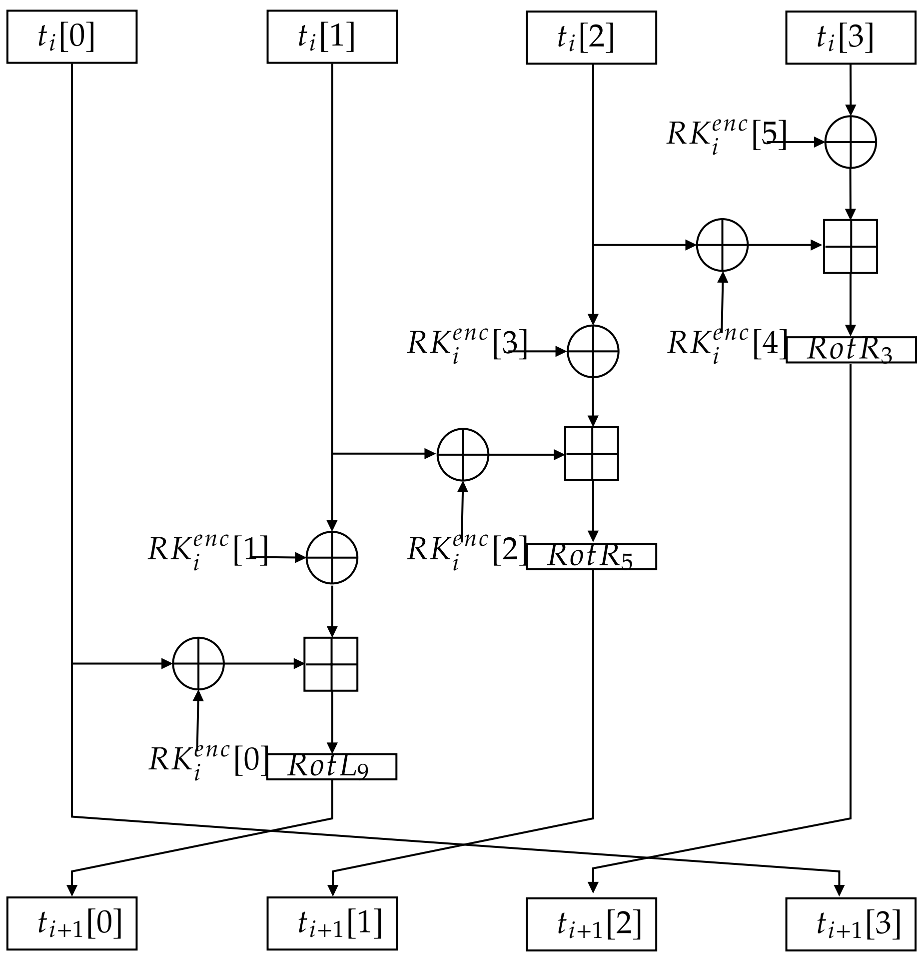
Lightweight Encryption Algorithm
- LEA -

Ji Yong-Hyeon



Department of Information Security, Cryptology, and Mathematics
College of Science and Technology
Kookmin University

January 11, 2024

List of Symbols

Contents

Chapter 1

LEA: Implementation in Little-Endian

1.1 Specification

Table 1.1: Specification Comparison between AES and LEA Block Ciphers

Specification	**AES**	**LEA**
Block Size (bits)	128	128
Key Size (bits)	128/192/256	128/192/256
Structure	Substitution-Permutation Network	Generalized Feistel Network
Rounds	10/12/14 (depends on key size)	24/28/32 (depends on key size)
Design Year	1998	2013

Table 1.2: Parameters of the Block Cipher LEA (1-word = 32-bit)

Algorithms	Block Size (N_b-byte)	Key Length (N_k-byte)	Number of Rounds (N_r)	Round-Key Length (byte)	Total Size of Round-Keys ((N_r * 192)-bit)
LEA-128	16(4-word)	16(4-word)	24	24	4608 (144-word)
LEA-192	16(4-word)	24(6-word)	28	24	5376 (168-word)
LEA-256	16(4-word)	32(8-word)	32	24	6144 (192-word)

1.2 State Representation

Let $\mathsf{state}[0], \mathsf{state}[1], \ldots$ be representation of arrays of bytes. Note that

$$\mathsf{state}[i] := \left\{input_{8i}, input_{8i+1}, \ldots, input_{8i+7}\right\} \in \mathbb{F}_{2^8}$$

for $input_i \in \mathbb{F}_2$. For example, $\mathsf{state}[0] = \left\{input_0, input_1, \ldots, input_7\right\}$.

The 128-bit plaintext P of LEA is represented as an array of four 32-bit words $P[0], P[1], P[2]$ and $P[3]$. Then

$$P[i] = \mathsf{state}[4i+3] \parallel \mathsf{state}[4i+2] \parallel \mathsf{state}[4i+1] \parallel \mathsf{state}[4i] \quad \text{for} \quad 0 \le i \le 3.$$

Here, $P[i] \in \mathbb{F}_{2^{32=8\cdot 4}}$The key K of LEA is also represented as the same way.

Table 1.3: Representations for words, bytes, and bits

Input Bit Sequence	24	⋯	31	16	⋯	23	8	⋯	15	0	⋯	7
Word Number	0											
Byte Number	3			2			1			0		
Bit Numbers in Word	31	⋯										1

Example 1.1.

128-bit Input String	`0x0f1e2d3c4b5a69788796a5b4c3d2e1f0`											
Split into Words	0x0f1e2d3c $P[0]$			0x4b5a6978 $P[1]$			0x8796a5b4 $P[2]$			0xc3d2e1f0 $P[3]$		
$P[0]$ **(Word)**	0x0f1e2d3c											
$P[0]$ **(Bit)**	0b 0000:1111:0001:1110:0010:1101:0011:1010											
Split into Bytes	0x0f state[3]			0x1e state[2]			0x2d state[1]			0x3c state[0]		
state[0] **(Byte)**	0x3c											
Split into Bits	1111:0000			-			-			-		
	24	⋯	31	16	⋯	23	8	⋯	15	0	⋯	7

```c
void stringToWordArray(const char* hexString, u32* wordArray) {
    size_t length = strlen(hexString);
    for (size_t i = 0; i < length; i += 8) {
        sscanf(&hexString[i], "%8x", &wordArray[i / 8]);
    }
}

const char* inputString = "0f1e2d3c4b5a69788796a5b4c3d2e1f0";
u32 key[4];
stringToWordArray(inputString, key);
```

```
(gdb) x/16xb key
0x7fffffffd9c0: 0x3c    0x2d    0x1e    0x0f    0x78    0x69    0x5a    0x4b
0x7fffffffd9c8: 0xb4    0xa5    0x96    0x87    0xf0    0xe1    0xd2    0xc3
```

1.3 Key Schedule

$$\text{KeySchedule}^{\text{enc}}_{128} : \{\mathtt{0}, 1\}^{128=8\cdot 16} \longrightarrow \{\mathtt{0}, 1\}^{4608=192\cdot 24}$$

$$\text{KeySchedule}^{\text{enc}}_{192} : \{\mathtt{0}, 1\}^{192=8\cdot 24} \longrightarrow \{\mathtt{0}, 1\}^{5376=192\cdot 28}$$

$$\text{KeySchedule}^{\text{enc}}_{256} : \{\mathtt{0}, 1\}^{256=8\cdot 32} \longrightarrow \{\mathtt{0}, 1\}^{6144=192\cdot 24}$$

1.3.1 Round Constant

The constant $\delta[i] \in \mathbb{F}_{2^{32}}$ ($i \in \{1, \ldots, 7\}$) is as follows:

i	$\delta[i]$	value
0	$\delta[0]$	`0xc3efe9db`
1	$\delta[1]$	`0x44626b02`
2	$\delta[2]$	`0x79e27c8a`
3	$\delta[3]$	`0x78df30ec`
4	$\delta[4]$	`0x715ea49e`
5	$\delta[5]$	`0xc785da0a`
6	$\delta[6]$	`0xe04ef22a`
7	$\delta[7]$	`0xe5c40957`

1.3.2 Rotation Function

Algorithm 1: Rotation to Left and Right

/* $\mathtt{RotL} : \{0, 1\}^{32} \times \{0, 1\}^{32} \to \{0, 1\}^{32}$ */

Function RotL(value, shift):

  return (value $\ll$ shift) | (value $\gg$ (32 − shift));

end

/* $\mathtt{RotR} : \{0, 1\}^{32} \times \{0, 1\}^{32} \to \{0, 1\}^{32}$ */

Function RotR(value, shift):

  return (value $\gg$ shift) | (value $\ll$ (32 − shift));

end

1.3.3 Encryption Key Schedule of LEA-128

Algorithm 2: Encryption Key Schedule (LEA-128)

Input: User-key $\mathsf{UK} = \mathsf{UK}[0] \parallel \mathsf{UK}[1] \parallel \mathsf{UK}[2] \parallel \mathsf{UK}[3]$ ($\mathsf{UK}[i] \in \{0, 1\}^{32}$)

Output: Encryption Round-keys $\{\mathsf{RK}_i^{\mathsf{enc}}\}_{i=0}^{23}$ ($\mathsf{RK}_i^{\mathsf{enc}} \in \{0, 1\}^{192}$)

/* $\mathsf{UK} \in \{0, 1\}^{128}$ is 16-byte and $\left\{\mathsf{RK}_i^{\mathsf{enc}}\right\}_{i=0}^{23} \in \{0, 1\}^{4608}$ is 576-byte */

for $i = 0$ **to** 3 **do**

  $T[i] = \mathsf{UK}[i]$ // $T = T[0] \parallel \cdots \parallel T[3] \in \{0, 1\}^{128=32*4}$

end

for $i = 0$ **to** 23 **do**

  $T[0] \leftarrow \mathtt{RotL}(T[0] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i + 0), 1)$ // $T[i] \in \{0, 1\}^{32}$

  $T[1] \leftarrow \mathtt{RotL}(T[1] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i + 1), 3)$

  $T[2] \leftarrow \mathtt{RotL}(T[2] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i + 2), 6)$

  $T[3] \leftarrow \mathtt{RotL}(T[3] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i + 3), 11)$

  $\mathsf{RK}_i^{\mathsf{enc}} \leftarrow T[0] \parallel T[1] \parallel T[2] \parallel T[1] \parallel T[3] \parallel T[1]$ // $\mathsf{RK}_i^{\mathsf{enc}} \in \{0, 1\}^{196=32*6}$

end

return $\left\{\mathsf{RK}_i^{\mathsf{enc}}\right\}_{i=0}^{23}$

1.3.4 Decryption Key Schedule of LEA-128

Algorithm 3: Decryption Key Schedule (LEA-128)

Input: User-key $\mathsf{UK} = \mathsf{UK}[0] \parallel \mathsf{UK}[1] \parallel \mathsf{UK}[2] \parallel \mathsf{UK}[3]$ ($\mathsf{UK}[i] \in \{0,1\}^{32}$)

Output: Decryption Round-keys $\{\mathsf{RK}_i^{\mathsf{dec}}\}_{i=0}^{23}$ ($\mathsf{RK}_i^{\mathsf{dec}} \in \{0,1\}^{192}$)

/* $\mathsf{UK} \in \{0,1\}^{128}$ is 16-byte and $\left\{\mathsf{RK}_i^{\mathsf{dec}}\right\}_{i=0}^{23} \in \{0,1\}^{4608}$ is 576-byte */

for $i = 0$ **to** 3 **do**

  $T[i] = \mathsf{UK}[i]$ // $T = T[0] \parallel \cdots \parallel T[3] \in \{0,1\}^{128=32*4}$

end

for $i = 0$ **to** 23 **do**

  $T[0] \leftarrow \mathtt{RotL}(T[0] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i+0), 1)$ // $T[i] \in \{0,1\}^{32}$

  $T[1] \leftarrow \mathtt{RotL}(T[1] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i+1), 3)$

  $T[2] \leftarrow \mathtt{RotL}(T[2] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i+2), 6)$

  $T[3] \leftarrow \mathtt{RotL}(T[3] \boxplus \mathtt{RotL}(\delta[i \bmod 4], i+3), 11)$

  $\mathsf{RK}_{\mathbf{23-i}}^{\mathsf{dec}} \leftarrow T[0] \parallel T[1] \parallel T[2] \parallel T[1] \parallel T[3] \parallel T[1]$ // $\mathsf{RK}_i^{\mathsf{dec}} \in \{0,1\}^{196=32*6}$

end

return $\left\{\mathsf{RK}_i^{\mathsf{dec}}\right\}_{i=0}^{23}$

1.4 Encryption of LEA-128

Algorithm 4: Encryption of LEA-128

Input: block src = src[0] ‖ src[1] ‖ src[2] ‖ src[3] $\in \{0, 1\}^{128=32*4}$ and $\{\mathsf{RK}_i^{\mathsf{enc}}\}_{i=0}^{N_r-1=23}$

Output: block dst = dsc[0] ‖ dsc[1] ‖ dsc[2] ‖ dsc[3] $\in \{0, 1\}^{128=32*4}$

$t_0 = t[0] \parallel t[1] \parallel t[2] \parallel t[3] \leftarrow$ src

for $i = 0$ **to** 23 **do**

  $tmp \leftarrow t[0]$

  $t_{i+1}[0] \leftarrow \mathtt{RotL}(\, t_i[0] \oplus \mathsf{RK}_i^{\mathsf{enc}}[0] \boxplus (t_i[1] \oplus \mathsf{RK}_i^{\mathsf{enc}}[1])\, , 9)$

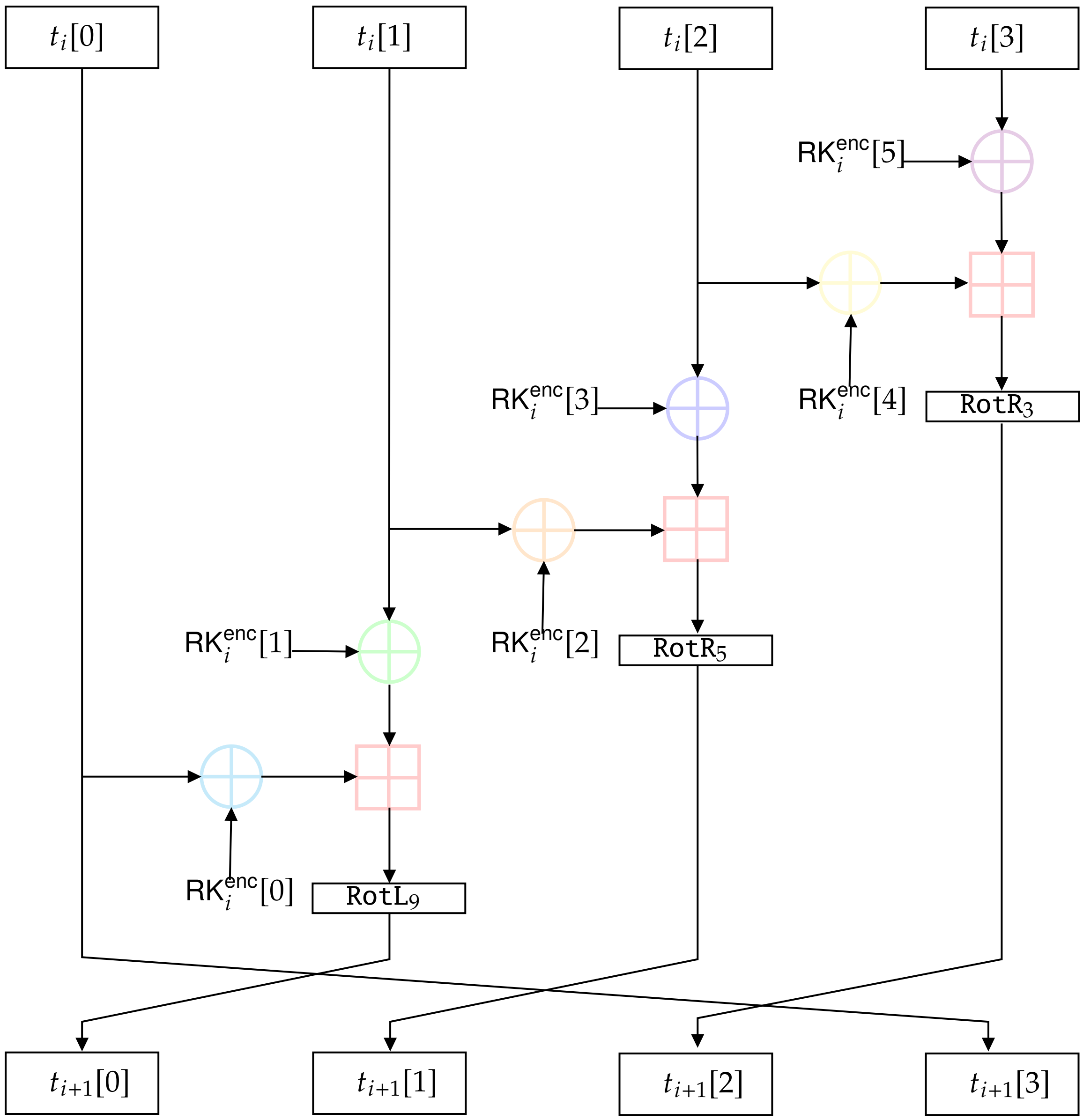
  $t_{i+1}[1] \leftarrow \mathtt{RotR}(\, t_i[1] \oplus \mathsf{RK}_i^{\mathsf{enc}}[2] \boxplus (t_i[2] \oplus \mathsf{RK}_i^{\mathsf{enc}}[3])\, , 5)$

  $t_{i+1}[2] \leftarrow \mathtt{RotR}(\, t_i[2] \oplus \mathsf{RK}_i^{\mathsf{enc}}[4] \boxplus (t_i[3] \oplus \mathsf{RK}_i^{\mathsf{enc}}[5])\, , 3)$

  $t_{i+1}[3] \leftarrow tmp$

end

return dst $\leftarrow t_{N_r}$



1.5 Decryption of LEA-128

Algorithm 5: Encryption of LEA-128

Input: block $\mathsf{src} = \mathsf{src}[0] \,\|\, \mathsf{src}[1] \,\|\, \mathsf{src}[2] \,\|\, \mathsf{src}[3] \in \{0, 1\}^{128=32*4}$ and $\{\mathsf{RK}_i^{\mathsf{enc}}\}_{i=0}^{N_r-1=23}$

Output: block $\mathsf{dst} = \mathsf{dsc}[0] \,\|\, \mathsf{dsc}[1] \,\|\, \mathsf{dsc}[2] \,\|\, \mathsf{dsc}[3] \in \{0, 1\}^{128=32*4}$

$t_0 = t[0] \,\|\, t[1] \,\|\, t[2] \,\|\, t[3] \leftarrow \mathsf{src}$

for $i = 0$ **to** 23 **do**

- $tmp \leftarrow t[0]$
- $t_{i+1}[0] \leftarrow \mathtt{RotL}(t_i[0] \oplus \mathsf{RK}_i^{\mathsf{enc}}[0] \boxplus (t_i[1] \oplus \mathsf{RK}_i^{\mathsf{enc}}[1]), 9)$
- $t_{i+1}[1] \leftarrow \mathtt{RotR}(t_i[1] \oplus \mathsf{RK}_i^{\mathsf{enc}}[2] \boxplus (t_i[2] \oplus \mathsf{RK}_i^{\mathsf{enc}}[3]), 5)$
- $t_{i+1}[2] \leftarrow \mathtt{RotR}(t_i[2] \oplus \mathsf{RK}_i^{\mathsf{enc}}[4] \boxplus (t_i[3] \oplus \mathsf{RK}_i^{\mathsf{enc}}[5]), 3)$
- $t_{i+1}[3] \leftarrow tmp$

end

return $\mathsf{dst} \leftarrow t_{N_r}$

Algorithm 6: Decryption of LEA-128

Input: block $\mathsf{src} \in \{0, 1\}^{128=8*16}$, decryption round-keys $\{\mathsf{RK}_i^{\mathsf{dec}}\}_{i=0}^{N_r-1=23}$

Output: block $\mathsf{dst} \in \{0, 1\}^{128=8*16}$

$t_0 \leftarrow \mathsf{src}$

for $i = 0$ **to** $N_r - 1$ **do**

- $t_{i+1}[0] \leftarrow t_i[3]$
- $t_{i+1}[1] \leftarrow (\mathtt{RotR}(t_i[0], 9) \boxminus (t_{i+1}[0] \oplus \mathsf{RK}_i^{\mathsf{dec}}[0])) \oplus \mathsf{RK}_i^{\mathsf{dec}}[1]$
- $t_{i+1}[2] \leftarrow (\mathtt{RotL}(t_i[1], 5) \boxminus (t_{i+1}[1] \oplus \mathsf{RK}_i^{\mathsf{dec}}[2])) \oplus \mathsf{RK}_i^{\mathsf{dec}}[3]$
- $t_{i+1}[3] \leftarrow (\mathtt{RotL}(t_i[2], 3) \boxminus (t_{i+1}[2] \oplus \mathsf{RK}_i^{\mathsf{dec}}[4])) \oplus \mathsf{RK}_i^{\mathsf{dec}}[5]$

end

return $\mathsf{dst} \leftarrow t_{N_r}$

Chapter 2

Modes of Operation

Table 2.1: Comparison of Modes

Mode	Integrity	Authentication	EncryptBlk	DecryptBlk	Padding	IV	$\lvert P\rvert \stackrel{?}{=} \lvert C\rvert$
ECB	O	X	O	O	O	X	$\lvert P\rvert < \lvert C\rvert$
CBC	O	X	O	O	O	O	$\lvert P\rvert < \lvert C\rvert$
OFB	O	X	O	X	X	O	$\lvert P\rvert = \lvert C\rvert$
CFB	O	X	O	X	X	O	$\lvert P\rvert = \lvert C\rvert$
CTR	O	X	O	X	X	O	$\lvert P\rvert = \lvert C\rvert$
CBC-CS	O	X	O	O	X	O	$\lvert P\rvert = \lvert C\rvert$

2.1 Padding

Block ciphers require input lengths to be a multiple of the block size. Padding is used to extend the last block of plaintext to the required length. Without proper padding, the encryption process may be insecure or infeasible.

There are several padding schemes used in practice, such as:

Table 2.2: Padding Standards in Block Ciphers

Standard Name	Padding Method
PKCS#7	Pad with bytes all the same value as the number of padding bytes `...dd	dd dd dd dd dd dd dd dd dd dd dd dd 04 04 04 04 |`
ANSI X9.23	Pad with zeros, last byte is the number of padding bytes `...dd	dd dd dd dd dd dd dd dd dd dd dd 00 00 00 00 05 |`
ISO/IEC 7816-4	First byte is '80' (hex), followed by zeros `...dd	dd dd dd dd dd dd dd dd dd dd 80 00 00 00 00 00 |`
ISO 10126	Pad with random bytes, last byte is the number of padding bytes `...dd	dd dd dd dd dd dd dd dd dd dd 2e 49 1b c1 aa 06 |`

2.1.1 PKCS#7

```c
void PKCS7_PAD(u32* block, size_t block_len, size_t input_len) {
    if (block_len < input_len) {
        fprintf(stderr,
            "Block length must be greater than input length.\n");
        return;
    }

    u8 padding_value = block_len - input_len;
    for (size_t i = input_len; i < block_len; ++i) {
        block[i] = padding_value;
    }
}
```

2.2 ECB (Electronic CodeBook)

Algorithm 7: Electronic CodeBook

Input: K and $P = P_1 \parallel \cdots \parallel P_N$ ($P_i \in \{\mathtt{0}, \mathtt{1}\}^n$)
Output: $C = C_1 \parallel \cdots \parallel C_N$ ($C_i \in \{\mathtt{0}, \mathtt{1}\}^n$)

for $i \leftarrow 1$ **to** N **do**
  $C_i \leftarrow \mathsf{EncryptBlk}(K, P_i)$;
end
return $C = C_1 \parallel \cdots \parallel C_N$;

Input: K and $C = C_1 \parallel \cdots \parallel C_N$ ($C_i \in \{\mathtt{0}, \mathtt{1}\}^n$)
Output: $P = P_1 \parallel \cdots \parallel P_N$ ($P_i \in \{\mathtt{0}, \mathtt{1}\}^n$)

for $i \leftarrow 1$ **to** N **do**
  $P_i \leftarrow \mathsf{DecryptBlk}(K, C_i)$;
end
return $C = C_1 \parallel \cdots \parallel C_N$;

Appendix A

Additional Data A

A.1 Substitution-BOX