

Project Assignment 1

1. Introduction

In this project, we need to implement a stream cipher that generates a sequence of 32-bit words under the control of a 128-bit key and a 128-bit initialization variable. These words can be used to mask the plaintext. First a key initialization is performed, i.e. the cipher is clocked without producing output, and then with every clock tick it produces a 32-bit word of output.

16进制

We use the prefix **0x** to indicate **hexadecimal** numbers. All data variables are presented with the most significant bit on the left hand side and the least significant bit on the right hand side. Where a variable is broken down into a number of sub-strings, the left most (most significant) sub-string is numbered 0, the next most significant is numbered 1 and so on through to the least significant.

For example if a 64-bit value *X* is subdivided into four 16-bit substrings *P*, *Q*, *R*, *S* we have:

$X = 0x0123456789ABCDEF$

with

$P = 0x0123$, $Q = 0x4567$, $R = 0x89AB$, $S = 0xCDEF$.

In binary this would be:

$X = 0000000100100011010001010110011110001001101010111100110111101111$

with $P = 0000000100100011$

$Q = 0100010101100111$

$R = 1000100110101011$

$S = 1100110111101111$

We use the assignment operator '=', as used in several programming languages. When we write

$\langle \text{variable} \rangle = \langle \text{expression} \rangle$

we mean that $\langle \text{variable} \rangle$ assumes the value that $\langle \text{expression} \rangle$ had before the assignment took place.

For instance,

$x = x + y + 3$

means

(new value of *x*) becomes (old value of *x*) + (old value of *y*) + 3.

We also define the following symbols:

= The assignment operator.

\oplus The bitwise exclusive-OR operation.

\boxplus Integer addition modulo 2^{32} .

\parallel The concatenation of the two operands.

$\ll_n t$ *t*-bit left shift in an *n*-bit register.

2. Functions defined

2.1 Function MULx()

MULx maps 16 bits to 8 bits. Let *V* and *c* be 8-bit input values. Then MULx is defined:

If the leftmost (i.e. the most significant) bit of *V* equals 1, then

$$\text{MUL}_x(V, c) = (V \ll_8 1) \oplus c,$$
 else

$$\text{MUL}_x(V, c) = V \ll_8 1.$$

For instance: $\text{MUL}_x(0x69, 0x1B) = 0xD2$; $\text{MUL}_x(0x96, 0x1B) = 0x37$.

2.2 Function MULy()

MULy maps 16 bits and a positive integer i to 8 bit. Let V and c be 8-bit input values, then $\text{MUL}_y(V, i, c)$ is recursively defined:

If i equals 0, then

$$\text{MUL}_y(V, i, c) = V,$$
 else

$$\text{MUL}_y(V, i, c) = \text{MUL}_x(\text{MUL}_y(V, i - 1, c), c).$$

3. Linear Feedback Shift Register (LFSR)

The Linear Feedback Shift Register (LFSR) consists of sixteen stages $s_0, s_1, s_2, \dots, s_{15}$ each holding 32 bits.

4. Finite State Machine (FSM)

The Finite State Machine (FSM) has three 32-bit registers R_1, R_2 and R_3 . The S-boxes S_1 and S_2 are used to update the registers R_2 and R_3 .

4.1 The 32x32-bit S-Box S_1

The S-Box S_1 maps a 32-bit input to a 32-bit output.

Let $w = w_0 \parallel w_1 \parallel w_2 \parallel w_3$ the 32-bit input with w_0 the most and w_3 the least significant byte. Let $S_1(w) = r_0 \parallel r_1 \parallel r_2 \parallel r_3$ with r_0 the most and r_3 the least significant byte. We use the 8 to 8 bit Rijndael S-Box S_R defined later.

Then r_0, r_1, r_2, r_3 are defined as:

$$\begin{array}{llll}
 r_0 = & \text{MUL}_x(S_R(w_0), 0x1B) & \oplus & S_R(w_1) & \oplus & S_R(w_2) & \oplus & \text{MUL}_x(S_R(w_3), 0x1B) \oplus S_R(w_3), \\
 r_1 = & \text{MUL}_x(S_R(w_0), 0x1B) \oplus S_R(w_0) & \oplus & \text{MUL}_x(S_R(w_1), 0x1B) & \oplus & S_R(w_2) & \oplus & S_R(w_3), \\
 r_2 = & S_R(w_0) & \oplus & \text{MUL}_x(S_R(w_1), 0x1B) \oplus S_R(w_1) & \oplus & \text{MUL}_x(S_R(w_2), 0x1B) & \oplus & S_R(w_3), \\
 r_3 = & S_R(w_0) & \oplus & S_R(w_1) & \oplus & \text{MUL}_x(S_R(w_2), 0x1B) \oplus S_R(w_2) & \oplus & \text{MUL}_x(S_R(w_3), 0x1B).
 \end{array}$$

4.2 The 32x32-bit S-Box S_2

The S-Box S_2 maps a 32-bit input to a 32-bit output.

Let $w = w_0 \parallel w_1 \parallel w_2 \parallel w_3$ the 32-bit input with w_0 the most and w_3 the least significant byte. Let $S_2(w) = r_0 \parallel r_1 \parallel r_2 \parallel r_3$ with r_0 the most and r_3 the least significant byte. We use the 8 to 8 bit S-Box S_Q defined later.

Then r_0, r_1, r_2, r_3 are defined as

$$\begin{array}{llll}
 r_0 = & \text{MUL}_x(S_Q(w_0), 0x69) & \oplus & S_Q(w_1) & \oplus & S_Q(w_2) & \oplus & \text{MUL}_x(S_Q(w_3), 0x69) \oplus S_Q(w_3), \\
 r_1 = & \text{MUL}_x(S_Q(w_0), 0x69) \oplus S_Q(w_0) & \oplus & \text{MUL}_x(S_Q(w_1), 0x69) & \oplus & S_Q(w_2) & \oplus & S_Q(w_3), \\
 r_2 = & S_Q(w_0) & \oplus & \text{MUL}_x(S_Q(w_1), 0x69) \oplus S_Q(w_1) & \oplus & \text{MUL}_x(S_Q(w_2), 0x69) & \oplus & S_Q(w_3), \\
 r_3 = & S_Q(w_0) & \oplus & S_Q(w_1) & \oplus & \text{MUL}_x(S_Q(w_2), 0x69) \oplus S_Q(w_2) & \oplus & \text{MUL}_x(S_Q(w_3), 0x69).
 \end{array}$$

5. The Clocking Operations

5.1 Clocking the LFSR

The clocking of the LFSR has two different modes of operation, the Initialization Mode (see Section 5.4) and the Keystream Mode (see Section 5.5).

In both modes the functions MUL_{α} and DIV_{α} are used which are defined as follows.

5.2 The function MUL_{α}

The function MUL_{α} maps 8 bits to 32 bits. Let c be the 8-bit input, then MUL_{α} is defined as $MUL_{\alpha}(c) =$

$$(MULy(c, 23, 0xA9) \parallel MULy(c, 245, 0xA9) \parallel MULy(c, 48, 0xA9) \parallel MULy(c, 239, 0xA9)).$$

5.3 The function DIV_{α}

The function DIV_{α} maps 8 bits to 32 bits. Let c be the 8-bit input, then DIV_{α} is defined as $DIV_{\alpha}(c) =$

$$(MULy(c, 16, 0xA9) \parallel MULy(c, 39, 0xA9) \parallel MULy(c, 6, 0xA9) \parallel MULy(c, 64, 0xA9)).$$

5.4 Initialization Mode

In the Initialization Mode the LFSR receives a 32-bit input word F , which is the output of the FSM. Let $s_0 = s_{0,0} \parallel s_{0,1} \parallel s_{0,2} \parallel s_{0,3}$ with $s_{0,0}$ being the most and $s_{0,3}$ being the least significant byte of s_0 .

Let $s_{11} = s_{11,0} \parallel s_{11,1} \parallel s_{11,2} \parallel s_{11,3}$ with $s_{11,0}$ being the most and $s_{11,3}$ being the least significant byte of s_{11} . Compute the intermediate value v as

$$v = (s_{0,1} \parallel s_{0,2} \parallel s_{0,3} \parallel 0x00) \oplus MUL_{\alpha}(s_{0,0}) \oplus s_2 \oplus (0x00 \parallel s_{11,0} \parallel s_{11,1} \parallel s_{11,2}) \oplus DIV_{\alpha}(s_{11,3}) \oplus F.$$

Set

$$\begin{aligned} s_0 &= s_1, & s_1 &= s_2, & s_2 &= s_3, & s_3 &= s_4, & s_4 &= s_5, & s_5 &= s_6, & s_6 &= s_7, & s_7 &= s_8, \\ s_8 &= s_9, & s_9 &= s_{10}, & s_{10} &= s_{11}, & s_{11} &= s_{12}, & s_{12} &= s_{13}, & s_{13} &= s_{14}, & s_{14} &= s_{15}, & s_{15} &= v. \end{aligned}$$

5.5 Keystream Mode

In the Keystream Mode the LFSR does not receive any input.

Let $s_0 = s_{0,0} \parallel s_{0,1} \parallel s_{0,2} \parallel s_{0,3}$ with $s_{0,0}$ being the most and $s_{0,3}$ being the least significant byte of s_0 .

Let $s_{11} = s_{11,0} \parallel s_{11,1} \parallel s_{11,2} \parallel s_{11,3}$ with $s_{11,0}$ being the most and $s_{11,3}$ being the least significant byte of s_{11} .

Compute the intermediate value v as

$$v = (s_{0,1} \parallel s_{0,2} \parallel s_{0,3} \parallel 0x00) \oplus MUL_{\alpha}(s_{0,0}) \oplus s_2 \oplus (0x00 \parallel s_{11,0} \parallel s_{11,1} \parallel s_{11,2}) \oplus DIV_{\alpha}(s_{11,3}).$$

Set

$$\begin{aligned} s_0 &= s_1, & s_1 &= s_2, & s_2 &= s_3, & s_3 &= s_4, & s_4 &= s_5, & s_5 &= s_6, & s_6 &= s_7, & s_7 &= s_8, \\ s_8 &= s_9, & s_9 &= s_{10}, & s_{10} &= s_{11}, & s_{11} &= s_{12}, & s_{12} &= s_{13}, & s_{13} &= s_{14}, & s_{14} &= s_{15}, & s_{15} &= v. \end{aligned}$$

5.6 Clocking the FSM

The FSM has two input words s_{15} and s_5 from the LFSR.

It produces a 32-bit output word F :

$$F = (s_{15} \boxplus R1) \oplus R2$$

Then the registers are updated.

Compute the intermediate value r as:

$$r = R2 \boxplus (R3 \oplus s_5).$$

Set

$$R3 = S_2(R2),$$

$$R2 = S_1(R1),$$

$$R1 = r.$$

6. Operation

6.1 Initialization

The cipher is initialized with a 128-bit key consisting of four 32-bit words $\mathbf{k}_0, \mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ and an 128-bit initialization variable consisting of four 32-bit words $\mathbf{IV}_0, \mathbf{IV}_1, \mathbf{IV}_2, \mathbf{IV}_3$ as follows.

Let $\mathbf{1}$ be the all-ones word (0xffffffff).

$$s_{15} = \mathbf{k}_3 \oplus \mathbf{IV}_0$$

$$s_{14} = \mathbf{k}_2$$

$$s_{13} = \mathbf{k}_1$$

$$s_{12} = \mathbf{k}_0 \oplus \mathbf{IV}_1$$

$$s_{11} = \mathbf{k}_3 \oplus \mathbf{1}$$

$$s_{10} = \mathbf{k}_2 \oplus \mathbf{1} \oplus \mathbf{IV}_2$$

$$s_9 = \mathbf{k}_1 \oplus \mathbf{1} \oplus \mathbf{IV}_3$$

$$s_8 = \mathbf{k}_0 \oplus \mathbf{1}$$

$$s_7 = \mathbf{k}_3$$

$$s_6 = \mathbf{k}_2$$

$$s_5 = \mathbf{k}_1$$

$$s_4 = \mathbf{k}_0$$

$$s_3 = \mathbf{k}_3 \oplus \mathbf{1}$$

$$s_2 = \mathbf{k}_2 \oplus \mathbf{1}$$

$$s_1 = \mathbf{k}_1 \oplus \mathbf{1}$$

$$s_0 = \mathbf{k}_0 \oplus \mathbf{1}$$

The FSM is initialized with $R1 = R2 = R3 = 0$.

Then the cipher runs in a special mode without producing output:

Repeat 32-times {

STEP 1: The FSM is clocked producing the 32-bit word F .

STEP 2: Then the LFSR is clocked in Initialization Mode consuming F .

}

6.2 Generation of Keystream

First the FSM is clocked once. The output word of the FSM is discarded.

Then the LFSR is clocked once in Keystream Mode.

After that n 32-bit words of keystream are produced:

for $t = 1$ to n {

STEP 1: The FSM is clocked and produces a 32-bit output word F .

STEP 2: The next keystream word is computed as $z_t = F \oplus s_0$.

STEP 3: Then the LFSR is clocked in Keystream Mode.

}

7. The Rijndael S-box S_R

The S-box S_R maps 8 bit to 8 bit. Here the input and output is presented in hexadecimal form.

Let x_0, x_1, y_0, y_1 be hexadecimal digits with $S_R(x_0 2^4 + x_1) = y_0 2^4 + y_1$, then the cell at the intersection of the x_0^{th} row and the x_1^{th} column contains the values for $y_0||y_1$ in hexadecimal form.

For example $S_R(42) = S_R(0x2A) = 0xE5 = 229$.

x_1 x_0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 1: Rijndael S-Box

8. The S-box S_Q

The S-box S_Q maps 8 bit to 8 bit. Here the input is presented in hexadecimal form.

Let x_0, x_1, y_0, y_1 be hexadecimal digits with $S_Q(x_0 2^4 + x_1) = y_0 2^4 + y_1$, then the cell at the intersection of the x_0^{th} row and the x_1^{th} column contains the values for $y_0||y_1$ in hexadecimal form.

For example $S_Q(42) = S_Q(0x2A) = 0xAC = 172$.

x_1 x_0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	25	24	73	67	D7	AE	5C	30	A4	EE	6E	CB	7D	B5	82	DB
1	E4	8E	48	49	4F	5D	6A	78	70	88	E8	5F	5E	84	65	E2
2	D8	E9	CC	ED	40	2F	11	28	57	D2	AC	E3	4A	15	1B	B9
3	B2	80	85	A6	2E	02	47	29	07	4B	0E	C1	51	AA	89	D4
4	CA	01	46	B3	EF	DD	44	7B	C2	7F	BE	C3	9F	20	4C	64
5	83	A2	68	42	13	B4	41	CD	BA	C6	BB	6D	4D	71	21	F4
6	8D	B0	E5	93	FE	8F	E6	CF	43	45	31	22	37	36	96	FA
7	BC	0F	08	52	1D	55	1A	C5	4E	23	69	7A	92	FF	5B	5A
8	EB	9A	1C	A9	D1	7E	0D	FC	50	8A	B6	62	F5	0A	F8	DC
9	03	3C	0C	39	F1	B8	F3	3D	F2	D5	97	66	81	32	A0	00
A	06	CE	F6	EA	B7	17	F7	8C	79	D6	A7	BF	8B	3F	1F	53
B	63	75	35	2C	60	FD	27	D3	94	A5	7C	A1	05	58	2D	BD
C	D9	C7	AF	6B	54	0B	E0	38	04	C8	9D	E7	14	B1	87	9C
D	DF	6F	F9	DA	2A	C4	59	16	74	91	AB	26	61	76	34	2B
E	AD	99	FB	72	EC	33	12	DE	98	3B	C0	9B	3E	18	10	3A
F	56	E1	77	C9	1E	9E	95	A3	90	19	A8	6C	09	D0	F0	86

Table 2: S-Box S_Q

9. High-level diagrams

Figure 1: Key initialization

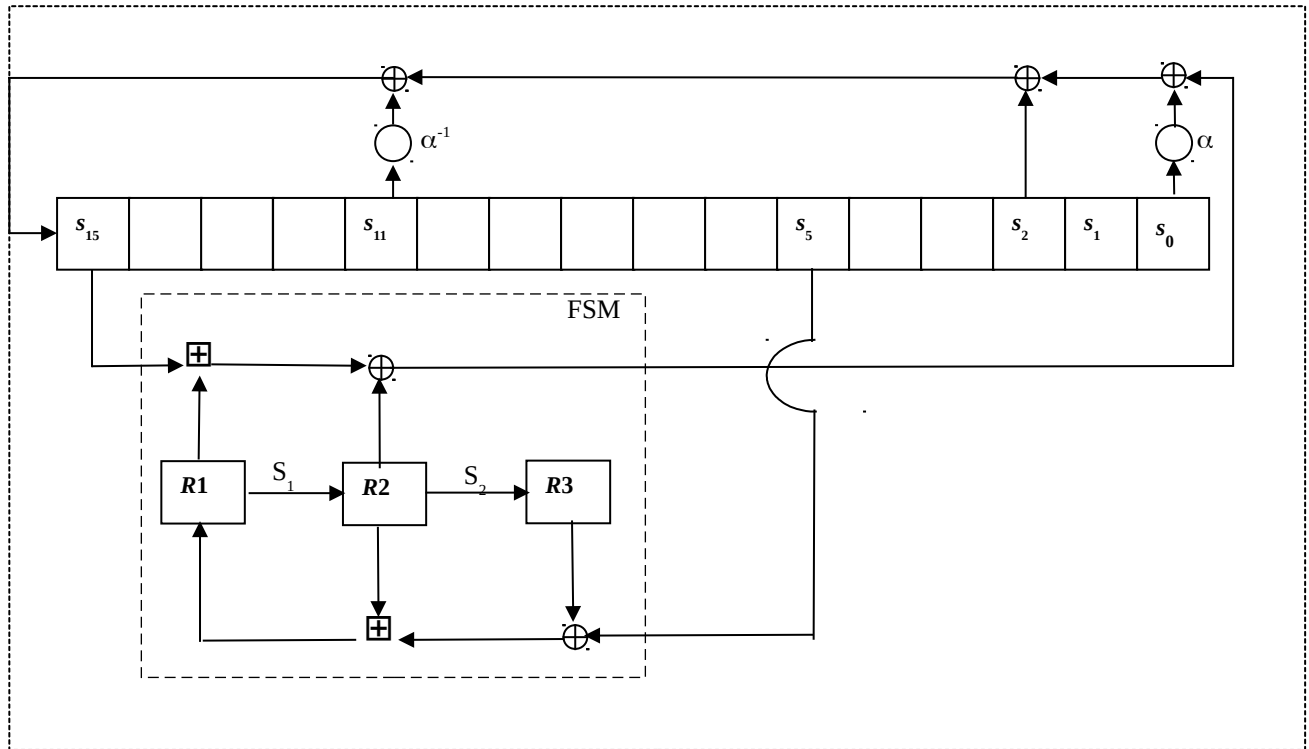
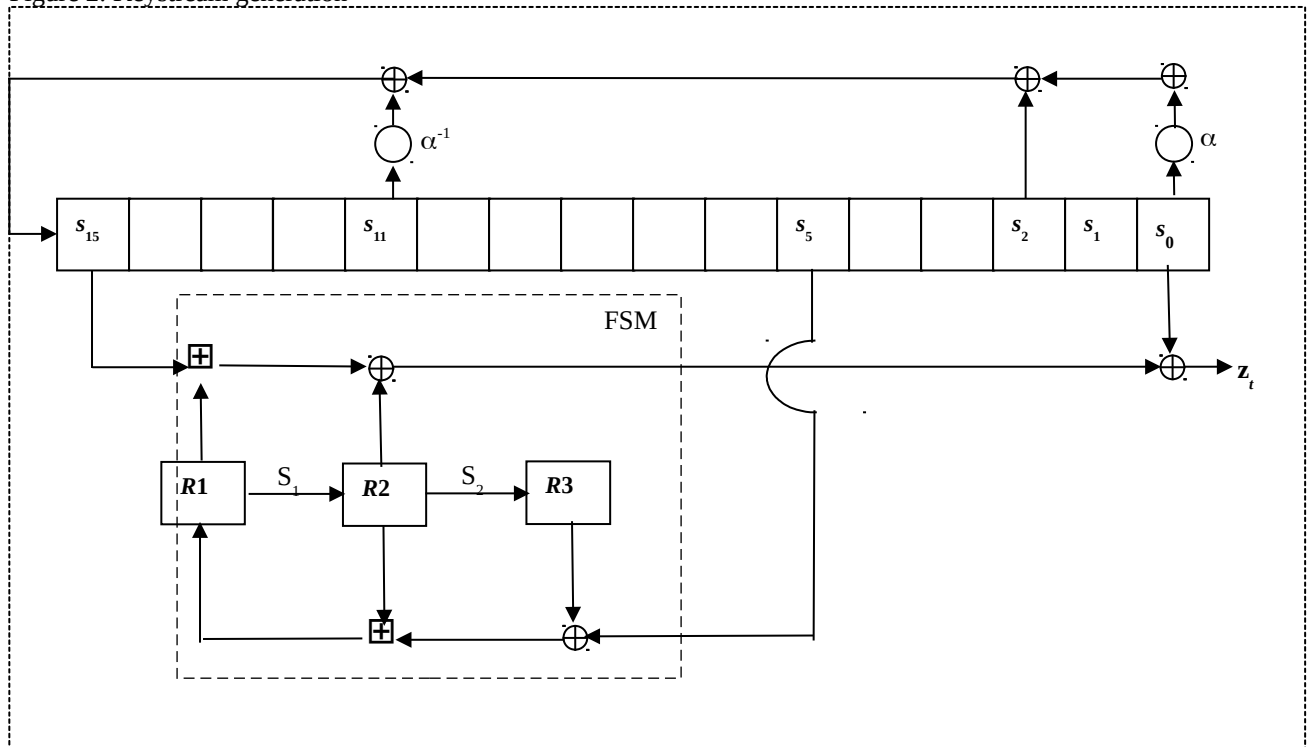


Figure 2: Keystream generation



10. Submission instructions

The requirements for submitting the first project assignment include:

- (1) Each student should independently finish the project. Whenever it is found that a student has copied code OR has his or her code copied by another student, his or her project score will be 0 automatically. Even if a student cannot finish the project completely, the teaching staff will give out the score based on the efforts.
- (2) Please test your code thoroughly with different inputs, and your grades will depend on the number of cases your programs pass successfully.
- (3) You can choose whatever language you are most comfortable with. But please make sure that your code can be compiled and executed easily by the teaching staff. Please include a README file that helps the teaching staff with repeating what you have done.
- (4) For evaluation purpose, two functions will be tested, one for initialization of the cipher (see Section 6.1) and the other generation of keystream (see Section 6.2). To make it easy for grading, please define two functions:
 - `initialization(K, IV)`, where both K and IV are an array of four 32-bit words (i.e., unsigned integers);
 - `generate_keystream(n, KS)`, where n is the number of 32-bit words produced in the keystream, and KS is an array of n 32-bit words that have been produced.