

Key Generation Algorithm

KeyGeneration(Key, Sboxes, Blocks)

in: 1D integer array of length 32 Key[]; 2D array of 10 Sboxes [16][16]; integer number representing number of blocks in plaintext, Blocks

out: 2D array of Key DerivedKey[][]

1: key $\leftarrow$ Key

2: sboxes $\leftarrow$ Sboxes

3: **for** j $\leftarrow$ 0 ... (Blocks) **do**

4: key $\leftarrow$ preFunction(key)

5: **for** i $\leftarrow$ 0 ... (8) **do**

6: k1_i $\leftarrow$ key_i

7: **end for**

8: **for** i $\leftarrow$ 8 ... (16) **do**

9: k2_{i-8} $\leftarrow$ key_i

10: **end for**

11: **for** i $\leftarrow$ 16 ... (24) **do**

12: k3_{i-16} $\leftarrow$ key_i

13: **end for**

14: **for** i $\leftarrow$ 24 ... (32) **do**

15: k4_{i-24} = key_i

16: **end for**

17: Derived_Key, key $\leftarrow$ Expand_Key_256(k1,k2,k3,k4);

18: writee("Keys.txt",Derived_Key)

19: **end for**

Derive Key Generation Algorithm

DeriveKey(k1, k2, k3, k4)

in: 1D integer arrays Key1[8]; Key2[8]; Key3[8]; Key4[8]

out: 1D integer array Derived_Key[256]; 1D integer array Seed_Key[32]

1: DerivedKey1, seedKey1 $\leftarrow$ Key_Generation_Round(k1,k2,k3,k4

2: SCounter $\leftarrow$ 0

3: DerivedKey2, seedKey2 $\leftarrow$ Key_Generation_Round(k2,k1,k3,k4)

4: SCounter $\leftarrow$ 0

5: DerivedKey3, seedKey3 $\leftarrow$ Key_Generation_Round(k3,k1,k2,k4)

6: SCounter $\leftarrow$ 0

7: DerivedKey4, seedKey4 $\leftarrow$ Key_Generation_Round(k3,k1,k2,k4)

8: Derived_Key $\leftarrow$ DerivedKey1 + DerivedKey2 + DerivedKey3 + DerivedKey4

9: **return** Derived_Key

KeyMixture Algorithm

KeyMixture(DerivedKey)

1: SNumber $\leftarrow$ { 0, 1, 2, 3, 4, 5, 6, 7, 6, 7, 8, 9, 0, 1, 2, 2, 3, 4, 6, 7, 8, 5, 4, 5, 6, 9, 7, 0, 1, 6, 1, 0,
7, 9, 6, 3, 3, 2, 8, 6, 1, 7, 5, 1, 0, 5, 6, 2, 3, 9, 5, 9, 3, 0, 8, 2, 1, 8, 1, 7, 6, 4, 0, 2}

2: sbox $\leftarrow$ new int[16][16]

3: var $\leftarrow$ 0

4: **for** i $\leftarrow$ 0 ... (16) **do**

5: **for** j $\leftarrow$ 0 ... (16) **do**

6: sbox[i][j] $\leftarrow$ sboxes[SNumber[SCounter]][var]

7: var++

8: **end for**

```

9: end for

9:  $A \leftarrow \text{IntegerToBinary}(n)$ 

10:  $\text{row}, \text{column} \leftarrow \text{list}()$ 

11: for  $r \leftarrow 0 \dots (4)$  do

12:    $\text{row.append}(A_r)$ 

13: end for

14: for  $c \leftarrow 4 \dots (8)$  do

15:    $\text{column.append}(A_r)$ 

16: end for

17: for  $i \leftarrow 0 \dots (4)$  do

18:    $\text{Row} \leftarrow \text{Row} + \text{IntegerToString}(\text{row}_i)$ 

19: end for

20: for  $i \leftarrow 0 \dots (4)$  do

21:    $\text{Col} \leftarrow \text{Col} + \text{IntegerToString}(\text{column}_i)$ 

22: end for

23:  $\text{decRow} \leftarrow \text{BinaryToDecimal}(\text{Row})$ 

24:  $\text{decCol} \leftarrow \text{BinaryToDecimal}(\text{Col})$ 

25:  $\text{sblock} \leftarrow \text{sbox}[\text{decRow}][\text{decCol}]$ 

26:  $\text{SCounter}++;$ 

27: return  $\text{sblock}$ 

```

Round Key Generation Algorithm

Key_Generation_Round(Key1, Key2, Key3, Key4)

in: 1D integer arrays Key1[8]; Key2[8]; Key3[8]; Key4[8]

out: 1D integer DerivedKey[64], seedKey[8]

1: $\text{DerivedKey} \leftarrow \text{new int}[64]$

```

2: seedKey  $\leftarrow$  new int[8]
3: S0  $\leftarrow$  SelectionOfSbox(Key10)
4: DerivedKey0  $\leftarrow$  S0 XOR Key20
5: DerivedKey0  $\leftarrow$  DerivedKey0 XOR Key31
6: for i  $\leftarrow$  1 ... (8) do
7:     S  $\leftarrow$  sbox(Key1i)
8:     if (i==7) then
9:         seedKey0 = S XOR DerivedKeyi-1
10:    else
11:        DerivedKeyi  $\leftarrow$  S XOR DerivedKeyi-1
12:        DerivedKeyi  $\leftarrow$  Key31 XOR DerivedKeyi
13:    end if
14: end for
15: seedKey0  $\leftarrow$  seedKey0 XOR Key42
16: Levels(DerivedKey, seedKey, 0,0, Key21, Key32)           ▶ Generating 7-13
17: seedKey1  $\leftarrow$  seedKey1 XOR Key43
18: Levels(DerivedKey, seedKey, 7, 1, Key22, Key33)         ▶ Generating 14-20
19: seedKey2  $\leftarrow$  seedKey2 ^ Key44
20: Levels(DerivedKey, seedKey, 14, 2, Key23, Key34)         ▶ Generating 21-27
21: seedKey3  $\leftarrow$  seedKey3 XOR Key45
22: Levels(DerivedKey, seedKey, 21, 3, Key24, Key35)         ▶ Generating 28-34
23: seedKey4  $\leftarrow$  seedKey4 XOR Key46
24: Levels(DerivedKey, seedKey, 28, 4, Key25, Key36)         ▶ Generating 35-41
25: seedKey5  $\leftarrow$  seedKey5 XOR Key47
26: Levels(DerivedKey, seedKey, 35, 5, Key26, Key37)         ▶ Generating 42-48

```

```

27: seedKey6 ← seedKey6 XOR Key10
28: Levels(DerivedKey, seedKey, 42, 6, Key27, Key40)           ▷ Generating 49-55
29: seedKey7 ← seedKey7 XOR Key11
30: x ← 49
31: S0 ← SelectionOfSbox(DerivedKeyx)                       ▷ Generating 56-64
32: DerivedKeyx+7 ← S0 XOR (seedKey7 XOR Key30)
33: x ← x+8
34: for i ← x ... (x+6) do
35:   S ← SelectionOfSbox(DerivedKeyi-7)
36:   DerivedKeyi ← S XOR DerivedKeyi-1
37:   DerivedKeyi ← Key41 XOR DerivedKeyi
38: end for
39: DerivedKey63 ← Key42 XOR DerivedKey62
40: return DerivedKey, seedKey

```

Round Encryption Algorithm

Encryption_Round(integers, round)

in: 1D integer array integer[]; integer number representing current round number round

out: 1D integer array Pbox[]

1: header_i++

2: cipher ← list()

3: **if** (randomNumber%2==0) **then** ▷ selecting the key

4: Knum ← RandomNumber(blocksnumber)

5: **else**

6: Knum ← Keys[round][CounterK[round]]

7: header_arr[header_i] ← Knum

```

8: key  $\leftarrow$  Key[Knum]
9: for i  $\leftarrow$  0 ... (|integers|) do
10:   cipher.append(integersi XOR keyi)
11: end for
12: Sbox  $\leftarrow$  Substitution_Process(cipher,round)
13: Pbox  $\leftarrow$  Permutation_Process(Sbox,round)
14: return Pbox

```

Round Substitution Algorithm

Substitution_Process (cipher, round)

in: 1D integer array cipher[]; integer number representing current round number round

out: 1D integer array sbox[]

```

1: header_i++;
2: cipher1  $\leftarrow$  list()
3: if (randomNumber%2==0) then
4:   Snum  $\leftarrow$  RandomNumber(blocksnumber)
5: else
6:   Snum = Sboxes[round][CounterS[round]];
7: end if
8: duplication  $\leftarrow$  "No"
9: for i  $\leftarrow$  0 ... (|SboxFile|) do
10:   if (Snum == SboxFilei) then
11:     duplication = "Yes";
12:   end if
13: end for
14: if (duplication=="No") then

```

```

15:   SboxFile.add(Snum)
16: end if
17: header_arr[header_i]  $\leftarrow$  Snum
18: sbox  $\leftarrow$  Sbox[Snum]
19: var  $\leftarrow$  0
20: for i  $\leftarrow$  0 ... (16) do
21:   for j  $\leftarrow$  0 ... (16) do
22:     Sbox2Di,j  $\leftarrow$  sboxvar
23:     var++
24:   end for
25: end for
26: for f  $\leftarrow$  0 ... (|cipher|) do
27:   value  $\leftarrow$  cipherf
28:   row  $\leftarrow$  value div 16
29:   col  $\leftarrow$  value mod 16
30:   cipher1f  $\leftarrow$  Sbox2Drow,col
31: end for
32: return cipher1

```

Round Permutation Algorithm

Permutation_Process(cipher, round)

in: 1D integer array cipher[]; integer number representing current round number round

out: 1D integer array pbox[]

```

1: header_i++;
2: pcipher  $\leftarrow$  list()
3: if (randomNumber%2==0) then

```

```

4:   Pnum  $\leftarrow$  RandomNumber(blocksnumber)
5: else
6:   Pnum  $\leftarrow$  Pboxes[round][CounterP[round]]
7: duplication  $\leftarrow$  "No"
8: for i  $\leftarrow$  0 ... (|PboxFile|) do
9:   if (Pnum == PboxFilei) then
10:     duplication = "Yes"
11:   end if
12: end for
13: if (duplication=="No")
14:   PboxFile.add(Pnum)
15: end if
16: header_arr[header_i]  $\leftarrow$  Pnum
17: pbox  $\leftarrow$  Pbox[Pnum]
18: var  $\leftarrow$  0
19: for i  $\leftarrow$  0 ... (16) do
20:   for j  $\leftarrow$  0 ... (16) do
21:     Pbox2Di,j  $\leftarrow$  pboxvar
22:     var++
23:   end for
24: end for
25: for w  $\leftarrow$  0 ... (16) do
26:   for (int s=0; s<16; s++) do
27:     temp  $\leftarrow$  Pbox2Dw,s
28:     pciphery  $\leftarrow$  ciphertemp

```



```

29:         y++
30:     end for
31: end for
32: return pcipher

```

Block Extraction Algorithm

Block_Extraction(fullfile)

in: 1D integer array fullfile[]

out: 1D integer array blockarray[256]

1: global increment

2: increment $\leftarrow$ increment + 256 $\triangleright$ for 1st time increment=0 then increment =256 then increment =768

3: blockarray $\leftarrow$ |256|

4: **for** y $\leftarrow$ 0 ... (256) **do**

5: x $\leftarrow$ increment + y

6: blockarray_y=fullfile_x

7: **end for**

8: **return** blockarray

Algorithm for Constructing Protocol Headers

DynamicNetwork(intarray)

in: 1D integer array intarray[]

out: 1D integer array Full_array[]

1: sizeofheader $\leftarrow$ 0

2: round_array $\leftarrow$ |blocksnumber|

3: minlimit $\leftarrow$ blocksnumber div 2

4: **for** i $\leftarrow$ 0 ... (blocksnumber) **do**

```

5:   x ← RandomNumber(blocksnumber-minlimit)+minlimit+1
6:   round_array [i] ← x
7:   size ← 1+(x*3)
8:   sizeofheader +=size
9: end for
10: fullarray ← |sizeofheader+intarray.length|
11: maxRound ← 0
12: for i ← 0 ... (blocksnumber) do
13:   if (maxRound < round_arrayi) then
14:     maxRound ← round_arrayi
15:   end if
16: end for
17: for iteration ← 0 ... (blocksnumber) do
18:   header_i ← 0
19:   block_array ← Block_Extraction(intarray)
20:   no_of_rounds ← round_arrayiteration
21:   header_arr_size ← 1 + (no_of_rounds*3)
22:   header_arr ← |header_arr_size|
23:   header_arr[header_i] ← no_of_rounds
24:   Round ← block_array
25:   for i ← 1 ... (no_of_rounds) do
26:     Round ← Encryption_Round(Round,i)
27:     Sboxes[iteration][i-1] ← Snum
28:     Pboxes[iteration][i-1] ← Pnum
29:     Keys[iteration][i-1] ← Knum

```

```

30:   end for
31:   for yy  $\leftarrow$  0 ... (header_arr_size) do
32:       finalDatayy  $\leftarrow$  header_arryy
33:   end for
34:   for i  $\leftarrow$  0 ... (|Round|) do
35:       x  $\leftarrow$  i+header_arr_size
36:       finalDatax  $\leftarrow$  Roundi
37:   end for
38:   Full_array  $\leftarrow$  Full_array + finalData
39: end for
40: return Full_array

```

Algorithm of Constructing Round Headers

DynamicNetwork2(intarray)

in: 1D integer array intarray[]

out: 1D integer array Full_array[]

```

1: sizeofheader  $\leftarrow$  0
2: no_of_rounds  $\leftarrow$  RandomNumber(blocksnumber) + 1
3: size  $\leftarrow$  1+(no_of_rounds*3)
4: if (blocksnumber mod 2==0 && randomNumber mod 2==0) then
5:     sizeofheader  $\leftarrow$  size*(blocksnumber/2)
6: else
7:     sizeofheader  $\leftarrow$  size*blocksnumber
8: for i  $\leftarrow$  0 ... (no_of_rounds) do
9:     number_of_sboxesi  $\leftarrow$  RandomNumber(blocksnumber) + 1
10:    number_of_pboxesi  $\leftarrow$  RandomNumber(blocksnumber) + 1

```

```

11:   number_of_keysi ← RandomNumber(blocksnumber) + 1
12: end for
13: for i ← 0 ... (no_of_rounds) do
14:   for j ← 0 ... (number_of_sboxesi) do
15:     Sboxesi,j ← RandomNumber(blocksnumber)
16:   end for
17:   for j ← 0 ... (number_of_pboxesi) do
18:     Pboxesi,j ← RandomNumber(blocksnumber)
19:   end for
20:   for j ← 0 ... (number_of_keysi) do
21:     Keysi,j ← RandomNumber(blocksnumber)
22:   end for
23: end for
24: fullarray ← |sizeofheader+(blocksnumber*2)+intarray.length|
25: if (blocksnumber mod 2==0 && randomNumber mod 2==0) then
26:   fullarray ← |sizeofheader+blocksnumber+ intarray.length|
27: end if
28: if (blocksnumber mod 2==0 && randomNumber mod 2==0) then
29:   for (iteration ← 0 ... (blocksnumber/2)) do
30:     headeri ← 0
31:     blockarr1 ← Block_Extraction (intarray)
32:     blockarr2 ← Block_Extraction (intarray)
33:     header_arr_size ← 1+(no_of_rounds*3)
34:     header_arr ← new int[header_arr_size]
35:     header_arr[headeri] ← no_of_rounds

```

```

36:      for (i  $\leftarrow$  0 ... (no_of_rounds) do
37:          if (CounterSi >= number_of_sboxesi) then
38:              CounterSi  $\leftarrow$  0
39:          if (CounterPi >= number_of_pboxesi) then
40:              CounterPi  $\leftarrow$  0
41:          if (CounterKi >= number_of_keysi) then
42:              CounterKi  $\leftarrow$  0
43:              blockarr1, blockarr2  $\leftarrow$  Feistel_Structure(blockarr1,blockarr2,i)
44:              CounterSi++
45:              CounterPi++
46:              CounterKi++
47:          end for
48:      for yy  $\leftarrow$  0 ... (header_arr_size) do
49:          finalDatayy  $\leftarrow$  header_arryy
50:      end for
51:      for i  $\leftarrow$  0 ... (|blockarr2|) do
52:          x  $\leftarrow$  i+header_arr_size
53:          finalDatax  $\leftarrow$  blockarr2i
54:      end for
55:      for i  $\leftarrow$  0 ... (|blockarr1|) do
56:          x  $\leftarrow$  i+header_arr_size+256
57:          finalDatax  $\leftarrow$  blockarr1i
58:      end for
59:      Full_array  $\leftarrow$  Full_array + finalData
60:  end for

```

```

61: else
62:   for iteration  $\leftarrow$  0 ... (blocksnumber) do
63:     header_i  $\leftarrow$  0
64:     block_array  $\leftarrow$  Block_Extraction(intarray)
65:     no_of_rounds  $\leftarrow$  round_arrayiteration
66:     header_arr_size  $\leftarrow$  1 + (no_of_rounds*3)
67:     header_arr  $\leftarrow$  |header_arr_size|
68:     header_arr[header_i]  $\leftarrow$  no_of_rounds
69:     for i  $\leftarrow$  1 ... (no_of_rounds) do
70:       if (CounterSi >= number_of_sboxesi) then
71:         CounterSi  $\leftarrow$  0
72:       if (CounterPi >= number_of_pboxesi) then
73:         CounterPi  $\leftarrow$  0
74:       if (CounterKi >= number_of_keysi) then
75:         CounterKi  $\leftarrow$  0
76:       Round  $\leftarrow$  Encryption_Round(Round,i)
77:       CounterSi++
78:       CounterPi++
79:       CounterKi++
80:     end for
81:     for yy  $\leftarrow$  0 ... (header_arr_size) do
82:       finalDatayy  $\leftarrow$  header_arryy
83:     end for
84:     for i  $\leftarrow$  0 ... (|Round|) do
85:       x  $\leftarrow$  i+header_arr_size

```

```

86:          finalDatax ← Roundi
87:      end for
88:      Full_array ← Full_array + finalData
89:  end for
90: end if
91: return Full_array

```

Algorithm of Constructing Round Structure

Feistel_Structure(block0,block1,round)

in: 1D integer array, block0; 1D integer array, block1, integer number representing round

out: 1D integer array block1[]; 1D integer array cipher [];

```

1: header_i++
2: Knum ← Keys[round][CounterK[round]]
3: header_arrheader_i ← Knum
4: key ← Key[Knum]
5: for i ← 0 ... (256) do
6:     cipheri ← block1i ^ keyi
7: end for
8: sbox1 ← Substitution_Process (cipher,round)
9: pbox1 ← Permutation_Process (sbox1,round)
10: for i ← 0 ... (256) do
11:     cipheri ← block0i ^ pbox1i
12: end for
13: return block1,cipher

```
