Algorithm 1: Appling rule R1 on block b_i in odd time step t;

input : bs_i^{t-1}: cell states of block b_i at time $t-1$.
output: bs_i^t: cell states of block b_i at time t.

$bs_i^t[0] \leftarrow IntegerPart(\frac{bs_i^{t-1}[1]}{2})$;
$bs_i^t[1] \leftarrow null$;
$bs_i^t[2] \leftarrow FractionalPart(\frac{bs_i^{t-1}[1]}{2})$;
$bs_i^t[3] \leftarrow null$;

Algorithm 2: Appling rule R2 on block b_i in even time step t;

input : bs_i^{t-1}: cell states of block b_i at time $t-1$.
output: bs_i^t: cell states of block b_i at time t.

$bs_i^t[0] \leftarrow null$;
if *(i is equal to the number of blocks)* **then**
  $bs_i^t[1] \leftarrow bs_{i-1}^{t-1}[0]$;
else if *(i is equal to zero)* **then**
  $bs_i^t[1] \leftarrow bs_i^{t-1}[2]$;
else
  $bs_i^t[1] \leftarrow bs_i^{t-1}[2] + bs_{i-1}^{t-1}[0]$;
$bs_i^t[2] \leftarrow null$;
$bs_i^t[3] \leftarrow null$;

Algorithm 3: Sum of D2CA(x) and D2CA(y)

input : D2CA(x) and D2CA(y).
T: number of levels (time steps).

output: D2CA(z) = D2CA(x) + D2CA(y).

```
d ← Number of digit in x or y;
m ← Number of blocks in time step t;
for (t = 0 to T − 1) do
    bc ← ⌊t/2⌋ + 2; // block counts
    c ← 0; //carry digit
    if (t is even) then
        for (i = 0 to bc − 1) do
            sum ← (bs_i^t[1])_x + (bs_i^t[1])_y + c;
            if (sum >= 10) then
                (bs_i^t[1])_z ← sum − 10;
                c ← 1;
            else
                (bs_i^t[1])_z ← sum;
                c ← 0;
        if (c == 1) then
            (bs_{i+1}^t[1])_z ← 1;
    else //t is odd
        for (i = 0 to bc − 1) do
            sum ← (bs_i^t[2])_x + (bs_i^t[2])_y + c;
            if (sum >= 10) then
                (bs_i^t[2])_z ← sum − 10;
                c ← 1;
            else
                (bs_i^t[2])_z ← sum;
                c ← 0;
            sum ← (bs_i^t[0])_x + (bs_i^t[0])_y + c;
            if (sum > 4) then
                (bs_i^t[0])_z ← sum − 5;
                c ← 5;
            else
                (bs_i^t[0])_z ← sum;
                c ← 0;
        if (c > 0) then
            (bs_{i+1}^t[2])_z ← c;
            (bs_{i+1}^t[0])_z ← 0;
```

Algorithm 4: Produce subkeys in LSC algorithm

input : D2CA(key1) and D2CA(key2).
output: $subkey$.

$numberOfLozenge \leftarrow \lceil |plaintext|/8 \rceil$; //each lozenge consists of 8 cells
$L1 \leftarrow Extract\_Lozenge(D2CA(key1), numberOfLozenge)$;
$L2 \leftarrow Extract\_Lozenge(D2CA(key2), numberOfLozenge)$;
$subkey_i \leftarrow [(L1_{ij} + L2_{ij} * 169)], \forall i = 1..numberOfLozenge, j = 1..8$; //i: lozenge index, j: cell index in lozenge, 169 is arbitrary constant number

Algorithm 5: Division by Replacement)

input : x, a list of digits in number X.
output: y, the quotients of X/2 .

```
y[0] ← ((x[0] * 5) mod 10)/10;
for (i = 0 to n) do
    d1 ← x[i];
    d2 ← x[i + 1];
    if (d1 is even) then
        switch d2 do
            case 0,1 do
                y[i] ← 0
            case 2,3 do
                y[i] ← 1
            case 4,5 do
                y[i] ← 2
            case 6,7 do
                y[i] ← 3
            case 8,9 do
                y[i] ← 4
    else
        switch d2 do
            case 0,1 do
                y[i] ← 5
            case 2,3 do
                y[i] ← 6
            case 4,5 do
                y[i] ← 7
            case 6,7 do
                y[i] ← 8
            case 8,9 do
                y[i] ← 9
y[n + 1] ← decimal section of (x[n] * 5)/10;
```