Reprinted from *Fast Software Encryption* (Ed. R. Anderson), Lecture Notes in Computer Science No. 809. New York: Springer, 1994, pp. 1-17.

SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm

James L. Massey
Signal and Information Processing Laboratory
Swiss Federal Institute of Technology
CH-8092 Zürich

Abstract: A new non-proprietary secret-key block-enciphering algorithm, SAFER K-64 (for **S**ecure **A**nd **F**ast **E**ncryption **R**outine with a **K**ey of length **64** bits) is described. The blocklength is 64 bits (8 bytes) and only byte operations are used in the processes of encryption and decryption. New cryptographic features in SAFER K-64 include the use of an unorthodox linear transform, called the Pseudo-Hadamard Transform, to achieve the desired "diffusion" of small changes in the plaintext or the key over the resulting ciphertext and the use of additive key biases to eliminate the possibility of "weak keys". The design principles of K-64 are explained and a program is given, together with examples, to define the encryption algorithm precisely.

1. Introduction

This paper describes a new block encryption algorithm called **SAFER K-64** (for **S**ecure **A**nd **F**ast **E**ncryption **R**outine with a **K**ey of length **64** bits) that the author recently developed for Cylink Corporation (Sunnyvale, CA, USA) as a non-proprietary cipher. SAFER K-64 is a byte-oriented block enciphering algorithm. The block length is 8 bytes (64 bits) for plaintext and ciphertext; the user-selected key is also 8 bytes (64 bits) in length. SAFER K-64 is an *interated* cipher in the sense that encryption is performed by applying the same transformation repeatedly for r rounds, then applying an output transformation; r = 6 is recommended but larger values of r can be used if desired for even greater security. Each round uses two 8-byte (64-bit) subkeys determined by a key schedule from the secret 8-byte user-selected key. The output transformation uses another 8-byte subkey determined by the key schedule. One unusual feature of SAFER K-64 is that, in contrast to most recently proposed iterated block ciphers, encryption and decryption are slightly different (i.e., they differ by more than just the reversal of the key schedule).

SAFER K-64 uses only byte operations in the processes of encryption and decryption, which makes it particularly useful in applications such as smart cards where very limited processing power is available. Some bit-level rotations of bytes are used in the key schedule, but this is done "once and for all", i.e., until the user-selected key is changed. To achieve security with such simple processing, SAFER K-64 exploits two new cryptographic concepts, namely:
(1) an **unorthodox linear transform**, which we call the *Pseudo-Hadamard Transform (PHT)*, that allows the cipher rapidly to achieve the desired "diffusion" of small changes in the plaintext or the key over the resulting ciphertext [It is usually the case in block cipher design that one struggles to obtain such diffusion by carefully selecting permutations to imbed within the cipher and then doing massive statistical testing to see which ones give acceptable diffusion. As will be seen, the PHT provides a systematic way to ensure that the cipher provides the necessary diffisuion--in fact, the diffusion provided by the PHT appears to be better than that in any other cipher that we know.]
and (2) the use of additive **key biases** that eliminate the "weak keys" that plague most block ciphers. [SAFER K-64 includes a recursive procedure for generating these key biases that is easy to implement and that provides the very "random" biases desired.]

2. Description of the SAFER K-64 Algorithm

The encrypting structure of the SAFER K-64 cipher is shown in Fig. 1. The enciphering algorithm consists of r rounds of identical transformations that are applied in sequence to the plaintext, followed by an output transformation, to produce the final ciphertext. Our recommendation is to use $r = 6$ for most applications, but up to 10 rounds can be used if desired. Each round is controlled by two 8-byte subkeys and the output transformation is controlled by one 8-byte subkey. These $2r + 1$ subkeys are all derived from the 8-byte user-selected subkey K1 in a manner that will be explained later. The plaintext, ciphertext and all subkeys are 8 bytes (64 bits) long.

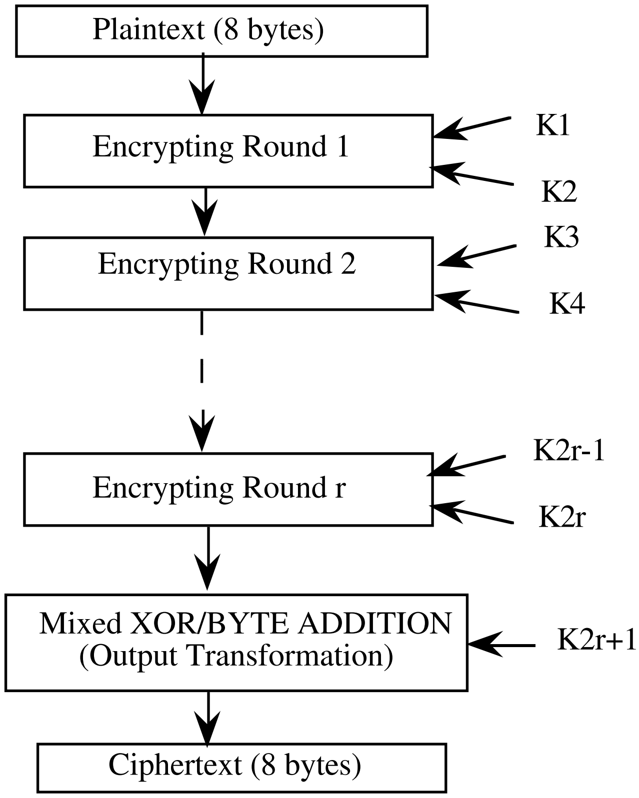

Fig. 1: Encrypting Structure of SAFER K-64

The *output transformation* of SAFER K-64 consists of the bit-by-bit XOR ("exclusive or" or modulo-2 sum) of bytes 1, 4, 5 and 8 of the last subkey, K2r+1, with the corresponding bytes of the output from the r-th round together with the byte-by-byte byte addition (modulo-256 addition) of bytes 2, 3, 6 and 7 of the last subkey, K2r+1, to the corresponding bytes of the output from the r-th round. [Higher order bytes are considered to be those on the left, i.e., byte 1 is the most significant byte--this convention is used throughout this paper.] Hereafter, we refer to this particular combination of two eight-byte words as the *Mixed XOR/Byte-Addition operation*.

The detailed *encryption round structure* of SAFER K-64 is shown in Fig. 2. The first step within the i^{th} round is the Mixed XOR/Byte-Addition of the round input with the subkey K2i-1. The eight bytes of the result are then passed through a *nonlinear layer* and individually subjected to one of two different "highly nonlinear" transformations, namely:
(1) the operation labelled "$45^{(\cdot)}$" in Fig. 2, which notation is to suggest that if the byte input is the integer j then the byte output is 45^j modulo 257 (*except that this output is taken to be 0 if the modular result is 256, which occurs for j = 128*) [The reasoning behind the use of this transformation is the following. Because 257 is a prime, arithmetic

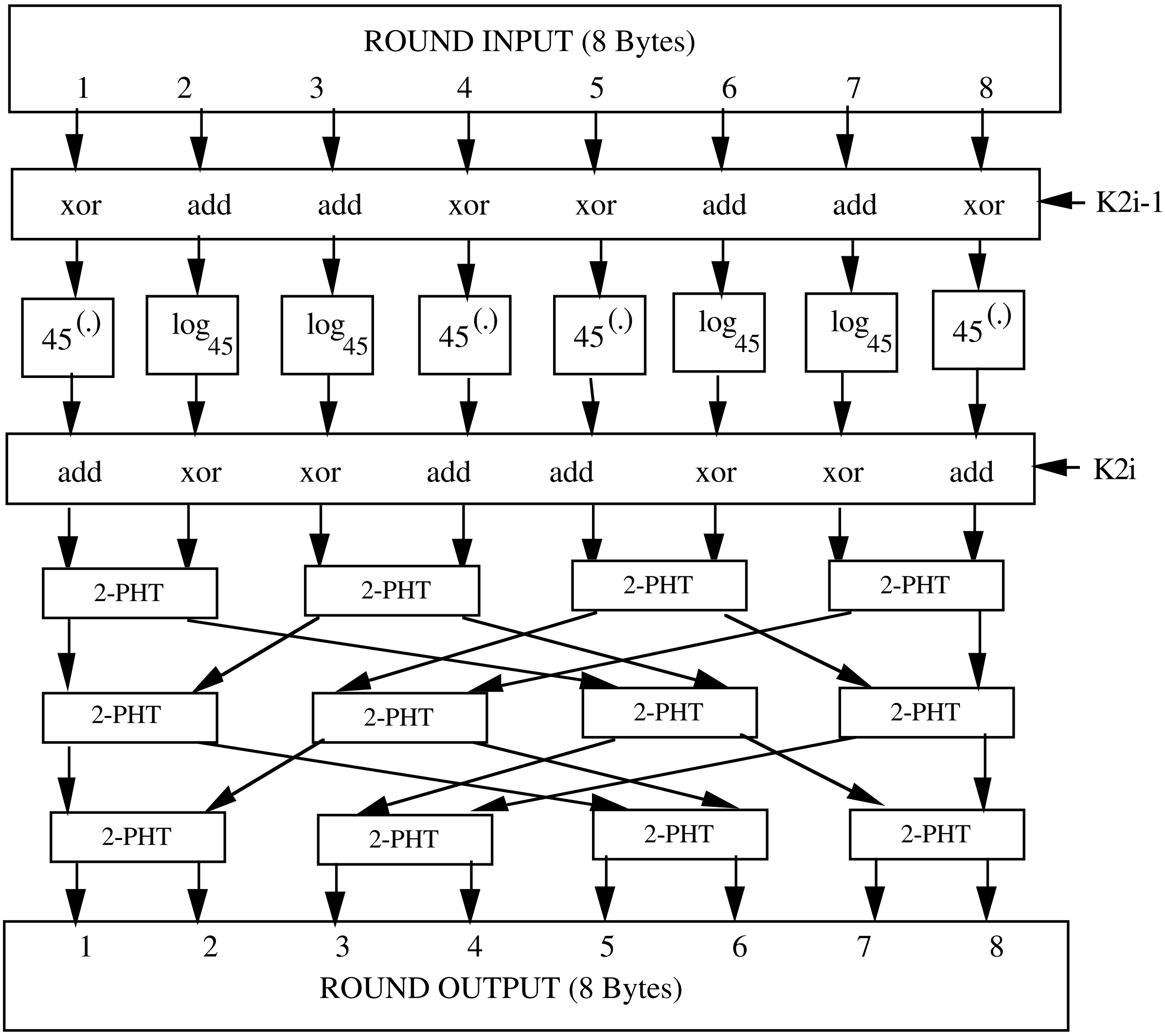

Fig. 2: Encryption round structure of SAFER K-64

modulo 257 is the arithmetic of the finite field GF(257). The element 45 is a primitive element of this field, i.e., its first 256 powers generate all 256 non-zero field elements. Thus the mapping "$45^{(\cdot)}$" is an invertible mapping from one byte to one byte that is very nonlinear with respect to the arithmetic of GF(257) as well as with respect to the vector space of 8-tuples over the binary field GF(2) whose addition is bit-by-bit XOR.]
and (2) the operation labelled "$\log_{45}$" in Fig. 2, which notation is to suggest that if the byte is the integer j then the byte output is $\log_{45}(j)$ (*except that this output is taken to be 128 if the input is j = 0*), i.e., the power to which one must raise 45 to obtain j modulo 257. [The nonlinear features of this mapping are similar to those described for exponentiation.]

In the appended programs for implementing the SAFER K-64 cipher, these two nonlinear operations are realized with two look-up tables of 256 bytes each, i.e., simple byte-in byte-out look-up tables.

The output of the eight nonlinear transformations is then combined with subkey K2i in an operation that consists of the byte-by-byte byte addition (modulo-256 addition) of bytes 1, 4, 5 and 8 of the subkey K2i to the corresponding bytes of the output from the nonlinear transformations together with the bit-by-bit XOR (modulo-2 sum) of bytes 2, 3, 6 and 7 of the subkey K2i to the corresponding bytes of the output from the nonlinear transformations. Hereafter, we refer to this particular combination of two eight-byte words as the *Mixed Byte-Addition/XOR operation*. [It is important to note the distinction between this Mixed XOR/Byte-Addition operation and the previously described Mixed Byte-Addition/XOR operation.]

The output of the Mixed Byte-Addition/XOR operation then passes through a *three-level "linear layer"* of boxes that are labelled "2-PHT" in Fig. 2. This notation indicates a 2-point PHT. If the two input bytes to a 2-PHT are (a_1, a_2), where a_1 is the more significant byte, then the two output bytes are (b_1, b_2) where

$$\begin{aligned} b_1 &= 2a_1 + a_2 \\ b_2 &= a_1 + a_2 \end{aligned} \qquad (1)$$

and where the arithmetic is normal byte arithmetic, i.e., arithmetic modulo 256. Between levels of the linear layer, the decimation-by-2 permutation [familiar from the Cooley-Tukey FFT and the ordinary discrete Hadamard Transform] is applied--as will be seen, this is what creates diffusion in the SAFER K-64 cipher. The output of this linear layer constitutes the round output.

3. Decryption for SAFER K-64

The decrypting structure of SAFER K-64 is shown in Fig. 3. The deciphering algorithm consists of an *input transformation* that is applied to the ciphertext block,

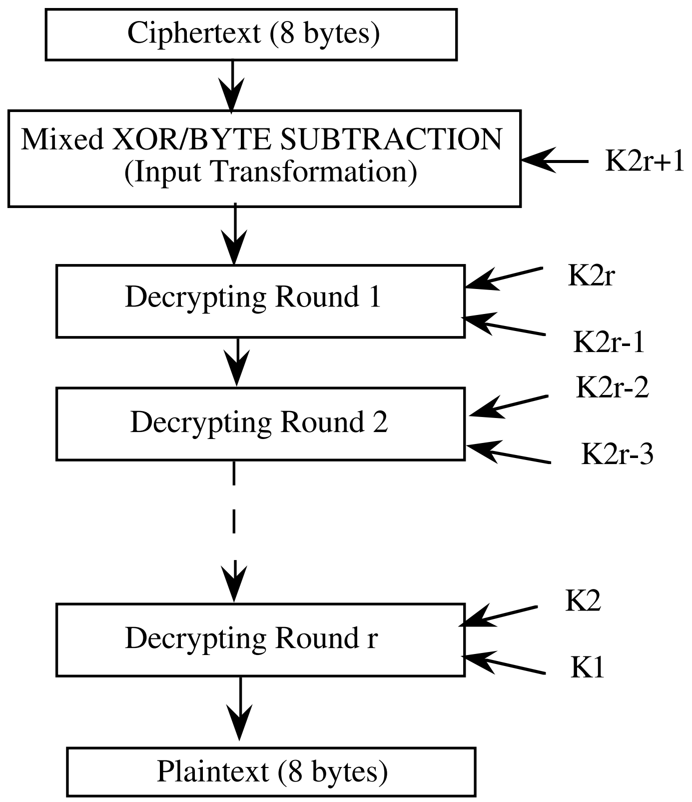

Fig. 3: Decrypting Structure of SAFER K-64

followed by r rounds of identical transformations. The input transformation consists of the Mixed XOR/Byte-Subtraction of subkey K2r+1 from the ciphertext block. A characterizing feature of SAFER K-64 is that decrypting rounds differ from encrypting rounds so that an encrypter cannot be converted to a decrypter by simply reversing the key schedule.

The detailed *decryption round structure* of SAFER K-64 is shown in Fig. 4. The i^{th} encryption round begins by passing the round input through the *three-level inverse linear layer*. It is easy to check from equations (1) that the Inverse PHT (IPHT) is given by

$$a_1 = b_1 - b_2$$
$$a_2 = -b_1 + 2b_2. \qquad (2)$$

This IPHT is just as simple to compute as the direct PHT. The fan-out-by-two permutation between levels of this inverse linear layer is the inverse of the decimate-by-two permutation used in the linear layer of an encryption round.

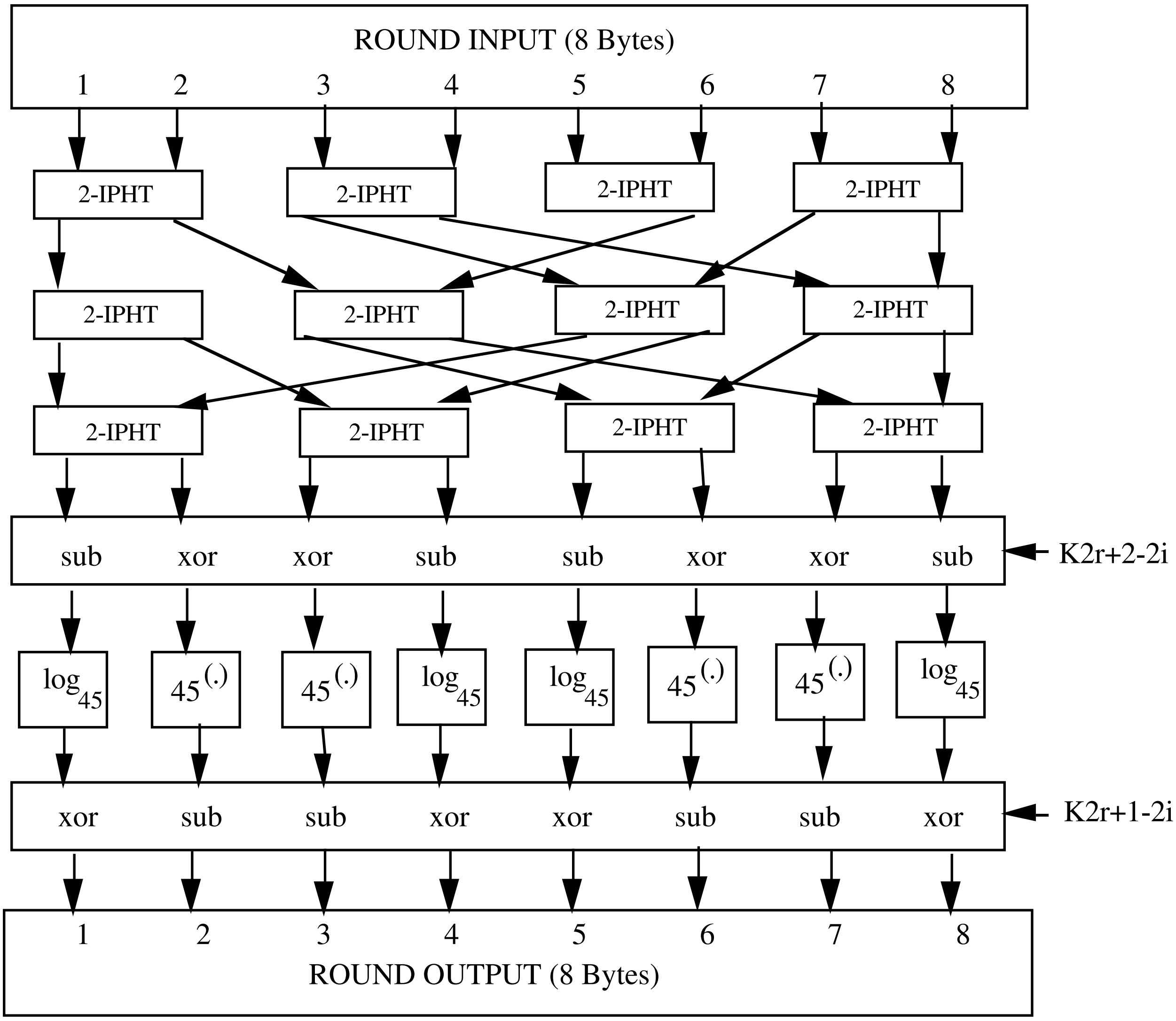

Fig. 4: Decryption round structure of SAFER K-64

The next step within the i^{th} decryption round is the *Mixed Byte-Subtraction/XOR* of the output of the inverse linear layer with the subkey K2r+2-2i, which consists of the byte-by-byte byte subtraction (modulo-256 subtraction) of bytes 1, 4, 5 and 8 of the subkey K2r+2-2i from the corresponding bytes of the output from the previous round together with the bit-by-bit XOR (modulo-2 sum) of bytes 2, 3, 6 and 7 of the subkey K2r+2-2i with the corresponding bytes of the output from the previous round.

In the next step of the decryption round, the eight bytes from the previous step are passed through the "*inverse nonlinear layer*", which differs from the "nonlinear layer" in the encryption round by interchanging of the locations of the four exponentiating boxes and the four logarithm-taking boxes.

The last step within the i^{th} decryption round is the *Mixed XOR/Byte-Subtraction* of the round input with the subkey K2r+1-2i, which consists of the bit-by-bit XOR (modulo-2 sum) of bytes 1, 4, 5 and 8 of the subkey K2r+1-2i with the corresponding bytes of the output from the previous round together with the byte-by-byte byte subtraction (modulo-256 subtraction) of bytes 2, 3, 6 and 7 of the subkey K2r+1-2i from the corresponding bytes of the output from the previous round.

4. How SAFER K-64 Works and Why

To see that the SAFER K-64 cipher correctly decrypts, we first note that the Mixed XOR/Byte-Subtraction of K2r+1 in the Input Transformation for decryption (cf. Fig. 3) undoes the Mixed XOR/Byte-Additon of K2r+1 in the Output Transformation for encryption (cf. Fig. 1). Then the inverse linear layer of the first decryption round (cf. Fig. 4) undoes the transformation performed by the linear layer in the last encryption round (cf. Fig. 2). Next, the Mixed Byte-Subtraction/XOR of K2r in the first decryption round (cf. Fig. 4) undoes the Mixed Byte-Addition/XOR of K2r in the last encryption round (cf Fig. 2). Then the inverse nonlinear layer in the first decryption round (cf. Fig. 4) undoes the transformation performed by the nonlinear layer in the last encryption round (cf. Fig. 2). Finally, the Mixed XOR/Byte-Subtraction of K2r-1 in the first decryption round (cf. Fig. 4) undoes the Mixed XOR/Byte-Addition of K2r-1 in the last encryption round (cf. Fig. 2). In the same way, decryption round i undoes the transformation performed by encryption round r + 1 - i for i = 2, 3, ... , r so that decryption indeed recovers the original plaintext.

SAFER K-64 was designed in accordance with Shannon's principles of confusion and diffusion for obtaining security in secret-key ciphers [1]. When a round subkey is not all-zero in SAFER K-64 encryption, its combination by Mixed XOR/Byte Addition (or Mixed Byte-Addition/XOR) with the signal within the round acts like a nonlinear combination with respect to the subsequent transformations in the nonlinear layer and in the linear layer. This gives the cipher the *confusion* required to make the statistics of the ciphertext depend in a complicated way on the statistics of the plaintext--provided that small changes diffuse quickly through the cipher. To guarantee this *diffusion* in SAFER K-64 is, in fact, why we developed a new and unorthodox linear transform, the Pseudo-Hadamard Transform (PHT).

The standard Hadamard Transform (HT) [sometimes called the "Walsh transform" or the "Walsh-Hadamard transform"] has in place of (1) the equations

$$\begin{aligned} b_1 &= a_1 + a_2 \\ b_2 &= a_1 - a_2. \end{aligned} \tag{3}$$

Notice that the determinant of the matrix of coefficients is -2, which makes these equations non-invertible for byte arithmetic (arithmetic modulo 256) where -2 = 254 has no multiplicative inverse. It also has the unpleasant effect of requiring a multiplication by 1/2 in the inverse transform in those number systems where 2 has a multiplicative inverse. By choosing equations (1), whose matrix of coefficients has determinant 1, we avoid both of these problems--we can use normal byte arithmetic and there is no unpleasant scale factor in the inverse transform! Moreover, we can still mimic the HT in the multi-dimensional case, which is what the decimations by-two and fanning-outs by-two accomplish. We are in fact using a three-dimensional PHT, i.e., independent 2-PHTs in each of 3 dimensions, which is why there are $2^3 = 8$ bytes in the input and output of the PHT within SAFER K-64.

Just as for the HT in number systems appropriate to it, every digit (here read "byte") of the input to the PHT effects every output byte, i.e., the PHT provides guaranteed complete diffusion within one linear layer. In Appendix A, we show the PHT for the unit-vector inputs where one sees this diffusion over all eight output bytes very clearly. By linearity, the PHT of any vector can be computed as the corresponding linear combination of these unit-vector PHT's. The "guaranteed complete diffusion" within one layer does not hold fully when one considers single-bit changes in the input bytes. Because of the factor of 2 in equations (1), a few bits of the input will effect only 4 bytes (or 2 bytes or 1 byte) of the output within one linear layer, but their effect is immediately spread over all 8 bytes in the next linear layer encountered. This can be seen from the last three examples in Appendix A. For instance, because (1,0,0,0,0,0,0,0) has the PHT (8,4,4,2,4,2,2,1), it follows [from the fact that 2 * 128 = 0 mod 256] that (128,0,0,0,0,0,0,0), which contains a single non-zero bit, will have the PHT (0,0,0,0,0,0,0,128), which shows no diffusion at all. However, in turn (0,0,0,0,0,0,0,128) has the PHT (128,128,128,128,128,128,128,128), which shows complete diffusion over output bytes. In fact, consideration of the unit-vector PHT's in Appendix A shows that (128,0,0,0,0,0,0,0) is the only vector that shows no diffusion under one application of the PHT. We know of no other cipher with such rapid and guaranteed diffusion. This rapid diffusion is the main reason that r = 6 rounds of encipherment are enough to make SAFER K-64 crack-resistant.

5. The Key Schedule for SAFER K-64

The *key schedule* for SAFER K-64, i.e., the procedure for generating the subkeys K2, K3, ... , K2r+1 from the user-selected subkey K1, is indicated in Fig. 5. The quantities B2, B3, ... , B2r-1 are the *key biases* that have the purpose of ensuring that the round subkeys appear individually "random" and, in particular, that no more than one round subkey can be all-zero. Letting b[i,j] denote the j-th byte of bias Bi, we can express this byte as the double exponential

$$b[i,j] \;=\; 45^{**}[45^{**}(9i+j) \bmod 257] \bmod 257, \qquad (4)$$

which equation defines the key biases used in SAFER K-64. We note here that we might have used the factor 8 instead of 9 in the exponent in (4)--we chose 9 to introduce an extra measure of "staccato" in the key schedule." A Table giving the precise values of the key biases for SAFER K-64 is given in Appendix B. Examination of the Table in Appendix B shows that the resulting sequence of biases is indeed very random appearing, which is all that is really needed. The use of such biases, which appears to be new, is clearly a good idea in general for iterated ciphers. The "weak keys" (also called "self-dual keys" and "keys with a dual") of the Data Encryption Standard (DES) [2] are a direct result of the fact that no key biases are used so that, for instance, all 16 round subkeys in DES can be all-zero.]

Fig. 5 shows how K1, the user-selected 64-bit subkey, is used to generate the additional 64-bit subkeys K2, K3, ... , K2r+1 that are required within the r-round SAFER K-64 algorithm. Note that, in the generation process, the subkey register is byte-wise rotated by 3-bits to the left between additions of a new bias. [The addition of a bias is always byte-by-byte byte addition (modulo-256 addition).] Ideally, one wishes the entire subkey sequence K1, K2, K3, ... , K2r+1 to have the character of a sequence of independently-chosen uniformly-random subkeys. Of course, this cannot be achieved in a strict sense because all of the subkeys in this sequence are determined entirely by the first (user-selected) subkey, K1. The real goal in the design of the key schedule is to make the departure from independence so complicated that it cannot be exploited by an attacker--and this is the purpose of both the byte rotations and the addition of subkey biases within the key schedule for SAFER K-64

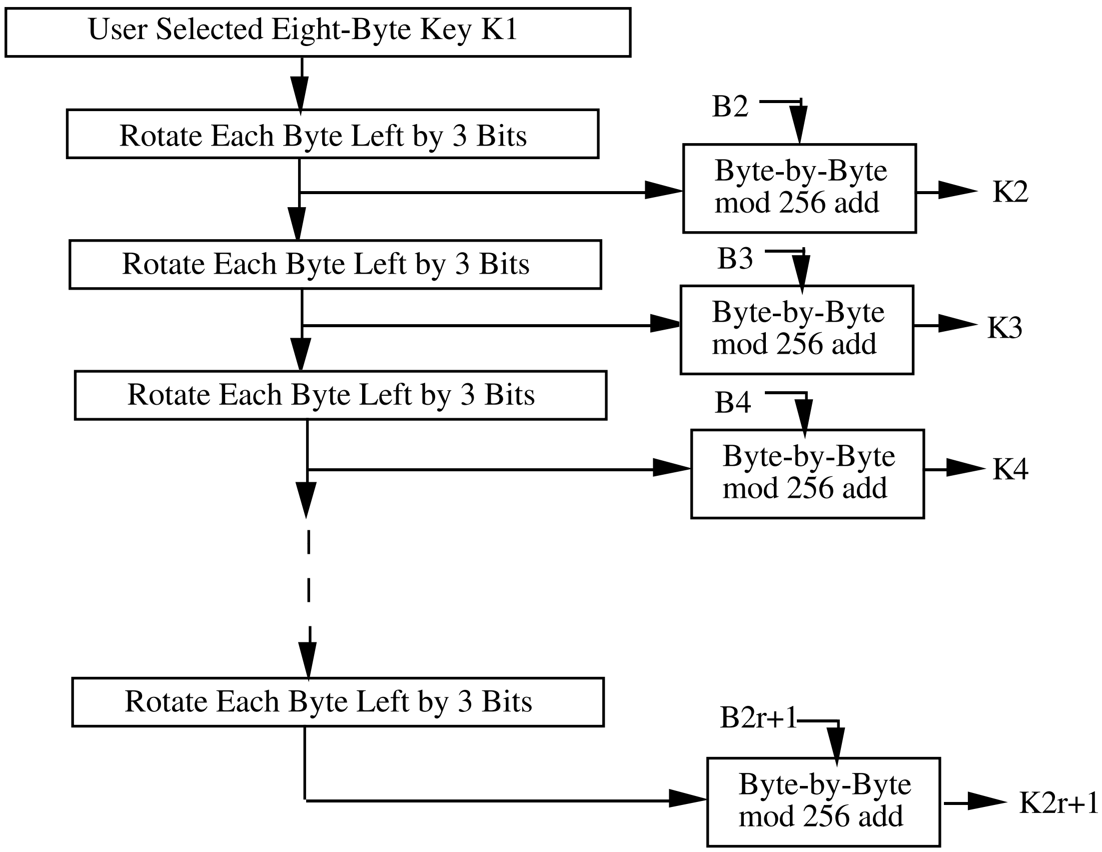

Fig. 5: Key Schedule for SAFER K-64

6. SAFER K-64 Program and Examples

Appendix C gives a TURBO PASCAL program that implements the full r-round SAFER K-64 cipher, both for encryption and decryption.. This program should be taken as the definition of the SAFER K-64 enciphering algorithm. Appendix C also gives examples of $r = 6$ round encryption (the recommended number of rounds) for use in checking implementations of SAFER K-64.

7. Security Considerations for SAFER K-64

In Section 4, we indicated how SAFER K-64 achieves both good diffusion and good confusion, the two basic features that contribute to the security of a block cipher. The best measure of security available today for an iterated block cipher is its resistance to attack by differential cryptanalysis [3]. It is easy to show that, for the appropriate definition of difference between a pair of plaintext blocks (or a pair of ciphertext blocks), SAFER K-64 is a *Markov cipher* [4], a fact that greatly simplifies its analysis for resistance to differential cryptanalysis. Cylink Corporation has contracted for such an analysis of SAFER K-64 by a group of cryptanalysts that does not include the designer of the algorithm. A considerable effort has been invested in this effort, whose conclusion is that six-round SAFER K-64 appears to be secure against differential cryptanalysis. This group of cryptanalysts has also done extensive statistical testing of SAFER K-64 with no detection of any weakness. The evidence available today suggests that SAFER K-64 is a strong cipher whose strength is well measured by the length (64 bits) of its user-selected key.

References

[1] C.E. Shannon, "Communication Theory of Secrecy Systems", *Bell System Tech. J.*, vol. 28, pp. 656-715, Oct., 1949.

[2] U.S. Department of Commerce/National Bureau of Standards, FIPS Pub 46, *Data Encryption Standard*, April 1977.

[3] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard.* New York: Springer-Verlag, 1993.

[4] X. Lai, J. L. Massey and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," pp. 17-38 in *Advances in Cryptology - EUROCRYPT '91* (Ed. D. W. Davies), Lecture Notes in Computer Science No. 547. Heidelberg and New York: Springer-Verlag, 1991

APPENDIX A:
Examples of the Pseudo-Hadamard Transform (PHT)

```
INPUT  VECTOR is    0   0   0   0   0   0   0   1
OUTPUT VECTOR is    1   1   1   1   1   1   1   1

INPUT  VECTOR is    0   0   0   0   0   0   1   0
OUTPUT VECTOR is    2   2   1   1   2   2   1   1

INPUT  VECTOR is    0   0   0   0   0   1   0   0
OUTPUT VECTOR is    2   2   2   2   1   1   1   1

INPUT  VECTOR is    0   0   0   0   1   0   0   0
OUTPUT VECTOR is    4   4   2   2   2   2   1   1

INPUT  VECTOR is    0   0   0   1   0   0   0   0
OUTPUT VECTOR is    2   1   2   1   2   1   2   1

INPUT  VECTOR is    0   0   1   0   0   0   0   0
OUTPUT VECTOR is    4   2   2   1   4   2   2   1

INPUT  VECTOR is    0   1   0   0   0   0   0   0
OUTPUT VECTOR is    4   2   4   2   2   1   2   1

INPUT  VECTOR is    1   0   0   0   0   0   0   0
OUTPUT VECTOR is    8   4   4   2   4   2   2   1

INPUT  VECTOR is  128   0   0   0   0   0   0   0
OUTPUT VECTOR is    0   0   0   0   0   0   0 128

INPUT  VECTOR is    0   0   0   0   0   0   0 128
OUTPUT VECTOR is  128 128 128 128 128 128 128 128
```

APPENDIX B:

Table of Key Biases for SAFER K-64 Cipher.

(Biases B2 to B21 are listed here although only B2 to B13 are required when r = 6 rounds are used with SAFER K-64.)

```
Bias  B2 is   22 115  59  30 142 112 189 134

Bias  B3 is   71 126  36  86 241 119 136  70

Bias  B4 is  177 186 163 183  16  10 197  55

Bias  B5 is  201  90  40 172 100 165 236 171

Bias  B6 is  198 103 149  88  13 248 154 246

Bias  B7 is  102 220   5  61 211 138 195 216

Bias  B8 is  106 233  54  73  67 191 235 212

Bias  B9 is  155 104 160 101  93  87 146  31

Bias B10 is  113  92 187  34 193 190 123 188

Bias B11 is   99 148  95  42  97 184  52  50

Bias B12 is  253 251  23  64 230  81  29  65

Bias B13 is  143  41 221   4 128 222 231  49

Bias B14 is  127   1 162 247  57 218 111  35

Bias B15 is  254  58 208  28 209  48  62  18

Bias B16 is  205  15 224 168 175 130  89  44

Bias B17 is  125 173 178 239 194 135 206 117

Bias B18 is   19   2 144  79  46 114  51 133

Bias B19 is  141 207 169 129 226 196  39  47

Bias B20 is  122 159  82 225  21  56  43 252

Bias B21 is   66 199   8 228   9  85  94 140
```

APPENDIX C:
Examples of Six-Round SAFER K-64 Encryption and Program for Implementation

PLAINTEXT is	1	2	3	4	5	6	7	8
The KEY is	0	0	0	0	0	0	0	0
after round 1	0	46	170	144	255	118	2	238
after round 2	35	175	193	103	246	87	43	202
after round 3	64	252	4	38	1	140	36	104
after round 4	2	62	127	41	25	97	179	196
after round 5	59	221	9	152	113	50	224	52
after round 6	242	255	38	130	179	219	71	133
CRYPTOGRAM is	125	40	3	134	51	185	46	180

PLAINTEXT is	0	0	0	0	0	0	0	0
The KEY is	1	2	3	4	5	6	7	8
after round 1	240	174	18	192	79	214	2	46
after round 2	51	154	197	181	138	198	236	83
after round 3	178	36	41	77	26	13	222	86
after round 4	111	39	188	122	73	216	30	100
after round 5	132	78	244	157	225	84	106	144
after round 6	197	105	114	54	196	101	227	80
CRYPTOGRAM is	90	178	127	114	20	163	58	225

PLAINTEXT is	1	2	3	4	5	6	7	8
The KEY is	8	7	6	5	4	3	2	1
after round 1	101	42	122	106	63	111	225	227
after round 2	102	122	66	171	75	196	228	30
after round 3	114	219	165	207	71	24	132	155
after round 4	117	53	164	99	161	204	201	48
after round 5	132	77	246	149	5	187	182	27
after round 6	199	89	95	137	71	106	55	152
CRYPTOGRAM is	200	242	156	221	135	120	62	217

PLAINTEXT is	0	0	0	0	0	0	0	0
The KEY is	0	0	0	0	0	0	0	0
after round 1	203	244	158	176	123	197	11	39
after round 2	27	47	1	53	133	49	233	187
after round 3	134	147	160	151	93	5	125	185
after round 4	190	249	153	140	109	203	139	58
after round 5	143	72	176	126	51	175	84	69
after round 6	140	255	43	205	142	9	196	78
CRYPTOGRAM is	3	40	8	201	14	231	171	127

PROGRAM Full_r_Rounds_max_10_of_SAFERK64_cipher;

```
VAR a1, a2, a3, a4, a5, a6, a7, a8, b1, b2, b3, b4, b5, b6, b7, b8, r: byte;
    k: ARRAY[1..21,1..8] OF byte; k1: ARRAY[1..8] OF byte;
    logtab, exptab: ARRAY[0..255] OF integer; i, j, flag: integer;

PROCEDURE mat1(VAR a1, a2, b1, b2: byte);
BEGIN b2:= a1 + a2; b1:= b2 + a1; END;

PROCEDURE invmat1(VAR a1, a2, b1, b2: byte);
BEGIN b1:= a1 - a2; b2:= -b1 + a2; END;

BEGIN
{The program here computes the powers of the primitive element 45 of the
 finite field GF(257) and stores these in the table "exptab".  Corresponding
 logarithms to the base 45 are stored in the table "logtab".}
  logtab[1]:= 0; exptab[0]:= 1;
  FOR i:= 1 TO 255 DO
   BEGIN
    exptab[i]:= (45 * exptab[i - 1]) mod 257;
    logtab[exptab[i]]:= i;
  END;
  exptab[128]:= 0; logtab[0]:= 128; exptab[0]:= 1;

  flag:= 0; writeln;
  writeln('Enter number of rounds r (max 10) desired then hit CR'); readln(r);

  REPEAT
    BEGIN
      writeln; writeln('Enter plaintext in 8 bytes with spaces');
      writeln('  between bytes, then hit CR.');
      writeln('(A byte is an integer between 0 and 255 inclusive.)');
      readln(a1, a2, a3, a4, a5, a6, a7, a8);
      writeln('Enter a key in 8 bytes');
        readln(k[1,1],k[1,2],k[1,3],k[1,4],k[1,5],k[1,6],k[1,7],k[1,8]);
      k1[1]:= k[1,1]; k1[2]:= k[1,2]; k1[3]:= k[1,3]; k1[4]:= k[1,4];
      k1[5]:= k[1,5]; k1[6]:= k[1,6]; k1[7]:= k[1,7]; k1[8]:= k[1,8];
      writeln('PLAINTEXT is ', a1:8,a2:4,a3:4,a4:4,a5:4,a6:4,a7:4,a8:4);
       writeln('The KEY is    ', k[1,1]:8,k[1,2]:4,k[1,3]:4,k[1,4]:4,
                          k[1,5]:4,k[1,6]:4,k[1,7]:4,k[1,8]:4);
{The next instructions implement the key schedule needed to derive keys
 K2, K3, ... K2r+1 from the user-selected key K1.}
     FOR i:= 2 TO 2*r + 1 DO
     FOR j:= 1 TO 8 DO
      BEGIN
       {Each byte of the  key K1 is further left rotated by 3.}
       k1[j]:= (k1[j] shl 3) + (k1[j] shr 5);
       {The key bias is added here.}
       k[i,j]:= k1[j] + exptab[exptab[9*i+j]];
     END;

{The r rounds of encryption begin here.}
     FOR i:= 1 TO r DO
      BEGIN
       {Key 2i-1 is mixed bit and byte added  to the round input.}
```

```
      a1:= a1 xor k[2*i-1,1]; a2:= a2  +  k[2*i-1,2];
      a3:= a3  +  k[2*i-1,3]; a4:= a4 xor k[2*i-1,4];
      a5:= a5 xor k[2*i-1,5]; a6:= a6  +  k[2*i-1,6];
      a7:= a7  +  k[2*i-1,7]; a8:= a8 xor k[2*i-1,8];

      {The result now passes through the nonlinear layer.}
      b1:= exptab[a1]; b2:= logtab[a2]; b3:= logtab[a3]; b4:= exptab[a4];
      b5:= exptab[a5]; b6:= logtab[a6]; b7:= logtab[a7]; b8:= exptab[a8];
      {Key 2i is now mixed byte and bit added to the result.}
      b1:= b1  +  k[2*i,1]; b2:= b2 xor k[2*i,2];
      b3:= b3 xor k[2*i,3]; b4:= b4  +  k[2*i,4];
      b5:= b5  +  k[2*i,5]; b6:= b6 xor k[2*i,6];
      b7:= b7 xor k[2*i,7]; b8:= b8  +  k[2*i,8];

      {The result now enters the first level of the linear layer.}
      mat1(b1, b2, a1, a2); mat1(b3, b4, a3, a4);
      mat1(b5, b6, a5, a6); mat1(b7, b8, a7, a8);
      {The result now enters the second level of the linear layer.}
      mat1(a1, a3, b1, b2); mat1(a5, a7, b3, b4);
      mat1(a2, a4, b5, b6); mat1(a6, a8, b7, b8);
      {The result now enters the third level of the linear layer.}
      mat1(b1, b3, a1, a2); mat1(b5, b7, a3, a4);
      mat1(b2, b4, a5, a6); mat1(b6, b8, a7, a8);

      {The round is now completed!}
       writeln('after round',i:2,a1:8,a2:4,a3:4,a4:4,a5:4,a6:4,a7:4,a8:4);
    END;

    {Key 2r+1 is now mixed bit and byte added to produce the cryptogram.}
    a1:= a1 xor k[2*r+1,1]; a2:= a2  +  k[2*r+1,2];
    a3:= a3  +  k[2*r+1,3]; a4:= a4 xor k[2*r+1,4];
    a5:= a5 xor k[2*r+1,5]; a6:= a6  +  k[2*r+1,6];
    a7:= a7  +  k[2*r+1,7]; a8:= a8 xor k[2*r+1,8];
    writeln('CRYPTOGRAM is',a1:8,a2:4,a3:4,a4:4,a5:4,a6:4,a7:4,a8:4); writeln;
    writeln('Type 0 and CR to continue or -1 and CR to stop run.'); read(flag);
  END
 UNTIL flag < 0;
END.
```