

1. Algorithm of S-boxes generation based on Quantum Random Bits

SboxGeneration(decimalArray,Block)

IN: 1D integer array decimalArray[];

OUT: 1D integer array UniqueArray[];

1: block $\leftarrow$ Block

2: **if** block < 10 **then**

3: block $\leftarrow$ 10

4: **end if**

5: **for** x $\leftarrow$ 0 ... (|decimalArray|) **do**

6: Numbers $\leftarrow$ list()

7: **for** i $\leftarrow$ x ... (|decimalArray|) **do**

8: Numbers.append(decimalArray_i)

9: **end for**

10: UniqueArray, index $\leftarrow$ UniqueNumber(Numbers)

11: x $\leftarrow$ x + index

12: **if** |UniqueArray| == 256 **then**

13: Nonlinearity $\leftarrow$ CalculateNonlinearity(UniqueArray)

14: **for** i $\leftarrow$ 0 ... (|Nonlinearity|) **do**

15: sum $\leftarrow$ sum + Nonlinearity

16: **end for**

17: **if** TotalSbox >= block **then**

18: break

19: **else if** ((sum div 8) > 104) **then**

20: TotalSbox ++;

21: writee("sboxes.txt", UniqueArray);

22: **else**

23: break

24: **end if**

25: **end for**

26: **return** TotalSbox

Walsh_Transform (BinaryArray)

IN: 1D integer array of 0,1 BinaryArray[]

OUT: 1D integer array walsh_transform[]

```
1: walsh_transform  $\leftarrow$  list()
2: for i  $\leftarrow$  0 ... | BinaryArray | do
3:     if BinaryArrayi  $\leftarrow$  0 then
4:         walsh_transform.append(1)
5:     else
6:         walsh_transform.append(-1)
7:     end if
8: end for
9: order  $\leftarrow$  |BinaryArray|
10: n  $\leftarrow$  8
11: size  $\leftarrow$  order div 2
12: while (size >= 1) do
13:     left  $\leftarrow$  0;
14:     while (left < order - 1) do
15:         for p  $\leftarrow$  0 ... (size) do
16:             right  $\leftarrow$  left + size;
17:             a  $\leftarrow$  walsh_transformleft
18:             b  $\leftarrow$  walsh_transformright
19:             walsh_transformleft  $\leftarrow$  a + b
20:             walsh_transformright  $\leftarrow$  a - b
21:             left  $\leftarrow$  left + 1
22:         end for
23:         left  $\leftarrow$  right + 1
24:     end while
25:     size  $\leftarrow$  size div 2
26: end while
27: return walsh_transform
```

BF_Nonlinearity(BinaryArray, n)

IN: 1D integer array of 0,1 BinaryArray[], integer number n

OUT: Double value x

1: walsh_transform $\leftarrow$ Walsh_Transform (BinaryArray)

2: **for** i $\leftarrow$ 0 ... |walsh_transform| **do**

3: value $\leftarrow$ abs(walsh_transform_i)

4: walsh_transform_i $\leftarrow$ value

5: **end for**

6: max $\leftarrow$ 0

7: **for** i $\leftarrow$ 0 ... |walsh_transform| **do**

8: value $\leftarrow$ walsh_transform_i

9: **if** (max < value) **then**

10: max = value;

11: **end if**

12: **end for**

13: x $\leftarrow$ (n-1)² - (max div 2)

14: **return** x

CalculateNonlinearity (Sbox)

IN: 1D integer array containing numbers from 0-255 Sbox[]

OUT: 1D array having nonlinearities of S-box, n1_array[]

```
1: order  $\leftarrow$  |Sbox|
2: n  $\leftarrow$  8
3: n1_array  $\leftarrow$  list()
4: for bitno  $\leftarrow$  0 ... (n) do
5:     f  $\leftarrow$  list()
6:     for index  $\leftarrow$  0 ... (order) do
7:         binary_array  $\leftarrow$  IntegerToBinary(Sboxindex)
8:         bit  $\leftarrow$  binary_arraybitno
9:         f.append(bit)
10:    end for
11:    bfnl  $\leftarrow$  BF_Nonlinearity(f, n)
12:    n1_array.append(bfnl)
13: end for
14: return n1_array
```

2. Algorithm of P-boxes generation based on Quantum Random Bits

PboxGeneration(decimalArray,Block)

IN: 1D integer array decimalArray[];

OUT: 1D integer array UniqueArray[];

1: block $\leftarrow$ Block

2: **for** x $\leftarrow$ 0 ... (|decimalArray|) **do**

3: Numbers $\leftarrow$ list()

4: **for** i $\leftarrow$ x ... (|decimalArray|) **do**

5: Numbers.append(decimalArray_i)

6: **end for**

7: UniqueArray, index $\leftarrow$ UniqueNumber(Numbers)

8: x $\leftarrow$ x + index

9: **if** |UniqueArray| == 256 **then**

10: **if** TotalPbox == block **then**

11: break

12: writee("pboxes.txt", UniqueArray)

13: TotalPbox++

14: **else**

15: break

16: **end if**

17: **end for**

18: **return** TotalPbox

3. Algorithm of Reverse S-boxes Generation

ReverseSboxGeneration(SboxArray,Block)

IN: 1D integer array containing numbers in range 0-225 SboxArray[]

OUT: Reverse of input array reverseSbox1D[]

```
1: sbox  $\leftarrow$  new int[16][16]
2: k  $\leftarrow$  0
3: for row  $\leftarrow$  0 ... (16) do
4:   for col  $\leftarrow$  0 ... (16) do
5:     sbox[row][col]  $\leftarrow$  SboxArrayk
6:     k++
7:   end for
8: end for
9: ReverseSbox  $\leftarrow$  new int[16][16]
10: for row  $\leftarrow$  0 ... (16) do
11:   for col  $\leftarrow$  0 ... (16) do
12:     rowRS  $\leftarrow$  sbox[row][col] div 16
13:     colRS  $\leftarrow$  sbox[row][col] mod 16
14:     value  $\leftarrow$  row*16 + col
15:     ReverseSbox[rowRS][colRS]  $\leftarrow$  value
16:   end for
17: end for
18: reverseSbox1D  $\leftarrow$  list()
19: for row  $\leftarrow$  0 ... (16) do
20:   for col  $\leftarrow$  0 ... (16) do
21:     reverseSbox1D.add(ReverseSbox[row][col])
22:   end for
23: end for
24: return reverseSbox1D
```
