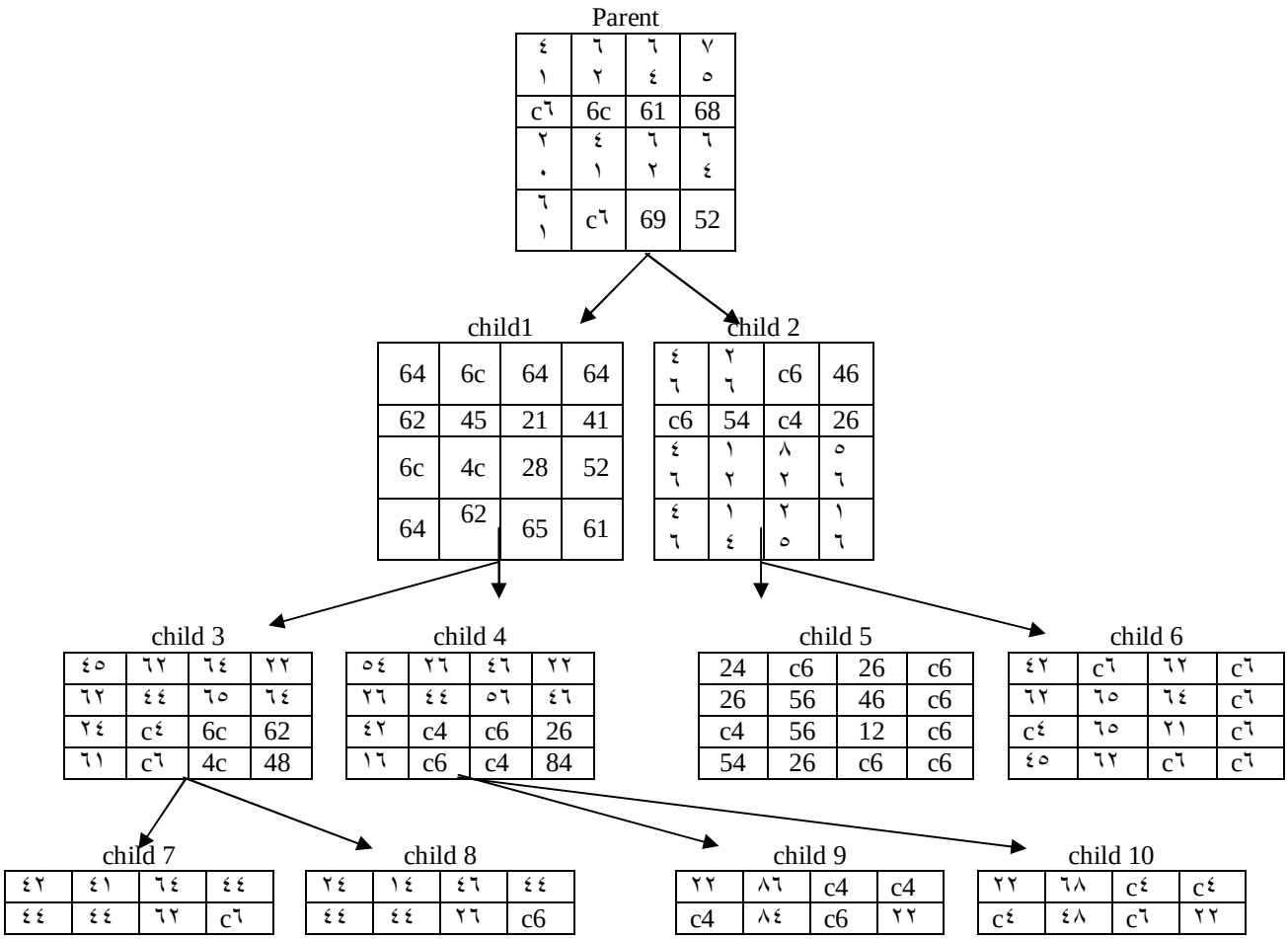
5. Advantages of the Proposed Scheme

The use of modified Genetic algorithm can introduce number of advantages to the whole process of crypto industry. These advantages will include the following:

- This approach is very fast with comparison to all other conventional methods used to date.
- In this approach only the cipher key and the crossover point for each round will be sent to the receiver.

6. Illustrative Example Key Expansion

Assuming that we have the cipher key =41 c6 20 61 62 6c 41 c6 64 61 62 69 75 68 64 52 the schedule key should be as shown in figure 2 and figure 3.



٤٥	٦٤	٦٢	٤١
٦٢	٦٢	٦٥	٦١

٥٤	٤٦	٢٦	١٤
٢٦	٢٦	٥٦	١٦

٢٦	٤٢	٢٦	٢٦
٤٢	c6	٥٦	٤٦

٦٢	٢٤	٦٢	٦٢
٢٤	c٦	٦٥	٦٤

Figure 2: MGA for Nk=4

7. Conclusion

This paper showed how modified genetic algorithms can be used to produce a ciphering or schedule key. However the Modified Genetic algorithm can be used to expand the cipher key to schedule key in any ciphering algorithm. The results obtained using this method has showed a highly secured and efficient algorithm and it decreased the complexity of the original AES algorithm by more than 50%. As future work the algorithm can be enhanced to have an inverse algorithm.

References

1. Ajlouni N. A. El-Sheikh and A.Abdali Rashed, **New Approach in Key Generation and Expansion in Rijndael Algorithm**, International Arab Journal of Information Technology, vol. 3, no. 1, January 2006, www.IAJIT.org.
2. Symmetric Ciphers, http://www.amasci.com/~weidai/scan-mirror/cs.html
3. Bagnall A., **The Application of Genetic Algorithm Cryptanalysis**, Mater Degree Thesis, 1996, School of Information Systems, University of East Anglia,
4. Bagnall A., McKeon G. and Rayward-Smith V., **The Cryptanalysis of a Three Rotor Machine Using Genetic Algorithm,** available at http://citeseer.nj.nec.com/162166.html

5. Carter G., Dawsony E. and Nielseny L., **Key Schedule Classification of the AES Candidates**, Proceedings of Second AES Candidate Conference (AES2), Rome, Italy, March 1999.

6. Daemen J. and Rijmen V., **AES Proposal: Rijndael**, AES Algorithm Submission (version 2), September 3, 1999, available at AES page available via http://www.nist.gov/CryptoToolkit. and at http://www.esat.kuleuven.ac.be/~rijmen/rijndael/
7. Dimovski A., Gligoroski D., **Attack On The Polyalphabetic Substitution Cipher Using A Parallel Genetic Algorithm**, Technical report, Swiss-Macedonian scientific cooperation trought SCOPES project, March 2003, Ohrid, Macedonia
8. Grundlingh W. and Vuuren J., **Using Genetic Algorithm to Break a Simple Cryptographic Cipher**, Available via http://dip.sun.ac.za/~vuuren/papers/genetic.ps
9. Mathews R., **The use of Genetic Algorithm in Cryptanalysis**, Cryptologia, Vol. 17 No. 4, 1993.
10. Nakaraha J., **Schedule key analysis of AES Candidates**, July 1999, available at http://www.esat.kuleuven.ac.be/~nakahara/aes_key_schedule.ps.gz

11. NIST 2001a. *Federal Information Processing Standards Publication (FIPS PUB)* 197. NIST, AES page available via http://www.nist.gov/publications

12. Rashed A., **Intelligent Encryption Decryption System Using Partial Genetic Algorithm and Rijndael Algorithm**, Ph.D. Thesis, Arab Academy for Banking and Financial Sciences, Computer Information System Department, 2004.
13. Spillman R., **Cryptanalysis of Knapsack using Genetic Algorithms**, Cryptologia, Volume 17 No. 1, 1993.

14. Spillman R. et al, **The Use a Genetic Algorithm in the Cryptanalysis of Simple Substitution Ciphers**, Cryptologia, Vol. 17, No. 1, 1993.
15. Yaseen I. and Sahasrabuddhe H., **A Genetic Algorithm for the Cryptanalysis of Chor-Rivest Knapsack Public Key Crypro System (PKC)**, Third International Conference on Computational Intelligence and Multimedia Application, September 23-26, 1999, New Delhi, India.

Appendix

All vectors are represented in hexadecimal format as a hexadecimal number can be represented as one byte as it consists of two digits (4 bits four one hexadecimal digit) These tests have been generated by RANA system; we use the same phrases that AES has been used. Date: Friday June 25 21:15:54 IDT 2004

Legend for cipher (round number r = 0 to 10, 12 or 14):

Input: cipher input

start: state at start of round[r]

s_box: state after SubBytes.subBytes(state)

s_col: state after Shif.shiftCols(state)

m_Row: state after MixRow. mixRow (state)

k_sch: key schedule value for round[r]

output: cipher output

Cipher Example: 128-bit cipher key: Example Vectors

Input String = 41 73 69 6d 20 41 20 45 6c 2d 53 68 65 69 6b 68

Cipher Key = 41 6c 20 61 62 6c 41 6c 64 61 62 69 75 68 64 52

And schedule key is as following:

parent

٤١	٦٢	٦٤	٧٥
c٦	6c	61	68
٢٠	٤١	٦٢	٦٤
٦١	c٦	69	52

child1

64	6c	64	64
62	45	21	41
6c	4c	28	52
64	62	65	61

child 2

٤٦	٢٦	c6	46
c6	54	c4	26
٤٦	١٢	٨٢	٥٦
٤٦	١٤	٢٥	١٦

child 3

٤٥	٦٢	٦٤	٢٢
٦٢	٤٤	٦٥	٦٤
٢٤	c٤	6c	62
٦١	c٦	4c	48

child 4

٥٤	٢٦	٤٦	٢٢
٢٦	٤٤	٥٦	٤٦
٤٢	c4	c6	26
١٦	c6	c4	84

child5

24	c6	26	c6
26	56	46	c6
c4	56	12	c6
54	26	c6	c6

child6

٤٢	c٦	٦٢	c٦
٦٢	٦٥	٦٤	c٦
c٤	٦٥	٢١	c٦
٤٥	٦٢	c٦	c٦

child7

٤٢	٤١	٦٤	٤٤
٤٤	٤٤	٦٢	c٦
٤٥	٦٤	٦٢	٤١
٦٢	٦٢	٦٥	٦١

Child 8

٢٤	١٤	٤٦	٤٤
٤٤	٤٤	٢٦	c6
٥٤	٤٦	٢٦	١٤
٢٦	٢٦	٥٦	١٦

Child 9

٢٢	٨٦	c4	c4
c4	٨٤	c6	٢٢
٢٦	٤٢	٢٦	٢٦
٤٢	c6	٥٦	٤٦

Child 10

٢٢	٦٨	c٤	c٤
c٤	٤٨	c٦	٢٢
٦٢	٢٤	٦٢	٦٢
٢٤	c٦	٦٥	٦٤

Figure 2: schedule key generated by MGA

r[0].input	4173696d204120456c2d536865696b68
r[0].k_sch	416c2061626c416c6461626975686452
r[0].input	001f490c422d6129084c310110010f3a
Round 1	
r[1].s_box	63c03bfe2cd8efa53029c77cca7c7680
r[1].s_row	63d8c7802c2976fe307c3ba5cac0ef7c
r[1].m_col	f21ab5a1ab1af0cc7a20ce4647074b92
r[1].start	9678d9c5c75fbcae1e01e623234619f3
r[1].s_sch	64626c646c454c626421286564415261
Round 2	
r[2].s_box	90bc35a6c6cf65e4727c8e26265ad40d
r[2].s_row	90cf8e0dc67cd4a6725a35e426bc6526
r[2].m_col	f2914ff061fff8aedb7d752ad0cc3aff
r[2].start	b45709b647abeaba1db9f70f96ea6ce9
r[2].s_sch	46c6464626541214c6c4822546265616
Round 3	
r[3].s_box	8d5b014ea06287f4a45668769087501e
r[3].s_row	8d62681ea056504ea48701f4905b8776
r[3].m_col	d1ef1dbabfb284613446268227c2449b
r[3].start	948d39dbddf6c80d50234ace05a626d3
r[3].s_sch	4562246162444c6c64656c4c22646248
Round 4	
r[4].s_box	225d12b9c142e8d75326d68b6b24f766
r[4].s_row	2242d666c126f7b9532412d76b5de88b
r[4].m_col	32a17d3ebd36c2e00ffa317652797b05
r[4].start	66873f289b72062649acf7b2703f5d81
r[4].s_sch	542642162644c4c64656c6c422462684
Round 5	
r[5].s_box	3317753414406ff73b91683751754c0c
r[5].s_row	3340680c14914c343b7575f751176f37
r[5].m_col	c207b765f8cd41896bb9a6b8c3f9c1e5
r[5].start	e62173313e9b17af4dffb47e053f0723
r[5].s_sch	2426c454c6565626264612c6c6c6c6c6
Round 6	
r[6].s_box	8efd8fc7b214f079e3168df36b75c526
r[6].s_row	8e148d26b216c5c7e3758f796bfdf0f3
r[6].m_col	900cf15c470d678bb4fa1836c972634d
r[6].start	d26ebd192b6802e9d69e395aa51e0f21
r[6].s_sch	42624c456c6565626264216c6c6c6c6c
Round 7	
r[7].s_box	b59f7ad4f145771ef60b12be067276fd
r[7].s_row	b54512fdf10b76d4f6727a1e069f77be
r[7].m_col	51f4c87246a971c6058252357f04ae85
r[7].start	13b08d1007ed15a461e030503b68efe4
r[7].s_sch	424445624144646264626265446c4161

Figure 3: Ciphering Phase (continue)

Round 8	
r[8].s_box	7de75dcac5555949efe10453e245df69
r[8].s_row	7d550469c5e1dfcaef455d49e2e75953
r[8].m_col	68b29b04bcacc4e51ecbcba0e78f4225
r[8].start	4cf6cf22a8e882c358ededf6a3495633
r[8].s_sch	244454261444462646262656 44c61416
Round 9	
r[9].s_box	29428a93c29b132e6a5555420a3bb1c3
r[9].s_row	299b55c3c255b1936a3b8a2e0a421342
r[9].m_col	72384628423340843db72c5383f9a8cb
r[9].start	50fc606ac4b70242f9710a0547db8e8d
r[9].s_sch	22c42642868442c6c4c62656c4222646
Round 10	
r[10].s_box	53b0d0021ca9772c99a3676ba0b9195d
r[10].s_box	53a9675d1ca3190299b9d02ca0b0776b
r[10].s_sch	416c2061626c416c6461626975686452
r[10].output	71e5057974eb3d6ed5d5b249ec92150f

Figure 3: Ciphering Phase

Cipher Example: 128-bit cipher key:

The following diagrams shows the hexadecimal values in the state array as the cipher progresses for a input block with length 16 bytes and a cipher key with 4 words too.

Input String = 41 736 96d 20 41 20 45 6c 2d 53 68 65 69 6b 68

Cipher Key = 41 6c 20 61 62 6c 41 6c 64 61 62 69 75 68 64 52

round number	start of round	after subbyte	after shiftrows	after mixrowss	round key values	
input	41 20 6c 65 73 41 2d 69 69 20 53 6b 6d 45 68 68				41 62 64 75 6c 6c 61 68 20 41 62 64 61 6c 69 52	=
1	00 42 08 10 1f 2d 4c 01 49 61 31 0f 0c 29 01 3a	٦٣ ٢c 30 ca c0 d8 29 7c ٣b ef c7 76 fe a5 7c 80	٦٣ c٢ 30 ca d8 29 7c c0 c7 76 3b ef ٨٠ fe a5 7c	f2 ab 7a 47 a١ 1a 20 07 b5 f0 ce 4b a1 cc 46 92	64 6c 64 64 62 45 21 41 6c 4c 28 52 64 62 65 61	=
2	٩٦ c7 1e 23 ٧٨ f٥ 01 46 d9 bc e6 19 c5 ae 23 f3	٩٠ c6 72 26 bc cf 7c 5a ٣٥ ٦٥ e٨ d4 a6 e4 26 0d	٩٠ c6 72 26 cf 7c 5a bc e٨ d4 35 65 ٠d a6 e4 26	f2 61 db d0 ٩١ ff 7d cc f٤ f8 75 3a f0 ae 2a ff	٤٦ ٢٦ c6 46 c6 54 c4 26 ٤٦ ١٢ ٨٢ ٥٦ ٤٦ ١٤ ٢٥ ١٦	=
3	b4 47 1d 96 ٥٧ ab b9 ea ٠٩ ea f7 6c b6 ba 0f e9	٨d a0 a4 90 ٥b 62 56 87 ٠١ ٨٧ ٦٨ ٥٠ e٤ f4 76 1e	٨d a0 a4 90 ٦٢ ٥٦ ٨٧ ٥b ٦٨ ٥٠ ٠١ ٨٧ e١ 4e f4 76	d1 bf 34 27 ef b2 46 c2 ١d 84 26 44 ba 61 82 9b	٤٥ ٦٢ ٦٤ ٢٢ ٦٢ ٤٤ ٦٥ ٦٤ ٢٤ c٤ 6c 62 ٦١ c٦ 4c 48	=
4	٩٤ dd 50 05 ٨ f6 23 a6	٢٢ c1 53 6b ٥ 42 26 24	٢٢ c1 53 6b ٤ ٢ ٢ ٥	٣٢ bd 0f 52 a1 36 fa 79	٥٤ ٢٦ ٤٦ ٢٢ ٢ ٤ ٥ ٤	=

Figure 4: Ciphering Phase(continue)

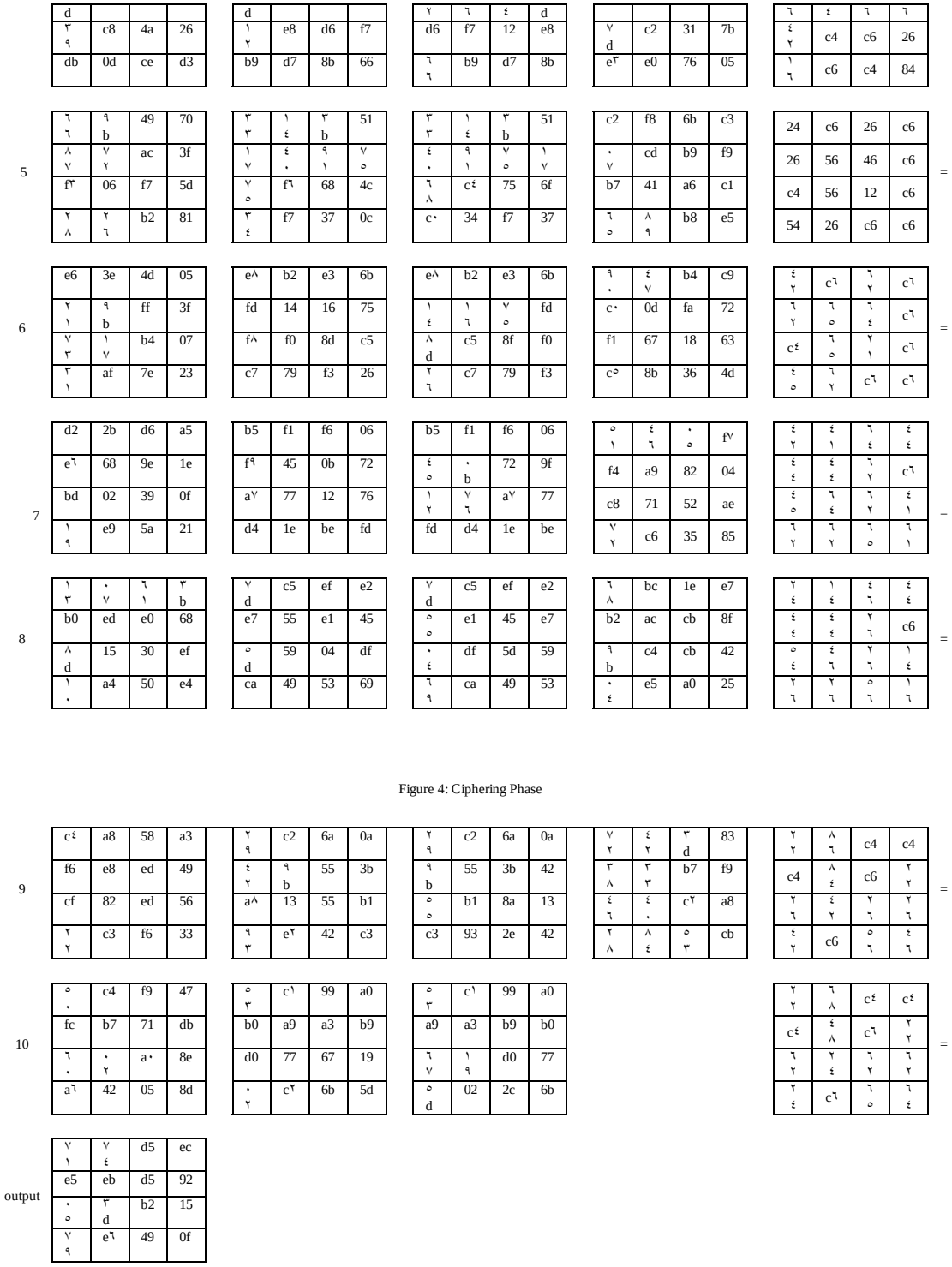


Figure 4: Ciphering Phase