

(a) Formal ISO entry name of the algorithm

{iso standard 9979 fsango (23)}

(b) Proper name of the algorithm given by the promoter or proprietor  
FSAnGo

(c) Intended application range of the algorithm

- (1) Secrecy
- (2) Hash function
- (3) Authentication

(d) Code interface parameters

- (1) Number of affine keys:  $M$
- (2) Number of coefficient bits:  $2z$
- (3) Affine-key coefficient:  $K[i].a, K[i].b \quad (i=0, \dots, M-1)$
- (4) Affine-key counter:  $K[i].c, \quad (i=0, \dots, M-1)$
- (5) Affine-key life:  $K[i].n, \quad (i=0, \dots, M-1)$
- (6) Initial random number:  $x_0$
- (7) Key rewriting procedure:  $w(i, j)$
- (8) Input data:  $m(k), \quad (k=1, 2, \dots)$
- (9) Output data:  $c(k), \quad (k=1, 2, \dots)$

(e) Basic function inspection data

(1) Number of affine keys

$$M = 64$$

(2) Number of affine keys

$$2z = 32$$

(3) Coefficient of affine key (hex)

$K[0].a = 905eb785$	$K[0].b = b2fc569d$
$K[1].a = 46862aab$	$K[1].b = 70a4cb45$
$K[2].a = 81481177$	$K[2].b = dfd1de98$
$K[3].a = 85982471$	$K[3].b = 2641bb80$
$K[4].a = af4b90d9$	$K[4].b = d9018def$
$K[5].a = 35ed3947$	$K[5].b = 65988de5$
$K[6].a = 73f18649$	$K[6].b = 485891b1$
$K[7].a = 150bdacc$	$K[7].b = d316b1f9$
$K[8].a = 317480da$	$K[8].b = 65f09ac5$
$K[9].a = a2438453$	$K[9].b = 95f1a5bb$
$K[10].a = 87111523$	$K[10].b = f81830c4$
$K[11].a = d768eddc$	$K[11].b = 01ab161d$
$K[12].a = c47a664f$	$K[12].b = 7424ebb4$
$K[13].a = 6261d8df$	$K[13].b = 159a22ee$
$K[14].a = 7094ca00$	$K[14].b = 6254a980$
$K[15].a = 41da35a6$	$K[15].b = 170b30ab$
$K[16].a = 37a75c78$	$K[16].b = 69357c8a$
$K[17].a = 4d04eef5$	$K[17].b = 7e7e9349$
$K[18].a = 23133ca1$	$K[18].b = 4a7ddf76$
$K[19].a = 9afc549e$	$K[19].b = da817997$
$K[20].a = 52f34ada$	$K[20].b = aea315b3$
$K[21].a = f0871675$	$K[21].b = 51d86abb$
$K[22].a = 9cb8d602$	$K[22].b = bf4493ce$
$K[23].a = 8ca2d2bf$	$K[23].b = 7ea3dba9$

K[24].a = 6af45d6d	K[24].b = 0aa3d476
K[25].a = 91d15412	K[25].b = 3c36922c
K[26].a = 4d608cbb	K[26].b = b0655c5d
K[27].a = 2d7ec1cc	K[27].b = 12de494e
K[28].a = edf74954	K[28].b = 30850067
K[29].a = 7a6c7ea4	K[29].b = 935f4a29
K[30].a = ffe69e87	K[30].b = 5e81d73c
K[31].a = 397c9dcb	K[31].b = 19950f37
K[32].a = 1626cfd9	K[32].b = 260fba1c
K[33].a = 23de7a80	K[33].b = e39457e7
K[34].a = f1eed528	K[34].b = 61f43661
K[35].a = 12e9491f	K[35].b = 69a5e65b
K[36].a = 8ee0ef4b	K[36].b = 5933505e
K[37].a = 12792d37	K[37].b = 20f25d98
K[38].a = bd3abfcf	K[38].b = 5c16705a
K[39].a = a61d7008	K[39].b = 6cac339a
K[40].a = 55cbeea3	K[40].b = 14cc89cf
K[41].a = 7c9f2835	K[41].b = 2c4734d8
K[42].a = c06daad4	K[42].b = 33a51f73
K[43].a = 0b6f03b3	K[43].b = a4b2856e
K[44].a = 20a41e50	K[44].b = ae2575c8
K[45].a = 25bea0da	K[45].b = a1e8eacd
K[46].a = 21979588	K[46].b = 330b3373
K[47].a = f42e6249	K[47].b = 9d73246b
K[48].a = c84b6a8d	K[48].b = cf18038d
K[49].a = 754448d0	K[49].b = 526225c5
K[50].a = 44db5b01	K[50].b = 92898861
K[51].a = 3553aaab	K[51].b = c1821def
K[52].a = 8c088bc3	K[52].b = 327bbe72
K[53].a = 8521ca39	K[53].b = 47503671
K[54].a = ae92d3f7	K[54].b = 54aec70a
K[55].a = 763144c7	K[55].b = 43c1e5f6
K[56].a = e9806c17	K[56].b = 9adedca7
K[57].a = 2a35e040	K[57].b = 80210edc
K[58].a = c7ee5c6f	K[58].b = a75a7e52
K[59].a = d5f6a884	K[59].b = d88048ba
K[60].a = f1d94395	K[60].b = 616e1d3c
K[61].a = ad9975da	K[61].b = 57b1bca6
K[62].a = 66c7590c	K[62].b = 63df2e8f
K[63].a = 6e9d3a5a	K[63].b = 2695dc9d

(4) number of affine key used(hex) (  $K[i].c = K[i].a \pmod{K[i].n}$  )

K[0].c = 2	K[1].c = 0
K[2].c = 1	K[3].c = 2
K[4].c = 2	K[5].c = 1
K[6].c = 2	K[7].c = 1
K[8].c = 1	K[9].c = 0
K[10].c = 1	K[11].c = 2
K[12].c = 1	K[13].c = 1
K[14].c = 0	K[15].c = 1
K[16].c = 2	K[17].c = 0
K[18].c = 2	K[19].c = 0
K[20].c = 2	K[21].c = 1
K[22].c = 1	K[23].c = 1
K[24].c = 0	K[25].c = 0
K[26].c = 2	K[27].c = 1
K[28].c = 2	K[29].c = 1
K[30].c = 1	K[31].c = 1

K[32].c = 1	K[33].c = 0
K[34].c = 0	K[35].c = 1
K[36].c = 2	K[37].c = 2
K[38].c = 0	K[39].c = 0
K[40].c = 2	K[41].c = 1
K[42].c = 2	K[43].c = 1
K[44].c = 0	K[45].c = 2
K[46].c = 1	K[47].c = 2
K[48].c = 0	K[49].c = 0
K[50].c = 1	K[51].c = 0
K[52].c = 2	K[53].c = 2
K[54].c = 1	K[55].c = 2
K[56].c = 0	K[57].c = 2
K[58].c = 1	K[59].c = 0
K[60].c = 2	K[61].c = 1
K[62].c = 0	K[63].c = 1

(5) Service life of affine key  
 $K[i].n = 3$  ( $i=0, \dots, 15$ )

(6) Initial random number  
 $x(0) = e4b8c094$  (hex)

(7) Key rewriting procedure  $w(i, j)$

$$KWa = ( K[j].a \times K[i].a + K[j].b ) \bmod (2^{32})$$

$$K[i].a = ( KWa \gg 16 ) \text{ xor } KWa \text{ or } 2$$

$$j = (j+1) \bmod M$$

$$KWb = ( K[j].a \times K[i].b + K[j].b ) \bmod (2^{32})$$

$$K[i].b = ( KWb \gg 16 ) \text{ xor } KWb \text{ or } 1$$

(8) Input data (hex):  $m(k)$ , ( $k=1, 2, \dots, 256$ )

0	:	efc9	2f65	79cb	8daf	c660	ae10	a1e1	cd83
8	:	d458	9c3d	b11f	0102	cf32	2d76	e346	c2cb
10	:	356e	fc43	c7af	c0e3	207f	c6aa	4f3a	d3cf
18	:	59b7	423f	fd6f	fc0e	836f	e958	8291	1745
20	:	d8a1	d408	0783	8c84	4e69	252d	4f6f	c6ab
28	:	6c8d	f926	11c8	a80f	6b4c	695d	9c2f	6dcd
30	:	3aff	ac45	9ec0	9eb9	0a6e	67ba	5da4	2b67
38	:	aadd	3849	c4d2	b726	3091	fc4d	791b	b17f
40	:	5c90	f316	2e88	94ef	d9a4	93d0	2f79	a2fd
48	:	3be9	fe10	4349	a882	1a09	e52a	2149	a829
50	:	ac4e	e8d4	55a6	7b7a	3d34	d924	4588	c984
58	:	c460	1f75	cc3d	6579	7f9e	7f00	fce8	fc64
60	:	0de7	753e	db5f	99df	f134	6fe8	bd18	14f1
68	:	6310	483c	ed5e	569a	f706	115f	c411	0eb9
70	:	003c	c96d	c52c	5363	82e8	3cdf	d109	d4d8
78	:	37f9	885c	14df	8111	8c8f	3a2a	fee4	d1a9
80	:	85bd	c44b	aa85	1b1c	bf2a	747c	614e	186a
88	:	71c2	c162	edff	27ee	2074	e569	3213	82eb
90	:	9995	0322	d8d2	1a2c	4745	2e3a	75cb	6a63
98	:	7800	3fa9	3041	688d	33e9	6ee8	1eb0	2eda
a0	:	0c17	5495	11b7	9cf3	cd75	395c	5abc	a021
a8	:	21f8	cf32	8202	fdc9	e591	ccec	1099	a012
b0	:	00c2	c611	8d31	d460	ed81	520c	5e6c	6585
b8	:	1c65	1df6	bb85	e4e6	88c9	872f	a3b2	6d15
c0	:	1839	4742	c486	5305	7a3b	a733	2e41	4878
c8	:	6fb2	7071	f8f4	10b6	26e8	fdf9	e03f	9bb8
d0	:	fb22	b486	c39d	3247	bb52	6403	747e	b8fc

d8 : faa2 58ec 76df c4c0 8e57 f3ad aa8b 00aa  
 e0 : 249a dabc e6b5 59d9 969c a52a a1b5 e5bf  
 e8 : 3766 b2dd aef6 1960 2d73 6502 e058 93eb  
 f0 : 3679 18ae c509 cf15 4290 60fb 171e 520f  
 f8 : 075c 7ea5 c24b 05bb c659 4f69 10ee 9e1f

## (9) Output data (hex): c(k), (k=1, 2, ..., 256)

0 : 763a b61a f6a1 f3ed 6b87 c765 8b4b a79d  
 8 : 7e6f 7e01 cf37 73b4 1aec 7ded fa54 4a95  
 10 : a3e4 9c64 a17f 2110 6279 2acd c9a6 bbe4  
 18 : 2984 fd58 daed c85f a014 daaa c066 8f1d  
 20 : fde2 fdde 8428 e862 f1c6 bce6 6a76 b48c  
 28 : e934 0369 20be e6e8 5b2f ccfe 2c9d 75fc  
 30 : 384e 06a1 81c5 78e6 5ee2 b6a5 728f 777e  
 38 : ca44 587c 788a 217c 609c 52ef d4ff 66eb  
 40 : aaec fa1f 56fe a23e 2cca 571e 03e6 3f3a  
 48 : 1157 0181 ed49 671f 9189 3aa0 4218 9723  
 50 : 26bd ac39 02b5 68f5 d267 d207 b8fe 179d  
 58 : 375a 5abe 00fd 964e bc2e 43c8 33d1 c175  
 60 : 8ca4 864a 5a38 e55c da75 fd7d d9b6 bffd  
 68 : 6782 1da1 3234 1821 d5f0 d4df 05ba 4c51  
 70 : c8e0 34ce 8c08 56d4 6db0 90b2 c1b1 3bf7  
 78 : 8879 8033 ff1a 1e13 affe 8e2f 3aa7 7511  
 80 : 7910 5676 5494 ecf5 7c86 f56b 470f 88a4  
 88 : 3e66 1632 d09c 08d8 2033 9c08 c364 0125  
 90 : f32e 07a8 0f7d 47a1 e845 ff87 bf6d a348  
 98 : 0990 9982 b0d1 ad7a f4a8 3b90 3877 32f7  
 a0 : 4584 98ed 1eda 30a2 2f2b 7f8b 4d0e bc50  
 a8 : b7fd 3c24 2793 d8d3 68e8 1681 5c2e d024  
 b0 : 3d7f 832f 0b2c 5056 e988 bc5a 3885 967f  
 b8 : 21e5 a14b b110 a71d 3d1b 7a36 5859 9c85  
 c0 : c08a 07e8 f89b 2e37 ef00 a83b a956 cb30  
 c8 : 0d51 789e 61f9 f46b 8ae0 f222 45fe bbdb  
 d0 : 7bb5 fb43 87f8 a2f6 88ec c06e ac89 a700  
 d8 : aedf 1547 13c7 e9f5 ef9b a23a fe8c 1158  
 e0 : d08d 9d7b b119 e280 2440 a261 154a 9ff2  
 e8 : 1a37 d6b8 3ed2 1983 54e6 3c59 2f8d b1e8  
 f0 : 560f cf7d e081 03a6 3fc8 4b2a fa42 6686  
 f8 : a256 ea04 9247 c854 6533 537e 26a6 ddf6

## (f) Information of the organization requesting algorithm registration

Organization: Information-technology Promotion Agency, Japan

16F Center Office, Bunkyo Green Court

2-28-8 Hon-Komagome, Bunkyo-Ku, Tokyo 113-6591 JAPAN

TEL:+81-3-5978-7508 FAX:+81-3-5978-7518

Contact : Research & Development Section, FUJISOFT ABC Inc.

1-10-10 Nishi-Kanagawa, Kanagawa-Ku, Yokohama 221-0822

JAPAN

TEL:+81-45-323-5660 FAX:+81-45-323-5657

## (g) Registration and correction dates

9<sup>th</sup> October 2000

## (h) Application of domestic standards

Not applicable

## (i) Restrictions on the exercise of patent rights

1. Patent application No. H11-122866 28-Apr.-1999 (Japan)
2. Patent application No. H11-216997 30-July-1999 (Japan)
3. Patent application No. 2291435 02-Dec.-1999 (Canada)
4. Patent application No. 09/453696 03-Dec.-1999 (U.S.A.)
5. Patent application No. 99309860.7 08-Dec.-1999 (EPC)
6. Patent application No. 99125981.5 10-Dec.-1999 (China)
7. Patent application No. 2000-211743 13-July-2000 (Japan)
8. Patent application No. 2000-225678 26-July-2000 (Japan)

## (j) List of references on all related algorithms

1. Shuichi Suzuki: "Cryptograpy using multi-affine key system", Technical report of IEICE Vol99. No.209, ISEC99-32, (1999-07)
2. Shuichi Suzuki: "The Hash Function Using the multi-affine key system", Technical report of IEICE Vol99. No.584, ISEC99-79, (2000-01)
3. Shuichi Suzuki: "A digital signature and the notion of master key using the multi-affine key system", SCIS2000-N01 (2000-01)

## (k) Algorithm-related description

"FSAnGo" is a stream encryption system of the secret key type.

The main part of this system is an algorithm that can produce series of random numbers with cycles long enough.

## Procedure of "FSAnGo"

1. Several affine keys are prepared.
2. A multi-affine key system (the interaction between a multi-affine key) is defined between the affine keys.
3. An affine key that output n random numbers in the process of encrypting is replaced with a randomly-selected key.

## Affine key K

An affine key is a structure consisting of four integers ( $K=\{a, b, c, n\}$ ). The effect of K on x of the integer (or origin of the finite circle) is defined as follows:

$$K(x) = a x + b$$

"c" indicates how many time the affine key has been used and "n" indicates how many times the affine key can be used (life of the affine key).

## Multi-affine key system

The multi-affine key system consists of several affine keys  $\{K[i]\}$  and a single key change procedure  $w(i, j)$ .  $w(i, j)$  means to rewrite  $K[i]$  using  $K[j]$ .  $K[i].a$  and  $K[i].b$  can be rewritten with  $K[j]$  in various ways.

## Example of algorithm

Here is an example using a 32-bit integer ( $2z=32$ ). When the number of multi-affine keys M is 64 and the key life n is 3, prescribe the affine key coefficients a and b and the initial random number  $x_0$ . This is a random-number sequence of 516 bytes. If the plain statement is  $\{m[k]\}$  and the encrypted output statement is  $\{C[k]\}$ , the following algorithm can be used for encryption.

1.  $i = (x(0) \text{ shr } z) \text{ mod } M, k = 1, vkey = 0$
2.  $x(k) = K[i].a * x(k-1) + K[i].b$
3.  $K[i].c = K[i].c + 1$
3.  $c[k] = m[k] \text{ xor } (x(k) \text{ and } (2^z - 1)) \dots(\text{ciphering})$

4.  $j = ( (x(k) \text{ shr } z) + vkey ) \bmod M$
5. if  $i = j$  then  $j = j+1 \bmod M$
6. if  $K[i].c \geq K[i].n$  then  $w(i,j)$ ,  $K[i].c = 0$
7.  $i = j$ ,  $k = k+1$ ,  $vkey = vkey+1$
8. goto 2.

(l) Application mode  
None

(m) Other information

Experimental confirmation of safety

With the 32-bit integer, the followings could be confirmed experimentally:

- (1) Three-word(16bits) random numbers  $r_1$ ,  $r_2$ , and  $r_3$  are determined arbitrarily.
- (2) One of the six words for the initial value  $x(0)$ ,  $K[i].a$ ,  $K[i].b$  is determined arbitrarily.
- (3) If the remaining five words are determined arbitrarily, an affine key configuration can be created for the random numbers  $r_1$ ,  $r_2$ , and  $r_3$  in (1).

From these facts, the followings can be surmised:

- If the same keys are used three times forcibly and the  $K$  value is replaced repeatedly ( $K_0$  with  $K_1$ ,  $K_1$  with  $K_2$ , and so on) with an infinite number of keys, the encrypted statement cannot be decrypted.
- If the number of keys is limited in this model, the encrypted statement might be decrypted only when the same key is used again.
- This encryption system makes it impossible to identify a key that replaced a specific key.

Under these circumstances, checking all available affine keys is the only method for decryption. If 64 multi-affine keys are used, the total number of combinations will exceed the 600th power of 10.

It has been experimentally confirmed that the cycle of random numbers generated in this way is not shorter than 260th power of 10 at the worst. This cycle is practically long enough.

If 1024/16384 bits of a random number is processed by Fourier transform to see whether the power spectra become zero, the followings become clear: None of the power spectra of random numbers generated using multi-affine keys became zero. Even the smallest power spectrum was not less than the  $-10$ th power of 10.

When a random-number sequence generated from repetitions of the same random number was processed by Fourier transform, a random number with the power spectrum of zero appeared cyclically. The power spectrum was about the  $-31$ st power of 10.

In other words, random numbers generated using multi-affine keys always keep the greatest linear complicacy. This result in a conclusion that random numbers generated using multi-affine keys cannot be realized with a linear feedback register.

Encryption speed

A 16bit random number can be generated almost by a single integral multiplication. Therefore, the encryption speed is very high. With a Pentium III processor (600MHz), the encryption and decryption speed is about 573Mbps.