Independent Submission
Request for Comments: 7008
Category: Informational
ISSN: 2070-1721

S. Kiyomoto
W. Shin
KDDI R&D Laboratories, Inc.

August 2013

A Description of the KCipher-2 Encryption Algorithm

Abstract

This document describes the KCipher-2 encryption algorithm. KCipher-2 is a stream cipher with a 128-bit key and a 128-bit initialization vector. Since the algorithm for KCipher-2 was published in 2007, security and efficiency have been rigorously evaluated through academic and industrial studies. As of the publication of this document, no security vulnerabilities have been found. KCipher-2 offers fast encryption and decryption by means of simple operations that enable efficient implementation. KCipher-2 has been used for industrial applications, especially for mobile health monitoring and diagnostic services in Japan.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7008.

Table of Contents

1. Introduction

KCipher-2 is a stream cipher that uses a 128-bit secret key and a 128-bit initialization vector. Since the algorithm for KCipher-2 was published in 2007 [SASC07], it has been evaluated in academic and industrial studies. The security and performance of KCipher-2 have been rigorously evaluated by its developers and other institutions [SECRYPT07] [ICETE07] [CRYPTEC] [SIIS11]. As of the publication of this document, no attack on KCipher-2 has been successful. KCipher-2 can be efficiently implemented in software to provide fast encryption and decryption, owing to its uncomplicated design. Only four simple operations are used: exclusive-OR, addition, shift, and table lookup. When the algorithm is implemented in hardware, internal computations can be parallel to yield greater efficiency. Moreover, since its internal state representation only amounts to several hundred bits, KCipher-2 is suitable for resource-limited environments. KCipher-2 has been actively used in several industrial applications in Japan, has been published by an international standardization body (ISO/IEC 18033-4 [ISO18033]), and has been designated a Japanese e-Government recommended cipher [CRYPTECLIST].

2. Algorithm Description

In this section, we describe the internal components of KCipher-2 and define the operations for deriving key streams from an input key and an initialization vector. We illustrate the detailed operations, mostly in pseudocode format, but also provide code snippets written in the C language syntax when necessary.

2.1. Notations

All values in this document are stored in big-endian order (aka network byte order). We use the following notations in the description of KCipher-2.

^	Bitwise exclusive-OR
n#m	mth power of n
+n	Integer addition modulo 2#n
<<_r n	n-bit left circular shift in an r-bit register
0x	Hexadecimal representation
E[i]	The (i + 1)th element of E when E is composed of consecutive multiple elements
GF	Galois field. GF(n#m) means the finite field of exactly n#m elements
**	Multiplication of elements on the finite field GF(2#32)

NOTE: Many texts denote "the mth power of n" by "n^m", but we write it using '#', instead of '^', to avoid reader confusion with the power operator and the XOR operator of the C language syntax.

2.2. Internal State

The internal state of KCipher-2 can be denoted by S. The internal state consists of six sub-components: two feedback shift registers, FSR-A and FSR-B, and four internal registers, L1, R1, L2, and R2. We, therefore, often write S = (A, B, L1, R1, L2, R2), where A and B refer to FSR-A and FSR-B, respectively.

2.2.1. Feedback Shift Registers

The two feedback shift registers (FSRs) are separately called Feedback Shift Register A (FSR-A) and Feedback Shift Register B (FSR-B). FSR-A is composed of five 32-bit units that are consecutively arranged. Each unit can be identified by A[0], A[1], A[2], A[3], and A[4]. Likewise, FSR-B is composed of eleven consecutive 32-bit units, B[0], ..., B[10]. All values stored in each 32-bit unit of FSR is in GF(2#32).

2.2.2. Internal Registers

Besides FSR, KCipher-2 has four internal registers to store intermediate computation results during operation. The four registers are named L1, R1, L2, and R2.

2.3. Operations

Three major operations constitute the behavior of KCipher-2: init(), next(), and stream(). The init() operation initializes the internal values of the system. The next() operation derives new values of S' from the values of S, where S' and S refer to the internal state. The stream() operation derives a key stream from the current state S.

2.3.1. next()

The next() operation takes the current state S = (A, B, L1, R1, L2, R2) as input. The size of the input amounts to twenty of the 32-bit units in total (five units for A, eleven for B, and one for L1, R1, L2, and R2). It produces the next state S' = (A', B', L1', R1', L2', R2'). This operation is mainly used to generate secure key streams by applying non-linear functions (NLFs) for every cycle of KCipher-2. Additionally, it is used to initialize the system. The behaviors are distinguished by the input parameter that indicates the operation modes.

Inside the next() operation, the internal registers are updated by the result of the substitution function described in Section 2.4.2. The feedback shift registers are also updated by feedback functions. The feedback functions include the multiplication of register units and the fixed elements a0, a1, a2, and a3 in a finite field. The fixed elements a0, ..., a3 are carefully chosen to provide the maximum length of the feedback shift registers. The theory behind the selection of fixed elements and the way to simplify the necessary multiplications are briefly described in Section 2.4.4.

The operation takes the following inputs:

- S = (A, B, L1, R1, L2, R2)
- mode = {INIT, NORMAL}, where INIT means the operation is used for initialization, and NORMAL means it is used for generating secure key streams.

The operation outputs a new state,

- S' = (A', B', L1', R1', L2', R2')

by performing the following steps:

1. Set registers in the nonlinear functions

```
L1' = sub_K2(R2 +32 B[4]);
R1' = sub_K2(L2 +32 B[9]);
L2' = sub_K2(L1);
R2' = sub_K2(R1);

for m from 0 to 3
   A'[m] = A[m + 1];

for m from 0 to 9
   B'[m] = B[m + 1];
```

   NOTE: sub_K2 is a substitution function described in Section 2.4.2.

2. Depending on the value of the operation mode, do the following:

   a. When the mode is NORMAL, A'[4] and B'[10] are computed as follows:

```
A'[4] = (a0 ** A[0]) ^ A[3];

if A[2][30] is 1:
  if A[2][31] is 1:
    B'[10] = (a1 ** B[0]) ^ B[1] ^ B[6] ^ (a3 ** B[8]);
  else if A[2][31] is 0:
    B'[10] = (a1 ** B[0]) ^ B[1] ^ B[6] ^ B[8];
else if A[2][30] is 0:
  if A[2][31] is 1:
    B'[10] = (a2 ** B[0]) ^ B[1] ^ B[6] ^ (a3 ** B[8]);
  else if A[2][31] is 0:
    B'[10] = (a2 ** B[0]) ^  B[1] ^ B[6] ^ B[8];
```

   b. When the mode is INIT, A'[4] and B'[10] are XOR-ed with the non-linear function output described in Section 2.4.1.

```
A'[4] = (a0 ** A[0]) ^ A[3] ^ NLF(B[0], R2, R1, A[4]);

if A[2][30] is 1:
  if A[2][31] is 1:
    B'[10] = (a1 ** B[0]) ^ B[1] ^ B[6] ^ (a3 ** B[8]) ^
           NLF(B[10], L2, L1, A[0]);
  else if A[2][31] is 0:
    B'[10] = (a1 ** B[0]) ^ B[1] ^ B[6] ^ B[8] ^
           NLF(B[10], L2, L1, A[0]);
```

```
            else if A[2][30] is 0:
              if A[2][31] is 1:
                B'[10] = (a2 ** B[0]) ^ B[1] ^ B[6] ^ (a3 ** B[8]) ^
                       NLF(B[10], L2, L1, A[0]);
              else if A[2][31] is 0:
                B'[10] = (a2 ** B[0]) ^ B[1] ^ B[6] ^ B[8] ^
                       NLF(B[10], L2, L1, A[0]);
```

3. Output S' = (A', B', L1', R1', L2', R2').

Note that A[2] is a 32-bit unit. Thus, A[2][j] is the value of the jth least significant bit of A[2], where 0 <= j <= 31.

The corresponding code snippets written in the C language syntax can be found in Section 2.4.4 and Appendix B.

2.3.2. init()

The init() operation takes a 128-bit key (K) and a 128-bit initialization vector (IV) and prepares the values of the state variables for generating key streams.

- K = (K[0], K[1], K[2], K[3]), where each K[i] is a 32-bit unit and 0 <= i <= 3

- IV =(IV[0], IV[1], IV[2], IV[3]), where each IV[i] is a 32-bit unit and 0 <= i <= 3,

and the output is an initialized state S, which will be referenced as S(0). The output is derived from the following steps:

1. Expand K to the 384-bit internal key IK = (IK[0], ..., IK[11]), where IK[i] is a 32-bit unit and 0 <= i <= 11. The expansion procedure is as follows:

```
      for m from 0 to 11
         if m is 0, 1, 2, or 3:
            IK[m] = K[m];
         else if m is 5, 6, 7, 9, 10, or 11:
            IK[m] = IK[m - 4] ^ IK[m - 1];
         else if m is 4:
            IK[4] = IK[0] ^ sub_K2(IK[3] <<_32 8) ^
            (0x01, 0x00, 0x00, 0x00);
         else if m is 8:
            IK[8] = IK[4] ^ sub_K2(IK[7] <<_32 8) ^
            (0x02, 0x00, 0x00, 0x00);
```

NOTE: sub_K2 is the substitution function described in Section 2.4.2.

2. Initialize the feedback shift registers and the internal registers using the values of IK and IV as follows:

```
for m from 0 to 4
   A[m] = IK[4 - m];

B[0] = IK[10]; B[1] = IK[11]; B[2] = IV[0];  B[3] = IV[1];
B[4] = IK[8];  B[5] = IK[9];  B[6] = IV[2];  B[7] = IV[3];
B[8] = IK[7];  B[9] = IK[5];  B[10] = IK[6];

L1 = R1 = L2 = R2 = 0x00000000;

Set S as (A, B, L1, R1, L2, R2).
```

3. Prepare the state values by applying the next() operation 24 times as follows:

```
 for m from 1 to 24
   Set S' as next(S, INIT);
   Set S as S';
```

4. Output S.

2.3.3. stream()

The stream() function derives a 64-bit key stream, Z, from the state values. Its input is an initialized state,

o S = (A, B, L1, R1, L2, R2)

and its output is Z = (ZH, ZL), where ZH and ZL are 32-bit units. stream() performs the following:

1. Set register values

```
ZH = NLF(B[10], L2, L1, A[0]);
ZL = NLF(B[0], R2, R1, A[4]);
```

2. Output Z = (ZH, ZL).

NOTE: The function NLF is described in Section 2.4.1.

2.4. Subroutines

We explain the functions used above: sub_K2(), NLF(), and S_box().

2.4.1. NLF()

NLF() is a non-linear function that takes the four 32-bit values, A, B, C, D, and outputs the 32-bit value, Q. The output Q is calculated as follows.

```
Q = (A +32 B) ^ C ^ D;
```

2.4.2. sub_K2()

sub_K2() is a substitution function that is a permutation of GF(2#32), based on components from the Advanced Encryption Standard (AES) [FIPS-AES]. Its input is a 32-bit value divided into four 8-bit strings. Inside sub_K2(), an 8-to-8-bit substitution function, S_box(), is applied to each 8-bit string separately, and then a 32-to-32-bit linear permutation is applied to the whole 32-bit string. Our S_box() function is identical to the S-Box operation of AES, and our linear permutation is identical to the AES Mix Column operation.

Consider the input of sub_K2 as a 32-bit value W = (w[3], w[2], w[1], w[0]), where each subelement of w is an 8-bit unit. Prepare two 32-bit temporary storages, T = (t[3], t[2], t[1], t[0]) and Q = (q[3], q[2], q[1], q[0]), where t[i] and q[i] are 8-bit units and 0 <= i <= 3.

The 32-bit output Q is obtained from the following procedures:

1. Apply S_box() to each 8-bit input string. Note that S_box() is defined in Section 2.4.3.

   ```
   for m from 0 to 3
      t[m] = S_box(w[m]);
   ```

2. Calculate q by the matrix multiplication, Q = M * T in GF(2#8) of the irreducible polynomial f(x) = x#8 + x#4 + x#3 + x + 1, where

   - Q is a 1x4 matrix, (q[0], q[1], q[2], q[3])
   - M is a 4x4 matrix,

   ```
   (02,  03,  01,  01,
    01,  02,  03,  01,
    01,  01,  02,  03,
    03,  01,  01,  02)
   ```

o T is a 1x4 matrix, (t[0], t[1], t[2], t[3]).

Namely, the procedure that calculates (q[3], q[2], q[1], q[0]) can be written in the C language syntax as:

```
q[0] = GF_mult_by_2(t[0]) ^ GF_mult_by_3(t[1]) ^ t[2] ^ t[3];
q[1] = t[0] ^ GF_mult_by_2(t[1]) ^ GF_mult_by_3(t[2]) ^ t[3];
q[2] = t[0] ^ t[1] ^ GF_mult_by_2(t[2]) ^ GF_mult_by_3(t[3]);
q[3] = GF_mult_by_3(t[0]) ^ t[1] ^ t[2] ^ GF_mult_by_2(t[3]);
```

where GF_mult_by_2 and GF_mult_by_3 are multiplication functions in GF(2#8), defined as follows:

o The function GF_mult_by_2(t) multiplies 2 by the given 8-bit value t in GF(2#8) and returns an 8-bit value q as follows (lq is a temporary 32-bit variable):

```
lq = t << 1;
if ((lq & 0x100) != 0) lq ^= 0x011B;
q = lq ^ 0xFF;
```

o The function GF_mult_by_3(t) multiplies 3 by the given 8-bit value t in GF(2#8) and returns an 8-bit value q as follows (lq is a temporary 32-bit variable):

```
lq = (t << 1) ^ t;
if ((lq & 0x100) != 0) lq ^= 0x011B;
q = lq ^ 0xFF;
```

3. Output Q = (q[3], q[2], q[1], q[0]).

2.4.3. S_box()

S_box() is a substitution that can be done by a simple table lookup operation. Thus, S_box() can be defined by the following value table:

```
S_box[256] = {
   0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
   0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
   0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
   0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
   0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
   0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
   0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
   0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
   0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
   0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
```

```
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };
```

2.4.4. Multiplications in GF(2#32)

FSR-A and FSR-B are word-oriented linear feedback shift registers (LFSRs). In the next() operation of Section 2.3.1, the feedback functions to the two LFSRs are shown, which include multiplication of fixed elements a0, a1, a2, or a3 in GF(2#32). The fixed elements are carefully chosen to maximize the period of the key stream generated by the two registers. Here, we briefly explain how we obtain the fixed elements. Further details and theories can be found in [SECRYPT07].

We obtain a0 as follows. First, to guarantee that the period is maximized for an 8-bit unit, we consider p as the root of the primitive polynomial:

x#8 + x#7 + x#6, + x + 1 in GF(2).

Therefore, an 8-bit string y = (y7, ..., y0), where y7 is the most significant bit, can be written as:

y = y7(p#7) + y6(p#6) + ... + y1(p) + y0

Next, a0 is the root of irreducible polynomial of degree four:

x#4 + p#24(x#3) + p#3(x#2) + p#12(x) + p#71 in GF(2#8).

Then, hierarchically, a 32-bit unit Y = (Y3, Y2, Y1, Y0), where Y3 is the most significant byte, can be written as:

```
Y3(a0#3) + Y2(a0#2) + Y1(a0) + Y0
```

The feedback polynomial to FSR-A,

```
f(x) = a0(x#5) + x#2 + 1
```

produces the maximum-length period of the key stream with a0.

Similarly, a1, a2, and a3 are the roots of irreducible polynomials of degree four of

```
x#4 + q#230(x#3) + q#156(x#2) + q#93(x) + q#29 in GF(2#8)
x#4 + r#34(x#3) + r#16(x#2) + r#199(x) + r#248 in GF(2#8)
x#4 + s#157(x#3) + s#253(x#2) + s#56(x) + s#16 in GF(2#8)
```

respectively. The feedback polynomial to FSR-B that uses a1, a2, and a3 can produce the maximum-length period. The feedback polynomials to FSR-A and FSR-B are as written in Step 2 of the next() operation, and the mathematical notations of these polynomials can also be found in [SECRYPT07].

Calculation of the original feedback polynomials might be time-consuming because it includes multiplications in finite fields. However, these multiplications can be done faster if the multiples of a0, ..., a3 were already calculated for all possible inputs. The tables of amul0, ..., amul3 in Appendix A provide such pre-calculation results. As shown in Step 2 of next(), we can utilize these tables to finish the necessary calculations efficiently.

For example, consider the input as a 32-bit value w, which represents an element of GF(2#32). The 32-bit output string w' = a0 ** w can be obtained using the amul0 table in Appendix A.1 as follows:

```
w' = (w << 8) ^ amul0[w >> 24];
```

Likewise, multiplications of (a1 ** w), (a2 ** w), and (a3 ** w) can be obtained in the same way, simply by using the amul1, amul2, and amul3 tables that we provide in Appendixes A.2, A.3, and A.4.

Eventually, Step 2 of the next() operation, which updates A'[4] and B'[10], can be written in the C language syntax as follows. Note that nA[4] and nB[10] correspond to A'[4] and B'[10], respectively, and temp1 and temp2 are 32-bit variables.

```
nA[4] = ((A[0] << 8) ^ amul0[(A[0] >> 24)]) ^ A[3];
if (mode == INIT)
  nA[4] ^= NLF(B[0], R2, R1, A[4]);

if (A[2] & 0x40000000) {
  temp1 = (B[0] << 8) ^ amul1[(B[0] >> 24)];
} else {
  temp1 = (B[0] << 8) ^ amul2[(B[0] >> 24)];
}

if (A[2] & 0x80000000) {
  temp2 = (B[8] << 8) ^ amul3[(B[8] >> 24)];
} else {
  temp2 = B[8];
}

nB[10] = temp1 ^ B[1] ^ B[6] ^ temp2;
if (mode == INIT)
  nB[10] ^= NLF(B[10], L2, L1, A[0]);
```

2.5. Encryption and Decryption Scheme

In this section, we use the notation S(i) to specifically reference the values of the internal state at i (where i >= 0), which is an arbitrary, discrete temporal moment (aka cycle) after the initialization.

2.5.1. Key Stream Generation

Given a 128-bit key K, a 128-bit initialization vector (IV), KCipher-2 is initialized as follows:

```
S(0) = init(K, IV);
```

where S(0) is a state representation. With an initialized state S(i), where i >= 0, a 64-bit key stream X(i) can be obtained using the stream() operation, as follows:

```
X(i) = stream(S(i));
```

To generate a new key stream X(i + 1), use the next() operation and the stream() operation as follows:

```
S(i + 1) = next(S(i), NORMAL);
X(i + 1) = stream(S(i + 1));
```

2.5.2. Encryption and Decryption of a Message

Given a 64-bit message block M and a key stream X, an encrypted message E is obtained by

```
E = M ^ X;
```

Conversely, the decrypted message D is obtained by

```
D = E ^ X;
```

The original message M and the decrypted message D are identical when the same key stream is used.

3. Security Considerations

We recommend reinitializing and rekeying after 2#58 cycles of KCipher-2, which means after generating 2#64 key stream bits. It is important to make sure that no IV is ever reused under the same key.

4. References

4.1. Normative References

[ISO18033] "Information technology -- Security techniques -- Encryption algorithms -- Part 4: Stream ciphers", ISO/IEC 18033-4:2012 Ed. 2, December 2012.

[FIPS-AES] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

4.2. Informative References

[SECRYPT07] Kiyomoto, S., Tanaka, T., and K. Sakurai, "K2: A Stream Cipher Algorithm Using Dynamic Feedback Control", Proc. SECRYPT 2007, pp. 204-213.

[ICETE07] Kiyomoto, S., Tanaka, T., and K. Sakurai, "K2 Stream Cipher", Proc. ICETE 2007, pp. 214-226.

[CRYPTEC] Bogdanov, A., Preneel, B., and V. Rijmen, "Security Evaluation of the K2 Stream Cipher", 2010, <http://www.cryptrec.go.jp/english/estimation.html>.

[CRYPTECLIST] "Cryptography Research and Evaluation Committees", <http://www.cryptrec.go.jp/english/estimation.html>.

[SIIS11] Priemuth-Schmid, D., "Attacks on Simplified Versions of K2", Proc. SIIS 2011, LNCS 7053, pp. 117-127.

[SASC07] Kiyomoto, S., Tanaka, T., and K. Sakurai, "A Word-Oriented Stream Cipher Using Clock Control", Proc. SASC 2007, pp. 260-274.

Appendix A. Tables for Multiplication in GF(2#32)

A.1. The table amul0

```
amul0[256] = {
   0x00000000,0xB6086D1A,0xAF10DA34,0x1918B72E,
   0x9D207768,0x2B281A72,0x3230AD5C,0x8438C046,
   0xF940EED0,0x4F4883CA,0x565034E4,0xE05859FE,
   0x646099B8,0xD268F4A2,0xCB70438C,0x7D782E96,
   0x31801F63,0x87887279,0x9E90C557,0x2898A84D,
   0xACA0680B,0x1AA80511,0x03B0B23F,0xB5B8DF25,
   0xC8C0F1B3,0x7EC89CA9,0x67D02B87,0xD1D8469D,
   0x55E086DB,0xE3E8EBC1,0xFAF05CEF,0x4CF831F5,
   0x62C33EC6,0xD4CB53DC,0xCDD3E4F2,0x7BDB89E8,
   0xFFE349AE,0x49EB24B4,0x50F3939A,0xE6FBFE80,
   0x9B83D016,0x2D8BBD0C,0x34930A22,0x829B6738,
   0x06A3A77E,0xB0ABCA64,0xA9B37D4A,0x1FBB1050,
   0x534321A5,0xE54B4CBF,0xFC53FB91,0x4A5B968B,
   0xCE6356CD,0x786B3BD7,0x61738CF9,0xD77BE1E3,
   0xAA03CF75,0x1C0BA26F,0x05131541,0xB31B785B,
   0x3723B81D,0x812BD507,0x98336229,0x2E3B0F33,
   0xC4457C4F,0x724D1155,0x6B55A67B,0xDD5DCB61,
   0x59650B27,0xEF6D663D,0xF675D113,0x407DBC09,
   0x3D05929F,0x8B0DFF85,0x921548AB,0x241D25B1,
   0xA025E5F7,0x162D88ED,0x0F353FC3,0xB93D52D9,
   0xF5C5632C,0x43CD0E36,0x5AD5B918,0xECDDD402,
   0x68E51444,0xDEED795E,0xC7F5CE70,0x71FDA36A,
   0x0C858DFC,0xBA8DE0E6,0xA39557C8,0x159D3AD2,
   0x91A5FA94,0x27AD978E,0x3EB520A0,0x88BD4DBA,
   0xA6864289,0x108E2F93,0x099698BD,0xBF9EF5A7,
   0x3BA635E1,0x8DAE58FB,0x94B6EFD5,0x22BE82CF,
   0x5FC6AC59,0xE9CEC143,0xF0D6766D,0x46DE1B77,
   0xC2E6DB31,0x74EEB62B,0x6DF60105,0xDBFE6C1F,
   0x97065DEA,0x210E30F0,0x381687DE,0x8E1EEAC4,
   0x0A262A82,0xBC2E4798,0xA536F0B6,0x133E9DAC,
   0x6E46B33A,0xD84EDE20,0xC156690E,0x775E0414,
   0xF366C452,0x456EA948,0x5C761E66,0xEA7E737C,
   0x4B8AF89E,0xFD829584,0xE49A22AA,0x52924FB0,
   0xD6AA8FF6,0x60A2E2EC,0x79BA55C2,0xCFB238D8,
   0xB2CA164E,0x04C27B54,0x1DDACC7A,0xABD2A160,
   0x2FEA6126,0x99E20C3C,0x80FABB12,0x36F2D608,
   0x7A0AE7FD,0xCC028AE7,0xD51A3DC9,0x631250D3,
   0xE72A9095,0x5122FD8F,0x483A4AA1,0xFE3227BB,
   0x834A092D,0x35426437,0x2C5AD319,0x9A52BE03,
   0x1E6A7E45,0xA862135F,0xB17AA471,0x0772C96B,
   0x2949C658,0x9F41AB42,0x86591C6C,0x30517176,
   0xB469B130,0x0261DC2A,0x1B796B04,0xAD71061E,
   0xD0092888,0x66014592,0x7F19F2BC,0xC9119FA6,
```

```
        0x4D295FE0,0xFB2132FA,0xE23985D4,0x5431E8CE,
        0x18C9D93B,0xAEC1B421,0xB7D9030F,0x01D16E15,
        0x85E9AE53,0x33E1C349,0x2AF97467,0x9CF1197D,
        0xE18937EB,0x57815AF1,0x4E99EDDF,0xF89180C5,
        0x7CA94083,0xCAA12D99,0xD3B99AB7,0x65B1F7AD,
        0x8FCF84D1,0x39C7E9CB,0x20DF5EE5,0x96D733FF,
        0x12EFF3B9,0xA4E79EA3,0xBDFF298D,0x0BF74497,
        0x768F6A01,0xC087071B,0xD99FB035,0x6F97DD2F,
        0xEBAF1D69,0x5DA77073,0x44BFC75D,0xF2B7AA47,
        0xBE4F9BB2,0x0847F6A8,0x115F4186,0xA7572C9C,
        0x236FECDA,0x956781C0,0x8C7F36EE,0x3A775BF4,
        0x470F7562,0xF1071878,0xE81FAF56,0x5E17C24C,
        0xDA2F020A,0x6C276F10,0x753FD83E,0xC337B524,
        0xED0CBA17,0x5B04D70D,0x421C6023,0xF4140D39,
        0x702CCD7F,0xC624A065,0xDF3C174B,0x69347A51,
        0x144C54C7,0xA24439DD,0xBB5C8EF3,0x0D54E3E9,
        0x896C23AF,0x3F644EB5,0x267CF99B,0x90749481,
        0xDC8CA574,0x6A84C86E,0x739C7F40,0xC594125A,
        0x41ACD21C,0xF7A4BF06,0xEEBC0828,0x58B46532,
        0x25CC4BA4,0x93C426BE,0x8ADC9190,0x3CD4FC8A,
        0xB8EC3CCC,0x0EE451D6,0x17FCE6F8,0xA1F48BE2 };
```

A.2. The table amul1

```
      amul1[256] = {
         0x00000000,0xA0F5FC2E,0x6DC7D55C,0xCD322972,
         0xDAA387B8,0x7A567B96,0xB76452E4,0x1791AECA,
         0x996B235D,0x399EDF73,0xF4ACF601,0x54590A2F,
         0x43C8A4E5,0xE33D58CB,0x2E0F71B9,0x8EFA8D97,
         0x1FD646BA,0xBF23BA94,0x721193E6,0xD2E46FC8,
         0xC575C102,0x65803D2C,0xA8B2145E,0x0847E870,
         0x86BD65E7,0x264899C9,0xEB7AB0BB,0x4B8F4C95,
         0x5C1EE25F,0xFCEB1E71,0x31D93703,0x912CCB2D,
         0x3E818C59,0x9E747077,0x53465905,0xF3B3A52B,
         0xE4220BE1,0x44D7F7CF,0x89E5DEBD,0x29102293,
         0xA7EAAF04,0x071F532A,0xCA2D7A58,0x6AD88676,
         0x7D4928BC,0xDDBCD492,0x108EFDE0,0xB07B01CE,
         0x2157CAE3,0x81A236CD,0x4C901FBF,0xEC65E391,
         0xFBF44D5B,0x5B01B175,0x96339807,0x36C66429,
         0xB83CE9BE,0x18C91590,0xD5FB3CE2,0x750EC0CC,
         0x629F6E06,0xC26A9228,0x0F58BB5A,0xAFAD4774,
         0x7C2F35B2,0xDCDAC99C,0x11E8E0EE,0xB11D1CC0,
         0xA68CB20A,0x06794E24,0xCB4B6756,0x6BBE9B78,
         0xE54416EF,0x45B1EAC1,0x8883C3B3,0x28763F9D,
         0x3FE79157,0x9F126D79,0x5220440B,0xF2D5B825,
         0x63F97308,0xC30C8F26,0x0E3EA654,0xAECB5A7A,
         0xB95AF4B0,0x19AF089E,0xD49D21EC,0x7468DDC2,
         0xFA925055,0x5A67AC7B,0x97558509,0x37A07927,
```

```
        0x2031D7ED,0x80C42BC3,0x4DF602B1,0xED03FE9F,
        0x42AEB9EB,0xE25B45C5,0x2F696CB7,0x8F9C9099,
        0x980D3E53,0x38F8C27D,0xF5CAEB0F,0x553F1721,
        0xDBC59AB6,0x7B306698,0xB6024FEA,0x16F7B3C4,
        0x01661D0E,0xA193E120,0x6CA1C852,0xCC54347C,
        0x5D78FF51,0xFD8D037F,0x30BF2A0D,0x904AD623,
        0x87DB78E9,0x272E84C7,0xEA1CADB5,0x4AE9519B,
        0xC413DC0C,0x64E62022,0xA9D40950,0x0921F57E,
        0x1EB05BB4,0xBE45A79A,0x73778EE8,0xD38272C6,
        0xF85E6A49,0x58AB9667,0x9599BF15,0x356C433B,
        0x22FDEDF1,0x820811DF,0x4F3A38AD,0xEFCFC483,
        0x61354914,0xC1C0B53A,0x0CF29C48,0xAC076066,
        0xBB96CEAC,0x1B633282,0xD6511BF0,0x76A4E7DE,
        0xE7882CF3,0x477DD0DD,0x8A4FF9AF,0x2ABA0581,
        0x3D2BAB4B,0x9DDE5765,0x50EC7E17,0xF0198239,
        0x7EE30FAE,0xDE16F380,0x1324DAF2,0xB3D126DC,
        0xA4408816,0x04B57438,0xC9875D4A,0x6972A164,
        0xC6DFE610,0x662A1A3E,0xAB18334C,0x0BEDCF62,
        0x1C7C61A8,0xBC899D86,0x71BBB4F4,0xD14E48DA,
        0x5FB4C54D,0xFF413963,0x32731011,0x9286EC3F,
        0x851742F5,0x25E2BEDB,0xE8D097A9,0x48256B87,
        0xD909A0AA,0x79FC5C84,0xB4CE75F6,0x143B89D8,
        0x03AA2712,0xA35FDB3C,0x6E6DF24E,0xCE980E60,
        0x406283F7,0xE0977FD9,0x2DA556AB,0x8D50AA85,
        0x9AC1044F,0x3A34F861,0xF706D113,0x57F32D3D,
        0x84715FFB,0x2484A3D5,0xE9B68AA7,0x49437689,
        0x5ED2D843,0xFE27246D,0x33150D1F,0x93E0F131,
        0x1D1A7CA6,0xBDEF8088,0x70DDA9FA,0xD02855D4,
        0xC7B9FB1E,0x674C0730,0xAA7E2E42,0x0A8BD26C,
        0x9BA71941,0x3B52E56F,0xF660CC1D,0x56953033,
        0x41049EF9,0xE1F162D7,0x2CC34BA5,0x8C36B78B,
        0x02CC3A1C,0xA239C632,0x6F0BEF40,0xCFFE136E,
        0xD86FBDA4,0x789A418A,0xB5A868F8,0x155D94D6,
        0xBAF0D3A2,0x1A052F8C,0xD73706FE,0x77C2FAD0,
        0x6053541A,0xC0A6A834,0x0D948146,0xAD617D68,
        0x239BF0FF,0x836E0CD1,0x4E5C25A3,0xEEA9D98D,
        0xF9387747,0x59CD8B69,0x94FFA21B,0x340A5E35,
        0xA5269518,0x05D36936,0xC8E14044,0x6814BC6A,
        0x7F8512A0,0xDF70EE8E,0x1242C7FC,0xB2B73BD2,
        0x3C4DB645,0x9CB84A6B,0x518A6319,0xF17F9F37,
        0xE6EE31FD,0x461BCDD3,0x8B29E4A1,0x2BDC188F };
```

A.3. The table amul2

```
amul2[256] = {
   0x00000000,0x5BF87F93,0xB6BDFE6B,0xED4581F8,
   0x2137B1D6,0x7ACFCE45,0x978A4FBD,0xCC72302E,
   0x426E2FE1,0x19965072,0xF4D3D18A,0xAF2BAE19,
   0x63599E37,0x38A1E1A4,0xD5E4605C,0x8E1C1FCF,
   0x84DC5E8F,0xDF24211C,0x3261A0E4,0x6999DF77,
   0xA5EBEF59,0xFE1390CA,0x13561132,0x48AE6EA1,
   0xC6B2716E,0x9D4A0EFD,0x700F8F05,0x2BF7F096,
   0xE785C0B8,0xBC7DBF2B,0x51383ED3,0x0AC04140,
   0x45F5BC53,0x1E0DC3C0,0xF3484238,0xA8B03DAB,
   0x64C20D85,0x3F3A7216,0xD27FF3EE,0x89878C7D,
   0x079B93B2,0x5C63EC21,0xB1266DD9,0xEADE124A,
   0x26AC2264,0x7D545DF7,0x9011DC0F,0xCBE9A39C,
   0xC129E2DC,0x9AD19D4F,0x77941CB7,0x2C6C6324,
   0xE01E530A,0xBBE62C99,0x56A3AD61,0x0D5BD2F2,
   0x8347CD3D,0xD8BFB2AE,0x35FA3356,0x6E024CC5,
   0xA2707CEB,0xF9880378,0x14CD8280,0x4F35FD13,
   0x8AA735A6,0xD15F4A35,0x3C1ACBCD,0x67E2B45E,
   0xAB908470,0xF068FBE3,0x1D2D7A1B,0x46D50588,
   0xC8C91A47,0x933165D4,0x7E74E42C,0x258C9BBF,
   0xE9FEAB91,0xB206D402,0x5F4355FA,0x04BB2A69,
   0x0E7B6B29,0x558314BA,0xB8C69542,0xE33EEAD1,
   0x2F4CDAFF,0x74B4A56C,0x99F12494,0xC2095B07,
   0x4C1544C8,0x17ED3B5B,0xFAA8BAA3,0xA150C530,
   0x6D22F51E,0x36DA8A8D,0xDB9F0B75,0x806774E6,
   0xCF5289F5,0x94AAF666,0x79EF779E,0x2217080D,
   0xEE653823,0xB59D47B0,0x58D8C648,0x0320B9DB,
   0x8D3CA614,0xD6C4D987,0x3B81587F,0x607927EC,
   0xAC0B17C2,0xF7F36851,0x1AB6E9A9,0x414E963A,
   0x4B8ED77A,0x1076A8E9,0xFD332911,0xA6CB5682,
   0x6AB966AC,0x3141193F,0xDC0498C7,0x87FCE754,
   0x09E0F89B,0x52188708,0xBF5D06F0,0xE4A57963,
   0x28D7494D,0x732F36DE,0x9E6AB726,0xC592C8B5,
   0x59036A01,0x02FB1592,0xEFBE946A,0xB446EBF9,
   0x7834DBD7,0x23CCA444,0xCE8925BC,0x95715A2F,
   0x1B6D45E0,0x40953A73,0xADD0BB8B,0xF628C418,
   0x3A5AF436,0x61A28BA5,0x8CE70A5D,0xD71F75CE,
   0xDDDF348E,0x86274B1D,0x6B62CAE5,0x309AB576,
   0xFCE88558,0xA710FACB,0x4A557B33,0x11AD04A0,
   0x9FB11B6F,0xC44964FC,0x290CE504,0x72F49A97,
   0xBE86AAB9,0xE57ED52A,0x083B54D2,0x53C32B41,
   0x1CF6D652,0x470EA9C1,0xAA4B2839,0xF1B357AA,
   0x3DC16784,0x66391817,0x8B7C99EF,0xD084E67C,
   0x5E98F9B3,0x05608620,0xE82507D8,0xB3DD784B,
   0x7FAF4865,0x245737F6,0xC912B60E,0x92EAC99D,
   0x982A88DD,0xC3D2F74E,0x2E9776B6,0x756F0925,
```

```
        0xB91D390B,0xE2E54698,0x0FA0C760,0x5458B8F3,
        0xDA44A73C,0x81BCD8AF,0x6CF95957,0x370126C4,
        0xFB7316EA,0xA08B6979,0x4DCEE881,0x16369712,
        0xD3A45FA7,0x885C2034,0x6519A1CC,0x3EE1DE5F,
        0xF293EE71,0xA96B91E2,0x442E101A,0x1FD66F89,
        0x91CA7046,0xCA320FD5,0x27778E2D,0x7C8FF1BE,
        0xB0FDC190,0xEB05BE03,0x06403FFB,0x5DB84068,
        0x57780128,0x0C807EBB,0xE1C5FF43,0xBA3D80D0,
        0x764FB0FE,0x2DB7CF6D,0xC0F24E95,0x9B0A3106,
        0x15162EC9,0x4EEE515A,0xA3ABD0A2,0xF853AF31,
        0x34219F1F,0x6FD9E08C,0x829C6174,0xD9641EE7,
        0x9651E3F4,0xCDA99C67,0x20EC1D9F,0x7B14620C,
        0xB7665222,0xEC9E2DB1,0x01DBAC49,0x5A23D3DA,
        0xD43FCC15,0x8FC7B386,0x6282327E,0x397A4DED,
        0xF5087DC3,0xAEF00250,0x43B583A8,0x184DFC3B,
        0x128DBD7B,0x4975C2E8,0xA4304310,0xFFC83C83,
        0x33BA0CAD,0x6842733E,0x8507F2C6,0xDEFF8D55,
        0x50E3929A,0x0B1BED09,0xE65E6CF1,0xBDA61362,
        0x71D4234C,0x2A2C5CDF,0xC769DD27,0x9C91A2B4 };
```

A.4. The table amul3

```
      amul3[256] = {
         0x00000000,0x4559568B,0x8AB2AC73,0xCFEBFAF8,
         0x71013DE6,0x34586B6D,0xFBB39195,0xBEEAC71E,
         0xE2027AA9,0xA75B2C22,0x68B0D6DA,0x2DE98051,
         0x9303474F,0xD65A11C4,0x19B1EB3C,0x5CE8BDB7,
         0xA104F437,0xE45DA2BC,0x2BB65844,0x6EEF0ECF,
         0xD005C9D1,0x955C9F5A,0x5AB765A2,0x1FEE3329,
         0x43068E9E,0x065FD815,0xC9B422ED,0x8CED7466,
         0x3207B378,0x775EE5F3,0xB8B51F0B,0xFDEC4980,
         0x27088D6E,0x6251DBE5,0xADBA211D,0xE8E37796,
         0x5609B088,0x1350E603,0xDCBB1CFB,0x99E24A70,
         0xC50AF7C7,0x8053A14C,0x4FB85BB4,0x0AE10D3F,
         0xB40BCA21,0xF1529CAA,0x3EB96652,0x7BE030D9,
         0x860C7959,0xC3552FD2,0x0CBED52A,0x49E783A1,
         0xF70D44BF,0xB2541234,0x7DBFE8CC,0x38E6BE47,
         0x640E03F0,0x2157557B,0xEEBCAF83,0xABE5F908,
         0x150F3E16,0x5056689D,0x9FBD9265,0xDAE4C4EE,
         0x4E107FDC,0x0B492957,0xC4A2D3AF,0x81FB8524,
         0x3F11423A,0x7A4814B1,0xB5A3EE49,0xF0FAB8C2,
         0xAC120575,0xE94B53FE,0x26A0A906,0x63F9FF8D,
         0xDD133893,0x984A6E18,0x57A194E0,0x12F8C26B,
         0xEF148BEB,0xAA4DDD60,0x65A62798,0x20FF7113,
         0x9E15B60D,0xDB4CE086,0x14A71A7E,0x51FE4CF5,
         0x0D16F142,0x484FA7C9,0x87A45D31,0xC2FD0BBA,
         0x7C17CCA4,0x394E9A2F,0xF6A560D7,0xB3FC365C,
         0x6918F2B2,0x2C41A439,0xE3AA5EC1,0xA6F3084A,
```

```
0x1819CF54,0x5D4099DF,0x92AB6327,0xD7F235AC,
0x8B1A881B,0xCE43DE90,0x01A82468,0x44F172E3,
0xFA1BB5FD,0xBF42E376,0x70A9198E,0x35F04F05,
0xC81C0685,0x8D45500E,0x42AEAAF6,0x07F7FC7D,
0xB91D3B63,0xFC446DE8,0x33AF9710,0x76F6C19B,
0x2A1E7C2C,0x6F472AA7,0xA0ACD05F,0xE5F586D4,
0x5B1F41CA,0x1E461741,0xD1ADEDB9,0x94F4BB32,
0x9C20FEDD,0xD979A856,0x169252AE,0x53CB0425,
0xED21C33B,0xA87895B0,0x67936F48,0x22CA39C3,
0x7E228474,0x3B7BD2FF,0xF4902807,0xB1C97E8C,
0x0F23B992,0x4A7AEF19,0x859115E1,0xC0C8436A,
0x3D240AEA,0x787D5C61,0xB796A699,0xF2CFF012,
0x4C25370C,0x097C6187,0xC6979B7F,0x83CECDF4,
0xDF267043,0x9A7F26C8,0x5594DC30,0x10CD8ABB,
0xAE274DA5,0xEB7E1B2E,0x2495E1D6,0x61CCB75D,
0xBB2873B3,0xFE712538,0x319ADFC0,0x74C3894B,
0xCA294E55,0x8F7018DE,0x409BE226,0x05C2B4AD,
0x592A091A,0x1C735F91,0xD398A569,0x96C1F3E2,
0x282B34FC,0x6D726277,0xA299988F,0xE7C0CE04,
0x1A2C8784,0x5F75D10F,0x909E2BF7,0xD5C77D7C,
0x6B2DBA62,0x2E74ECE9,0xE19F1611,0xA4C6409A,
0xF82EFD2D,0xBD77ABA6,0x729C515E,0x37C507D5,
0x892FC0CB,0xCC769640,0x039D6CB8,0x46C43A33,
0xD2308101,0x9769D78A,0x58822D72,0x1DDB7BF9,
0xA331BCE7,0xE668EA6C,0x29831094,0x6CDA461F,
0x3032FBA8,0x756BAD23,0xBA8057DB,0xFFD90150,
0x4133C64E,0x046A90C5,0xCB816A3D,0x8ED83CB6,
0x73347536,0x366D23BD,0xF986D945,0xBCDF8FCE,
0x023548D0,0x476C1E5B,0x8887E4A3,0xCDDEB228,
0x91360F9F,0xD46F5914,0x1B84A3EC,0x5EDDF567,
0xE0373279,0xA56E64F2,0x6A859E0A,0x2FDCC881,
0xF5380C6F,0xB0615AE4,0x7F8AA01C,0x3AD3F697,
0x84393189,0xC1606702,0x0E8B9DFA,0x4BD2CB71,
0x173A76C6,0x5263204D,0x9D88DAB5,0xD8D18C3E,
0x663B4B20,0x23621DAB,0xEC89E753,0xA9D0B1D8,
0x543CF858,0x1165AED3,0xDE8E542B,0x9BD702A0,
0x253DC5BE,0x60649335,0xAF8F69CD,0xEAD63F46,
0xB63E82F1,0xF367D47A,0x3C8C2E82,0x79D57809,
0xC73FBF17,0x8266E99C,0x4D8D1364,0x08D445EF };
```

Appendix B. A Simple Implementation Example of KCipher-2

We provide an example implementation of KCipher-2 written in C. The implementation is simple; we do not consider storage or time complexity, nor do we consider software engineering-related issues, such as encapsulation, modularity, and so on.

B.1. Code Components I - Definitions and Declarations

```
#include <stdio.h>
#include <stdint.h>

#define INIT    0
#define NORMAL  1

void init (unsigned int *, unsigned int *);
void next(int);
void stream (unsigned int *, unsigned int *);

static const uint8_t S_box[256] = {
    ...
    // as defined in Section 2.4.3
};

static const uint32_t amul0[256] = {
    ...
    // as defined in Appendix A.1
};

static const uint32_t amul1[256] = {
    ...
    // as defined in Appendix A.2
};

static const uint32_t amul2[256] = {
    ...
    // as defined in Appendix A.3
};

static const uint32_t amul3[256] = {
    ...
    // as defined in Appendix A.4
};

/* Global variables */

// State S
uint32_t A[5];              // five 32-bit units
```

```
    uint32_t B[11];             // eleven 32-bit units
    uint32_t L1, R1, L2, R2;    // one 32-bit unit for each

    // The internal key (IK) and the initialization vector (IV)
    uint32_t IK[12];    // (12*32) bits
    uint32_t IV[4];     // (4*32) bits
```

B.2. Code Components II - Functions

```
/**
* Do multiplication in GF(2#8) of the irreducible polynomial,
* f(x) = x#8 + x#4 + x#3 + x + 1. The given parameter is multiplied
* by 2.
* @param    t : (INPUT). 8 bits. The number will be multiplied by 2
* @return     : (OUTPUT). 8 bits. The multiplication result
*/
uint8_t GF_mult_by_2 (uint8_t t) {
    uint8_t q;
    uint32_t lq;

    lq = t << 1;
    if ((lq & 0x100) != 0) lq ^= 0x011B;
    q = lq ^ 0xFF;

    return q;
}

/**
* Do multiplication in GF(2#8) of the irreducible polynomial,
* f(x) = x#8 + x#4 + x#3 + x + 1. The given parameter is multiplied
* by 3.
* @param    t   : (INPUT). 8 bits. The number will be multiplied by 3
* @return       : (OUTPUT). 8 bits. The multiplication result
*/
uint8_t GF_mult_by_3 (uint8_t t) {
    uint8_t q;
    uint32_t lq;

    lq = (t << 1) ^ t;
    if ((lq & 0x100) != 0) lq ^= 0x011B;
    q = lq ^ 0xFF;

    return q;
}
```

```
/**
* Do substitution on a given input. See Section 2.4.2.
* @param    t   : (INPUT), (1*32) bits
* @return       : (OUTPUT), (1*32) bits
*/
uint32_t sub_k2 (uint32_t in) {
    uint32_t out;

    uint8_t w0 = in & 0x000000ff;
    uint8_t w1 = (in >> 8) & 0x000000ff;
    uint8_t w2 = (in >> 16) & 0x000000ff;
    uint8_t w3 = (in >> 24) & 0x000000ff;

    uint8_t t3, t2, t1, t0;
    uint8_t q3, q2, q1, q0;

    t0 = S_box[w0]; t1 = S_box[w1]; t2 = S_box[w2]; t3 = S_box[w3];

    q0 = GF_mult_by_2(t0) ^ GF_mult_by_3(t1) ^ t2 ^ t3;
    q1 = t0 ^ GF_mult_by_2(t1) ^ GF_mult_by_3(t2) ^ t3;
    q2 = t0 ^ t1 ^ GF_mult_by_2(t2) ^ GF_mult_by_3(t3);
    q3 = GF_mult_by_3(t0) ^ t1 ^ t2 ^ GF_mult_by_2(t3);

    out = (q3 << 24) | (q2 << 16) | (q1 << 8) | q0;

    return out;
}

/**
* Expand a given 128-bit key (K) to a 384-bit internal key
* information (IK).
* See Step 1 of init() in Section 2.3.2.
* @param    key[4]  : (INPUT), (4*32) bits
* @param    iv[4]   : (INPUT), (4*32) bits
* @modify   IK[12]  : (OUTPUT), (12*32) bits
* @modify   IV[12]  : (OUTPUT), (4*32) bits
*/
void key_expansion (uint32_t *key, uint32_t *iv) {
    // copy iv to IV
    IV[0] = iv[0];  IV[1] = iv[1];  IV[2] = iv[2];  IV[3] = iv[3];

    // m = 0 ... 3
    IK[0] = key[0];     IK[1] = key[1];
    IK[2] = key[2];     IK[3] = key[3];
    // m = 4
    IK[4] = IK[0] ^ sub_k2((IK[3] << 8) ^ (IK[3] >> 24)) ^
            0x01000000;
```

```
        // m = 4 ... 11, but not 4 nor 8
        IK[5] = IK[1] ^ IK[4];  IK[6] = IK[2] ^ IK[5];
        IK[7] = IK[3] ^ IK[6];

        // m = 8
        IK[8] = IK[4] ^ sub_k2((IK[7] << 8) ^ (IK[7] >> 24)) ^
                0x02000000;

        // m = 4 ... 11, but not 4 nor 8
        IK[9] = IK[5] ^ IK[8];  IK[10] = IK[6] ^ IK[9];
        IK[11] = IK[7] ^ IK[10];
    }

    /**
    * Set up the initial state value using IK and IV. See Step 2 of
    * init() in Section 2.3.2.
    * @param    key[4]  : (INPUT), (4*32) bits
    * @param    iv[4]   : (INPUT), (4*32) bits
    * @modify   S       : (OUTPUT), (A, B, L1, R1, L2, R2)
    */
    void setup_state_values (uint32_t *key, uint32_t *iv) {
        // setting up IK and IV by calling key_expansion(key, iv)
        key_expansion(key, iv);

        // setting up the internal state values
        A[0] = IK[4];   A[1] = IK[3];   A[2] = IK[2];
        A[3] = IK[1];   A[4] = IK[0];

        B[0] = IK[10];  B[1] = IK[11];  B[2] = IV[0];   B[3] = IV[1];
        B[4] = IK[8];   B[5] = IK[9];   B[6] = IV[2];   B[7] = IV[3];
        B[8] = IK[7];   B[9] = IK[5];   B[10] = IK[6];

        L1 = R1 = L2 = R2 = 0x00000000;
    }

    /**
    * Initialize the system with a 128-bit key (K) and a 128-bit
    * initialization vector (IV). It sets up the internal state value
    * and invokes next(INIT) iteratively 24 times. After this,
    * the system is ready to produce key streams. See Section 2.3.2.
    * @param    key[12] : (INPUT), (4*32) bits
    * @param    iv[4]   : (INPUT), (4*32) bits
    * @modify   IK      : (12*32) bits, by calling setup_state_values()
    * @modify   IV      : (4*32) bits,  by calling setup_state_values()
    * @modify   S       : (OUTPUT), (A, B, L1, R1, L2, R2)
    */
    void init (uint32_t *k, uint32_t *iv) {
        int i;
```

```
        setup_state_values(k, iv);

        for(i=0; i < 24; i++) {
            next(INIT);
        }
    }

    /**
    * Non-linear function. See Section 2.4.1.
    * @param    A   : (INPUT), 8 bits
    * @param    B   : (INPUT), 8 bits
    * @param    C   : (INPUT), 8 bits
    * @param    D   : (INPUT), 8 bits
    * @return      : (OUTPUT), 8 bits
    */
    uint32_t NLF (uint32_t A, uint32_t B,
            uint32_t C, uint32_t D ) {
        uint32_t Q;

        Q = (A + B) ^ C ^ D;

        return Q;
    }

    /**
    * Derive a new state from the current state values.
    * See Section 2.3.1.
    * @param    mode    : (INPUT) INIT (= 0) or NORMAL (= 1)
    * @modify   S       : (OUTPUT)
    */
    void next (int mode) {
        uint32_t nA[5];
        uint32_t nB[11];
        uint32_t nL1, nR1, nL2, nR2;
        uint32_t temp1, temp2;

        nL1 = sub_k2(R2 + B[4]);
        nR1 = sub_k2(L2 + B[9]);
        nL2 = sub_k2(L1);
        nR2 = sub_k2(R1);

        // m = 0 ... 3
        nA[0] = A[1];   nA[1] = A[2];   nA[2] = A[3];   nA[3] = A[4];

        // m = 0 ... 9
        nB[0] = B[1];   nB[1] = B[2];   nB[2] = B[3];   nB[3] = B[4];
        nB[4] = B[5];   nB[5] = B[6];   nB[6] = B[7];   nB[7] = B[8];
        nB[8] = B[9];   nB[9] = B[10];
```

```
        // update nA[4]
        temp1 = (A[0] << 8) ^ amul0[(A[0] >> 24)];
        nA[4] = temp1 ^ A[3];
        if (mode == INIT)
            nA[4] ^= NLF(B[0], R2, R1, A[4]);

        // update nB[10]
        if (A[2] & 0x40000000) /* if A[2][30] == 1 */ {
            temp1 = (B[0] << 8) ^ amul1[(B[0] >> 24)];
        } else /*if A[2][30] == 0*/ {
            temp1 = (B[0] << 8) ^ amul2[(B[0] >> 24)];
        }

        if (A[2] & 0x80000000) /* if A[2][31] == 1 */ {
            temp2 = (B[8] << 8) ^ amul3[(B[8] >> 24)];
        } else /* if A[2][31] == 0 */ {
            temp2 = B[8];
        }

        nB[10] = temp1 ^ B[1] ^ B[6] ^ temp2;

        if (mode == INIT)
            nB[10] ^= NLF(B[10], L2, L1, A[0]);

        /* copy S' to S */
        A[0] = nA[0];   A[1] = nA[1];   A[2] = nA[2];
        A[3] = nA[3];   A[4] = nA[4];

        B[0] = nB[0];   B[1] = nB[1];   B[2] = nB[2];   B[3] = nB[3];
        B[4] = nB[4];   B[5] = nB[5];   B[6] = nB[6];   B[7] = nB[7];
        B[8] = nB[8];   B[9] = nB[9];   B[10] = nB[10];

        L1 = nL1;   R1 = nR1;   L2 = nL2;   R2 = nR2;
    }

    /**
    * Obtain a key stream = (ZH, ZL) from the current state values.
    * See Section 2.3.3.
    * @param    ZH  : (OUTPUT) (1 * 32)-bit
    * @modify   ZL  : (OUTPUT) (1 * 32)-bit
    */
    void stream (uint32_t *ZH, uint32_t *ZL) {
        *ZH = NLF(B[10], L2, L1, A[0]);
        *ZL = NLF(B[0], R2, R1, A[4]);
    }
```

B.3. Use Case

```
void main (void)  {

    // Set the key and the iv
    uint32_t key[4] = ...;
    uint32_t iv[4] = ...;

    init(key, iv);

    // produce a key stream
    stream(&zh, &zl);
    next(NORMAL);

    // produce another key stream
    stream(&zh, &zl);
    next(NORMAL);
    ...
}
```

Appendix C. Test Vectors

This appendix provides running examples of KCipher-2 obtained from the naive implementation. All values are written in hexadecimal form.

C.1. Key Stream Generation Examples

The following is a series of the 64-bit key streams generated from the given 8-bit keys (K) and 128-bit initialization vectors (IVs).

- K : 00000000 00000000 00000000 00000000
- IV: 00000000 00000000 00000000 00000000
- Generated key streams at S(i) are as follows
  S(0): F871EBEF 945B7272
  S(1): E40C0494 1DFF0537
  S(2): 0B981A59 FBC8AC57
  S(3): 566D3B02 C179DBB4
  S(4): 3B46F1F0 33554C72
  S(5): 5DE68BCC 9872858F
  S(6): 57549602 4062F0E9
  S(7): F932C998 226DB6BA
  ...

- K : A37B7D01 2F897076 FE08C22D 142BB2CF
- IV: 33A6EE60 E57927E0 8B45CC4C A30EDE4A
- Generated key streams at S(i) are as follows
  S(0): 60E9A6B6 7B4C2524

```
  S(1): FE726D44 AD5B402E
  S(2): 31D0D1BA 5CA233A4
  S(3): AFC74BE7 D6069D36
  S(4): 4A75BB6C D8D5B7F0
  S(5): 38AAAA28 4AE4CD2F
  S(6): E2E5313D FC6CCD8F
  S(7): 9D2484F2 0F86C50D
  ...

- K : 3D62E9B1 8E5B042F 42DF43CC 7175C96E
- IV: 777CEFE4 541300C8 ADCACA8A 0B48CD55
- Generated key streams at S(i) are as follows
  S(0): 690F108D 84F44AC7
  S(1): BF257BD7 E394F6C9
  S(2): AA1192C3 8E200C6E
  S(3): 073C8078 AC18AAD1
  S(4): D4B8DADE 68802368
  S(5): 2FA42076 83DEA5A4
  S(6): 4C1D95EA E959F5B4
  S(7): 2611F41E A40F0A58
  ...
```

C.2. Another Key Stream Generation with the State Values

In this section, the initialization procedure and the key stream generation are illustrated in detail. The given 128-bit key (K) and the 128-bit initialization vector (IV) are as follows:

```
- K : 0F1E2D3C 4B5A6978 8796A5B4 C3D2E1F0
- IV: F0E0D0C0 B0A09080 70605040 30201000.
```

Based on K and IV, the init() operation (Section 2.3.2) sets up the internal state values, S = (A, B, L1, R1, L2, R2), as follows:

```
A[0]: 7993A6A2    A[1]: C3D2E1F0    A[2]: 8796A5B4
A[3]: 4B5A6978    A[4]: 0F1E2D3C

B[0]: 38AB371B    B[1] : 4E26BC85   B[2]: F0E0D0C0
B[3]: B0A09080    B[4] : BF3D92AF   B[5]: 8DF45D75
B[6]: 70605040    B[7] : 30201000   B[8]: 768D8B9E
B[9]: 32C9CFDA    B[10]: B55F6A6E

L1: 00000000   R1: 00000000   L2: 00000000   R2: 00000000
```

To complete the initialization, the next() operation is applied to the state values 24 times (in Section 2.3.2, Step 3). Let us denote each repeated application of the next() operation by init(i), where 1 <= i <= 24. The internal state values resulting from each init(i) are shown in Appendixes C.2.1 - C.2.24.

C.2.1. S after init(1)

```
A[0]: C3D2E1F0    A[1]: 8796A5B4    A[2]: 4B5A6978
A[3]: 0F1E2D3C    A[4]: 37070F7F

B[0]: 4E26BC85    B[1] : F0E0D0C0   B[2]: B0A09080
B[3]: BF3D92AF    B[4] : 8DF45D75   B[5]: 70605040
B[6]: 30201000    B[7] : 768D8B9E   B[8]: 32C9CFDA
B[9]: B55F6A6E    B[10]: 64DEFF24

L1: F360860C   R1: E81907D5   L2: 63636363   R2: 63636363
```

C.2.2. S after init(2)

```
A[0]: 8796A5B4    A[1]: 4B5A6978    A[2]: 0F1E2D3C
A[3]: 37070F7F    A[4]: 25BCF981

B[0]: F0E0D0C0    B[1] : B0A09080   B[2]: BF3D92AF
B[3]: 8DF45D75    B[4] : 70605040   B[5]: 30201000
B[6]: 768D8B9E    B[7] : 32C9CFDA   B[8]: B55F6A6E
B[9]: 64DEFF24    B[10]: 7E65CB6A

L1: 1B9542ED   R1: 9B259D28   L2: 971610F6   R2: 39C36E1D
```

C.2.3. S after init(3)

```
A[0]: 4B5A6978    A[1]: 0F1E2D3C    A[2]: 37070F7F
A[3]: 25BCF981    A[4]: FA2DD9D3

B[0]: B0A09080    B[1] : BF3D92AF   B[2]: 8DF45D75
B[3]: 70605040    B[4] : 30201000   B[5]: 768D8B9E
B[6]: 32C9CFDA    B[7] : B55F6A6E   B[8]: 64DEFF24
B[9]: 7E65CB6A    B[10]: 08573732

L1: 1F41CDFB   R1: CFAE13F3   L2: BCC7DC5B   R2: 1528DDA1
```

C.2.4. S after init(4)

```
A[0]: 0F1E2D3C    A[1]: 37070F7F    A[2]: 25BCF981
A[3]: FA2DD9D3    A[4]: AB820031

B[0]: BF3D92AF    B[1] : 8DF45D75   B[2]: 70605040
B[3]: 30201000    B[4] : 768D8B9E   B[5]: 32C9CFDA
B[6]: B55F6A6E    B[7] : 64DEFF24   B[8]: 7E65CB6A
B[9]: 08573732    B[10]: 40941D82

L1: 8D7100A7   R1: AA6C8F89   L2: B4F43081   R2: 81264AF3
```

C.2.5. S after init(5)

```
A[0]: 37070F7F    A[1]: 25BCF981    A[2]: FA2DD9D3
A[3]: AB820031    A[4]: D8F5995F

B[0]: 8DF45D75    B[1] : 70605040   B[2]: 30201000
B[3]: 768D8B9E    B[4] : 32C9CFDA   B[5]: B55F6A6E
B[6]: 64DEFF24    B[7] : 7E65CB6A   B[8]: 08573732
B[9]: 40941D82    B[10]: 1A8DA7FB

L1: D315A91D   R1: 751BC887   L2: 9E8539E3   R2: 929B1D3C
```

C.2.6. S after init(6)

```
A[0]: 25BCF981    A[1]: FA2DD9D3    A[2]: AB820031
A[3]: D8F5995F    A[4]: F697B5BB

B[0]: 70605040    B[1] : 30201000   B[2]: 768D8B9E
B[3]: 32C9CFDA    B[4] : B55F6A6E   B[5]: 64DEFF24
B[6]: 7E65CB6A    B[7] : 08573732   B[8]: 40941D82
B[9]: 1A8DA7FB    B[10]: 13B5E7F3

L1: 88658E94   R1: 7F1C023D   L2: B16F9402   R2: 5F06AB3F
```

C.2.7. S after init(7)

```
A[0]: FA2DD9D3    A[1]: AB820031    A[2]: D8F5995F
A[3]: F697B5BB    A[4]: 6B0A7012

B[0]: 30201000    B[1] : 768D8B9E   B[2]: 32C9CFDA
B[3]: B55F6A6E    B[4] : 64DEFF24   B[5]: 7E65CB6A
B[6]: 08573732    B[7] : 40941D82   B[8]: 1A8DA7FB
B[9]: 13B5E7F3    B[10]: D76ABD2C

L1: 21BF8813   R1: 743F68DE   L2: A1F603E6   R2: 3D1EA499
```

C.2.8. S after init(8)

```
A[0]: AB820031    A[1]: D8F5995F    A[2]: F697B5BB
A[3]: 6B0A7012    A[4]: 23995B7E

B[0]: 768D8B9E    B[1] : 32C9CFDA   B[2]: B55F6A6E
B[3]: 64DEFF24    B[4] : 7E65CB6A   B[5]: 08573732
B[6]: 40941D82    B[7] : 1A8DA7FB   B[8]: 13B5E7F3
B[9]: D76ABD2C    B[10]: 997C3F70

L1: B48EA08C   R1: 657C8FFD   L2: AAB50B58   R2: 281F9A12
```

C.2.9. S after init(9)

```
A[0]: D8F5995F    A[1]: F697B5BB    A[2]: 6B0A7012
A[3]: 23995B7E    A[4]: F8532F87

B[0]: 32C9CFDA    B[1] : B55F6A6E   B[2]: 64DEFF24
B[3]: 7E65CB6A    B[4] : 08573732   B[5]: 40941D82
B[6]: 1A8DA7FB    B[7] : 13B5E7F3   B[8]: D76ABD2C
B[9]: 997C3F70    B[10]: 95FFF657

L1: A2040C44   R1: EF19DC4E   L2: 543A1967   R2: 05D0CF60
```

C.2.10. S after init(10)

```
A[0]: F697B5BB    A[1]: 6B0A7012    A[2]: 23995B7E
A[3]: F8532F87    A[4]: BEDF1DEF

B[0]: B55F6A6E    B[1] : 64DEFF24   B[2]: 7E65CB6A
B[3]: 08573732    B[4] : 40941D82   B[5]: 1A8DA7FB
B[6]: 13B5E7F3    B[7] : D76ABD2C   B[8]: 997C3F70
B[9]: 95FFF657    B[10]: 6D2C2FA3

L1: C7AE66B0   R1: 9C075DB9   L2: 5554CBE7   R2: 866080C4
```

C.2.11. S after init(11)

```
A[0]: 6B0A7012    A[1]: 23995B7E    A[2]: F8532F87
A[3]: BEDF1DEF    A[4]: 983D37.

B[0]: 64DEFF24    B[1] : 7E65CB6A   B[2]: 08573732
B[3]: 40941D82    B[4] : 1A8DA7FB   B[5]: 13B5E7F3
B[6]: D76ABD2C    B[7] : 997C3F70   B[8]: 95FFF657
B[9]: 6D2C2FA3    B[10]: A02127BE

L1: 29F322A2   R1: 01F771D9   L2: 725670A2   R2: D4F24463
```

C.2.12. S after init(12)

```
A[0]: 23995B7E    A[1]: F8532F87    A[2]: BEDF1DEF
A[3]: 983D37CB    A[4]: 526A110D

B[0]: 7E65CB6A    B[1] : 08573732   B[2]: 40941D82
B[3]: 1A8DA7FB    B[4] : 13B5E7F3   B[5]: D76ABD2C
B[6]: 997C3F70    B[7] : 95FFF657   B[8]: 6D2C2FA3
B[9]: A02127BE    B[10]: 49F99042

L1: 51536DF4   R1: 66111E6A   L2: 8147B572   R2: 6CC2AC80
```

C.2.13. S after init(13)

```
A[0]: F8532F87    A[1]: BEDF1DEF    A[2]: 983D37CB
A[3]: 526A110D    A[4]: A5EEB8AE

B[0]: 08573732    B[1] : 40941D82   B[2]: 1A8DA7FB
B[3]: 13B5E7F3    B[4] : D76ABD2C   B[5]: 997C3F70
B[6]: 95FFF657    B[7] : 6D2C2FA3   B[8]: A02127BE
B[9]: 49F99042    B[10]: 406CE62C

L1: 9582D912   R1: 6953AFE8   L2: B22A3A1D   R2: 903A4823
```

C.2.14. S after init(14)

```
A[0]: BEDF1DEF    A[1]: 983D37CB    A[2]: 526A110D
A[3]: A5EEB8AE    A[4]: 70A5B5BA

B[0]: 40941D82    B[1] : 1A8DA7FB   B[2]: 13B5E7F3
B[3]: D76ABD2C    B[4] : 997C3F70   B[5]: 95FFF657
B[6]: 6D2C2FA3    B[7] : A02127BE   B[8]: 49F99042
B[9]: 406CE62C    B[10]: C57BED5B

L1: EB77DD2D   R1: 633CFD8F   L2: 32A4BCEF   R2: CB33BCB2
```

C.2.15. S after init(15)

```
A[0]: 983D37CB    A[1]: 526A110D    A[2]: A5EEB8AE
A[3]: 70A5B5BA    A[4]: B1145F18

B[0]: 1A8DA7FB    B[1] : 13B5E7F3   B[2]: D76ABD2C
B[3]: 997C3F70    B[4] : 95FFF657   B[5]: 6D2C2FA3
B[6]: A02127BE    B[7] : 49F99042   B[8]: 406CE62C
B[9]: C57BED5B    B[10]: 7BE2C520

L1: E11420CC   R1: 6730A956   L2: 8EC8ACEF   R2: C7FC060A
```

C.2.16. S after init(16)

```
A[0]: 526A110D    A[1]: A5EEB8AE    A[2]: 70A5B5BA
A[3]: B1145F18    A[4]: FA752FDC

B[0]: 13B5E7F3    B[1] : D76ABD2C   B[2]: 997C3F70
B[3]: 95FFF657    B[4] : 6D2C2FA3   B[5]: A02127BE
B[6]: 49F99042    B[7] : 406CE62C   B[8]: C57BED5B
B[9]: 7BE2C520    B[10]: 1F48829C

L1: 0D95C94D   R1: 8238B05F   L2: 7B00D356   R2: 0EFE8596
```

C.2.17. S after init(17)

```
A[0]: A5EEB8AE    A[1]: 70A5B5BA    A[2]: B1145F18
A[3]: FA752FDC    A[4]: DB29190A

B[0]: D76ABD2C    B[1] : 997C3F70   B[2]: 95FFF657
B[3]: 6D2C2FA3    B[4] : A02127BE   B[5]: 49F99042
B[6]: 406CE62C    B[7] : C57BED5B   B[8]: 7BE2C520
B[9]: 1F48829C    B[10]: F95DD14F

L1: 262687B5   R1: 9B9AC5E9   L2: 7C08EB5C   R2: 8C1300A3
```

C.2.18. S after init(18)

```
A[0]: 70A5B5BA    A[1]: B1145F18    A[2]: FA752FDC
A[3]: DB29190A    A[4]: 35623CDA

B[0]: 997C3F70    B[1] : 95FFF657   B[2]: 6D2C2FA3
B[3]: A02127BE    B[4] : 49F99042   B[5]: 406CE62C
B[6]: C57BED5B    B[7] : 7BE2C520   B[8]: 1F48829C
B[9]: F95DD14F    B[10]: D939E13E

L1: E478DEF0   R1: 06F84503   L2: 71350E88   R2: 14EF8E61
```

C.2.19. S after init(19)

```
A[0]: B1145F18    A[1]: FA752FDC    A[2]: DB29190A
A[3]: 35623CDA    A[4]: 746B4AE8

B[0]: 95FFF657    B[1] : 6D2C2FA3   B[2]: A02127BE
B[3]: 49F99042    B[4] : 406CE62C   B[5]: C57BED5B
B[6]: 7BE2C520    B[7] : 1F48829C   B[8]: F95DD14F
B[9]: D939E13E    B[10]: 9970C980

L1: C2AC94C4   R1: C708FAE8   L2: FC4900F1   R2: 7C260B6A
```

C.2.20. S after init(20)

```
A[0]: FA752FDC    A[1]: DB29190A    A[2]: 35623CDA
A[3]: 746B4AE8    A[4]: 2EB9213A

B[0]: 6D2C2FA3    B[1] : A02127BE   B[2]: 49F99042
B[3]: 406CE62C    B[4] : C57BED5B   B[5]: 7BE2C520
B[6]: 1F48829C    B[7] : F95DD14F   B[8]: D939E13E
B[9]: 9970C980    B[10]: 3C517031

L1: 8F007DE9   R1: B2AE0889   L2: DD68D5EA   R2: 3C8757AC
```

C.2.21. S after init(21)

```
A[0]: DB29190A    A[1]: 35623CDA    A[2]: 746B4AE8
A[3]: 2EB9213A    A[4]: BE3CA984

B[0]: A02127BE    B[1] : 49F99042   B[2]: 406CE62C
B[3]: C57BED5B    B[4] : 7BE2C520   B[5]: 1F48829C
B[6]: F95DD14F    B[7] : D939E13E   B[8]: 9970C980
B[9]: 3C517031    B[10]: D1439B63

L1: AFC4E32F   R1: 98FBC87F   L2: 58B22D36   R2: 481DC7D6
```

C.2.22. S after init(22)

```
A[0]: 35623CDA    A[1]: 746B4AE8    A[2]: 2EB9213A
A[3]: BE3CA984    A[4]: 974E6719

B[0]: 49F99042    B[1] : 406CE62C   B[2]: C57BED5B
B[3]: 7BE2C520    B[4] : 1F48829C   B[5]: F95DD14F
B[6]: D939E13E    B[7] : 9970C980   B[8]: 3C517031
B[9]: D1439B63    B[10]: 9334E221

L1: F9C43357   R1: E5539EA2   L2: C0B76A7C   R2: 06EE4ED5
```

C.2.23. S after init(23)

```
A[0]: 746B4AE8    A[1]: 2EB9213A    A[2]: BE3CA984
A[3]: 974E6719    A[4]: 86916EFF

B[0]: 406CE62C    B[1] : C57BED5B   B[2]: 7BE2C520
B[3]: 1F48829C    B[4] : F95DD14F   B[5]: D939E13E
B[6]: 9970C980    B[7] : 3C517031   B[8]: D1439B63
B[9]: 9334E221    B[10]: 50EF13E7

L1: 309527ED   R1: C473D814   L2: 1B107B6D   R2: 0180D95D
```

C.2.24. S(0) after init(24)

```
A[0]: 2EB9213A    A[1]: BE3CA984    A[2]: 974E6719
A[3]: 86916EFF    A[4]: F52DACF9

B[0]: C57BED5B    B[1] : 7BE2C520   B[2]: 1F48829C
B[3]: F95DD14F    B[4] : D939E13E   B[5]: 9970C980
B[6]: 3C517031    B[7] : D1439B63   B[8]: 9334E221
B[9]: 50EF13E7    B[10]: E0BD9F91

L1: 4370D8E6   R1: DABED76C   L2: 11C1ACCB   R2: C3BAAEDF
```

Note that the result of init(24) is also referred to as S(0) (in Section 2.3.2). Since the state is S(0), the stream() operation (in Section 2.3.3) can be applied and generate key streams.

```
Key stream at S(0) : 9FB6B580A6A5E7AF
```

Henceforth, a new key stream can be produced by 1) obtaining a new state by applying the next() operation to the current state, and 2) generating a new key stream by applying the stream() operation to the new state.

C.2.25. S(1) and the Key Stream at S(1)

```
A[0]: BE3CA984    A[1]: 974E6719    A[2]: 86916EFF
A[3]: F52DACF9    A[4]: 960329B5

B[0]: 7BE2C520    B[1] : 1F48829C   B[2]: F95DD14F
B[3]: D939E13E    B[4] : 9970C980   B[5]: 3C517031
B[6]: D1439B63    B[7] : 9334E221   B[8]: 50EF13E7
B[9]: E0BD9F91    B[10]: 5318AEE1

L1: 8FD86092   R1: 4BBDC0F6   L2: 8D63A5EF   R2: FEE0F24B

Key stream at S(1) : D1989DC6A77D5E28
```

C.2.26. S(2) and the Key Stream at S(2)

```
A[0]: 974E6719    A[1]: 86916EFF    A[2]: F52DACF9
A[3]: 960329B5    A[4]: 1A3DB24E

B[0]: 1F48829C    B[1] : F95DD14F   B[2]: D939E13E
B[3]: 9970C980    B[4] : 3C517031   B[5]: D1439B63
B[6]: 9334E221    B[7] : 50EF13E7   B[8]: E0BD9F91
B[9]: 5318AEE1    B[10]: C86C2C77

L1: 9686FE8C   R1: FAF89251   L2: 86C824E7   R2: 7BC21098

Key stream at S(2) : 4EFCC8CB7BCFB32B
```

Authors' Addresses

Shinsaku Kiyomoto
KDDI R&D Laboratories, Inc.
2-1-15 Ohara
Fujimino-shi, Saitama 356-8502
Japan

Phone: +81-49-278-7885
Fax: +81-49-278-7510
EMail: kiyomoto@kddilabs.jp

Wook Shin
KDDI R&D Laboratories, Inc.
2-1-15 Ohara
Fujimino-shi, Saitama 356-8502
Japan

EMail: ohpato@hanmail.net