COVER SHEET

Nomination of SAFER++ as Candidate Algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE)

26 September 2000

- **Name of algorithm submitted:** SAFER++
- **Type of submitted algorithm; proposed security level; and proposed environment:**
  Block cipher; high (block length 128 bits, key length 256 bits), normal (block length 128 bits, key length 128 bits), normal-legacy (block length 64 bits, key length 128 bits); general purpose symmetric encryption for all processors from 8-bit microprocessors upwards.
- **Principal submitter:**
  Cylink Corporation
  3131 Jay street, Santa Clara, CA 95056, USA.
  Represented by Dr. Gurgen Khachatrian, Tel: (408) 855 6270;
  Fax: (408) 855 6106, e-mail: gurgenk@cylink.com
- **Auxiliary submitters:**
  Prof. James L. Massey (Prof. *em.*, ETH Zürich, Switzerland)
  Dr. Gurgen H. Khachatrian (Cylink corporation, USA)
  Dr. Melsik K. Kuregian (National Academy of Sciences, Armenia)
- **Inventors of algorithm:**
  Prof. James L. Massey (Prof. *em.*, ETH Zürich, Switzerland)
  Dr. Gurgen H. Khachatrian (Cylink corporation, USA)
  Dr. Melsik K. Kuregian (National Academy of Sciences, Armenia)
- **Owner of algorithm:** Cylink relinquishes all proprietary rights to SAFER++ and consigns this algorithm to the public domain.
- **Signature of submitter:**

Contents

1 Introduction

This report nominates the cipher SAFER++ for use in the New European Schemes for Signatures, Integrity, and Encryption (NESSIE).

The proposed cipher SAFER++ is based on the existing SAFER family of ciphers, which comprises the ciphers SAFER K-64, SAFER K-128, SAFER SK-64, SAFER SK-128, SAFER SK-40 and SAFER+. The block size of all the "SAFER K" and "SAFER SK" ciphers in the existing SAFER family is 64 bits, while the user-selected-key length is 40 or 64 or 128 bits as indicated in the name of the particular cipher. The cipher SAFER+ has a block size of 128 bits and user-selected-key sizes of 128, 192 and 256 bits. The cipher SAFER+ was a candidate algorithm for the U. S. Advanced Encryption Standard (AES) and was subsequently adopted for use in the challenge/response entity authentication scheme in the Bluetooth protocol for wireless communications [1].

The "SAFER K" and "SAFER SK" ciphers in the existing SAFER family are non-proprietary ciphers that were designed by Prof. James L. Massey of the ETH Zürich (Swiss Federal Institute of Technology, Zürich) at the request of Cylink Corporation. The name "SAFER" was chosen by Massey as an acronym for "**S**ecure **A**nd **F**ast **E**ncryption **R**outine". The first of these ciphers, SAFER K-64, was publicly announced at the Dec. 9–11, 1993, Fast Software Encryption Workshop in Cambridge, England [2]. The other "SAFER K" and "SAFER SK" ciphers differ from SAFER K-64 only in their key schedules and in the recommended number of encryption rounds to be used.

The cipher SAFER+ was designed by Massey together with Dr. Gurgen H. Khachatrian and Dr. Melsik K. Kuregian, then both with the National Academy of Sciences, Armenia. The earlier ciphers in the SAFER family achieved rapid "diffusion" by exploiting a multidimensional linear transform, called the 2-point Pseudo-Hadamard Transform (2-PHT), in which the "shuffling" operation between levels of the 2-PHT operation was the "Hadamard shuffle", or "decimation by two", which is famililar from its use in the usual Fast Fourier Transform. SAFER+ resulted from the realization that even better diffusion could be obtained by judicious choice of the shuffle between levels of the 2-PHT operation. The shuffle chosen for use in SAFER+ was called the "Armenian Shuffle" by Massey in recognition of its development by Khachatrian and Kuregian and was shown by Massey to provide the best possible diffusion among all shuffles for use with the 2-PHT [3]. The name

SAFER+ was chosen to reflect the fact that this cipher was a significant improvement over the previous ciphers in the SAFER family.

The cipher SAFER++, which is the subject of this nomination, offers further substantial improvement over SAFER+, as reflected in the name chosen for this new cipher. In particular, SAFER++ is faster and simpler than SAFER+ and is at least as strong cryptographically. The improvements incorporated in SAFER++ were developed by Massey (now retired from the ETH-Zürich and working as an independent consultant), Khachatrian (now with Cylink Corporation, USA) and Kuregian. The main new feature of SAFER++ is its use of the 4-point Pseudo-Hadamard Transform (4-PHT) in place of the 2-PHT that was used in all previous ciphers in the SAFER family.

SAFER++ provides for a standard block size of 128 bits for plaintext and ciphertext and accommodates user-selected-key lengths of 128 and 256 bits. SAFER++ also includes a "legacy version" which provides for a block size of 64 bits with a user-selected-key length of 128 bits.

This nomination is structured as follows. In Section 2, we provide a complete description of the SAFER++ encrypting, decrypting and key–schedule algorithms. In Section 3, we explain in detail the rationale for the design of SAFER++. Section 4 documents the computational efficiency of SAFER++ for software implementation in portable C code. The cryptographic strength of SAFER++ and certain analyses are described in Section 5. The advantages and limitations of SAFER++ are indicated in Section 6. The non-proprietary status of SAFER++ is affirmed in Section 7. The final Section 8 lists the diskettes included with this nomination.

2 Specification of the SAFER++ Algorithm

2.1 System Specification of the SAFER++ Algorithm for the Standard Block Size of 128 Bits

The encryption and decryption structure of the SAFER++ algorithm is shown in Fig. 1 for the standard block size of 128 bits.

As indicated in Fig. 1, the input for encryption is the plaintext block of 16 bytes (128 bits). (The convention used throughout this proposal for numbering bytes and bits is the same as that in the Data Encryption Standard (DES) [4], i.e., byte 1 is the most significant [or leftmost] byte in the block and byte 16 is the least significant [or rightmost] byte. Similarly, bit 1 is the most significant bit of a byte and bit 8 is the least significant bit.) The plaintext block then passes through r rounds of encryption where r is determined by the key length chosen for encryption in the following manner:

- if key length = 128 bits, then $r = 7$ rounds.
- if key length = 256 bits, then $r = 10$ rounds.

Two 16-byte round subkeys are used within each round of encryption. These round subkeys $(K_1, K_2, \ldots, K_{2r})$ are determined from the user-selected key according to a key schedule described below. One further round subkey (K_{2r+1}) is "added" to the block produced by the r rounds of encryption in the manner that bytes 1, 4, 5, 8, 9, 12, 13, and 16 are added bit-by-bit modulo two (the bitwise "exclusive-or" operation) while bytes 2, 3, 6, 7, 10, 11, 14 and 15 are added modulo 256 ("byte addition"). This "addition" of round subkey K_{2r+1} constitutes the *output transformation* for SAFER++ encryption and produces the ciphertext block of 16 bytes.

As indicated in Fig. 1, the input for decryption is the ciphertext block of 16 bytes. Decryption begins with the *input transformation* that undoes the output transformation of the encryption process. In the input transformation, the round subkey K_{2r+1} is "subtracted" from the ciphertext block in the manner that the round subkey bytes 1, 4, 5, 8, 9, 12, 13, and 16 are added bit-by-bit modulo two (the bitwise "exclusive-or" operation) to the corresponding ciphertext bytes (because modulo-two addition and subtraction coincide) while round subkey bytes 2, 3, 6, 7, 10, 11, 14 and 15 are subtracted modulo 256 ("byte subtraction") from the corresponding ciphertext bytes. The result of this "subtraction" is the same 16-byte block as was produced from the r rounds of encryption before the output transformation was applied. This block then passes through the r rounds of decryption, round 1 of which undoes round r of encryption, round 2 of which undoes round $r-1$ of encryption, ... , and round r of which undoes round 1 of encryption to produce the original plaintext block. Note that the round keys for decryption are the same as those for encryption but are used in reverse order.

2.1.1 SAFER++ Encryption Round for 128-Bit Block Size

The details of one round of SAFER++ encryption for the standard block size of 128 bits are shown in Fig. 2. This round consists of a key-controlled substitution followed by an *invertible linear transformation*. The first operation within round i, $1 \leq i \leq r$, is the "addition" of the round subkey K_{2i-1} to the 16-byte round input in the manner that bytes 1, 4, 5, 8, 9, 12, 13, and 16 are added bit-by-bit modulo two while bytes 2, 3, 6, 7, 10, 11, 14 and 15 are added modulo 256. The 16-byte result of this "addition" is then processed by a *nonlinear layer* in the manner that the value x of byte j is converted to 45^x mod 257 for bytes $j = 1, 4, 5, 8, 9, 12, 13$, and 16 (with the convention that when $x = 128$ then 45^{128} mod $257 = 256$ is represented by 0), while the value x of byte j is converted to $\log_{45}(x)$ for bytes $j = 2, 3, 6, 7, 10, 11, 14$ and 15 (with the convention that when $x = 0$ then the output $\log_{45}(0)$ is represented by 128). The round key K_{2i} is then "added" to the output of the nonlinear layer in the manner that bytes 2, 3, 6, 7, 10, 11, 14 and 15 are added together bit-by-bit modulo two, while bytes 1, 4, 5, 8, 9, 12, 13, and 16 are added together modulo 256. The 16-byte result $\mathbf{x}$ of this "addition" is the ouput of the key-controlled substitution and becomes the input to the invertible linear transformation of SAFER++.

The first operation of this invertible linear transformation is application of the *pre-shuffle*, which is the coordinate permutation [9, 6, 3, 16, 1, 14, 11, 8, 5, 2, 15, 12, 13, 10, 7, 4] in DES notation, i.e., the first ouput byte is the ninth input byte, the second output byte is the sixth input byte, etc. The 16 output bytes from the pre-shuffle are then taken in sets of four consecutive bytes to form the inputs to four 4-PHT transformations. The 4-PHT [3] is the invertible linear transformation whose matrix is

$$\mathbf{H_4} = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

A 4-byte input $[\mathbf{a\ b\ c\ d}]$ to the 4-PHT is transformed to the 4-byte output $[\mathbf{A\ B\ C\ D}]$ in the manner

$$[\mathbf{A\ B\ C\ D}] = [\mathbf{a\ b\ c\ d}]\ \mathbf{H_4}$$

where the arithmetic is byte arithmethic, i.e., arithmetic modulo 256. This 4-PHT operation is realized with only six byte additions as shown in Fig. 9. The *mid-shuffle*, which again is the coordinate permutation [9, 6, 3, 16, 1, 14, 11, 8, 5, 2, 15, 12, 13, 10, 7, 4], is then applied to the outputs of the four 4-PHT transformations. The 16 output bytes from the mid-shuffle are then taken in sets of four consecutive bytes to form the inputs to another four 4-PHT transformations. The outputs of these latter 4-PHT transformations form the output of the encryption round.

2.1.2 SAFER++ Decryption Round for 128-Bit Block Size

The details of one round of SAFER++ decryption for the standard block size of 128 bits are shown in Fig. 3. The operations in the decryption round simply invert in reverse order the operations in the encryption round. Thus, the first operation in the decryption round is to apply four 4-IPHT transformations to the 16-byte round input, where IPHT stands for *Inverse Pseudo-Hadamard Transform.* The 4-IPHT [3] is the linear transformation whose matrix is

$$\mathbf{H_4}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & 4 \end{bmatrix}.$$

A 4-byte input $[\mathbf{A\ B\ C\ D}]$ to the 4-IPHT is transformed to the 4-byte output $[\mathbf{a\ b\ c\ d}]$ in the manner

$$[\mathbf{a\ b\ c\ d}] = [\mathbf{A\ B\ C\ D}]\ \mathbf{H_4}^{-1}$$

where the arithmetic is byte arithmethic, i.e., arithmetic modulo 256. This 4-IPHT operation is realized with only six byte subtractions as shown in Fig.

9. The coordinate permutation [5, 10, 3, 16, 9, 2, 15, 8, 1, 14, 7, 12, 13, 6, 11, 4] is then applied to the output of these 4-IPHT transformations. This coordinate permutation is the inverse of the mid-shuffle that was the next-to-last operation in the encryption round. Four additional 4-IPHT transformations are then applied to the output of this coordinate transformation, after which the coordinate permutation [5, 10, 3, 16, 9, 2, 15, 8, 1, 14, 7, 12, 13, 6, 11, 4] is again applied to the outputs to invert the pre-shuffle of the encryption round. The 16-byte result $\mathbf{x}$ is the same as that which formed the input to the invertible linear transformation in the corresponding encryption round. This 16-byte result $\mathbf{x}$ is the output of the inverse of the invertible linear transformation and thus becomes the input to the inverse of the key-controlled substitution.

The round subkey $K_{2r-2i+2}$ is now "subtracted" from $\mathbf{x}$ in the manner that the round subkey bytes 1, 4, 5, 8, 9, 12, 13, and 16 are subtracted modulo 256 from the corresponding bytes of $\mathbf{x}$ while round subkey bytes 2, 3, 6, 7, 10, 11, 14 and 15 are added bit-by-bit modulo 2 to the corresponding bytes of $\mathbf{x}$. The 16-byte result of this "subtraction" is then processed nonlinearly in the manner that the value x of byte j is converted to $\log_{45}(x)$ for bytes $j = 1, 4, 5, 8, 9, 12, 13$, and 16 (again with the convention that when $x = 0$ then $\log_{45}(0)$ is represented by 128), while the value x of byte j is converted to $45^x \bmod 257$ for bytes $j = 2, 3, 6, 7, 10, 11, 14$ and 15 (again with the convention that when $x = 128$ then $45^{128} \bmod 257 = 256$ is represented by 0). The round subkey $K_{2r-2i+1}$ is then "subtracted" from the 16-byte result in the manner that the round subkey bytes 1, 4, 5, 8, 9, 12, 13, and 16 are added bit-by-bit modulo 2 to the corresponding input bytes while round subkey bytes 2, 3, 6, 7, 10, 11, 14 and 15 are subtracted modulo 256 from the corresponding input bytes to produce the 16-byte round output.

2.1.3 SAFER++ Key Schedule

The $2r + 1$ 16-byte round subkeys required for the r encryption rounds and for the output transformation of SAFER++ encryption for the standard 128-bit block length (which are the same as the subkeys required for the input transformation and the r rounds of SAFER++ decryption) are produced from the input key according to a key schedule. Because the number of rounds r depends on the key length selected in the manner

- if key length = 128 bits, then $r = 7$ rounds

- if key length = 256 bits, then $r = 10$ rounds,

it follows that fewer round subkeys are required for the 128-bit key-length option than for the 256-bit key-length option. For the legacy-block-length option of 64 bits, an even smaller number of 16-byte round subkeys are required, as will be explained in Section 2.2. However, the required round subkeys are all "nested" in the sense that the round subkeys required for the 64-bit legacy-block-length option form the first part of the round subkeys required for the 128-bit key-length option, which in turn form the first part of the round subkeys required for the 256-bit key-length option. Thus, all the round subkeys required by the various options for SAFER++ can be produced with the same key schedule, which simplifies both software and hardware implementation.

Calculation of bias words in the key schedule

The key schedule of SAFER++ makes use of 16-byte *bias words* to "randomize" the round subkeys produced. The number of bias words for the standard 128-bit block length is the same as the number $2r + 1$ of round subkeys, i.e., 15 or 21 depending on whether the user-selected-key length is 128 bits or 256 bits, respectively. (For the legacy-block-length option of 64 bits, only 9 bias words are required.)

Let B_i, $i = 1, 2, 3, \ldots, 2r+1$, denote the $2r+1$ bias words for the standard 128-bit block length. The bias word B_1 is a "dummy" word that is never actually used in the SAFER++ key schedule but is convenient to have defined for programming purposes. Let $B_{i,j}$ denote the j-th byte of B_i. For bias words B_2, B_3, … B_{15}, which are used for both key lengths and are the only bias words used for a 128-bit user-selected key, the bias bytes are determined in the following manner:

$$B_{i,j} = 45^{(45^{17i+j} \bmod 257)} \bmod 257$$

(where $B_{i,j}$ is represented as 0 in case this expression gives a value of 256 and) where this expression applies for $i = 2, 3, \ldots, 15$ and $j = 1, 2, \ldots, 16$. The additional bias words $B_{16}, B_{17}, \ldots, B_{21}$, which are used with a 256-bit user-selected key, are computed in the following manner:

$$B_{i,j} = 45^{17i+j} \bmod 257$$

(where $B_{i,j}$ is represented as 0 in case this expression gives a value of 256 and) where this expression applies for $i = 16, 17, \ldots, 21$ and $j = 1, 2, \ldots, 16$.

70	151	177	186	163	183	16	10	197	55	179	201	90	40	172	100
236	171	170	198	103	149	88	13	248	154	246	110	102	220	5	61
138	195	216	137	106	233	54	73	67	191	235	212	150	155	104	160
93	87	146	31	213	113	92	187	34	193	190	123	188	153	99	148
42	97	184	52	50	25	253	251	23	64	230	81	29	65	68	143
221	4	128	222	231	49	214	127	1	162	247	57	218	111	35	202
58	208	28	209	48	62	18	161	205	15	224	168	175	130	89	44
125	173	178	239	194	135	206	117	6	19	2	144	79	46	114	51
192	141	207	169	129	226	196	39	47	108	122	159	82	225	21	56
252	32	66	199	8	228	9	85	94	140	20	118	96	255	223	215
250	11	33	0	26	249	166	185	232	158	98	76	217	145	80	210
24	180	7	132	234	91	164	200	14	203	72	105	75	78	156	53
69	77	84	229	37	60	12	74	139	63	204	167	219	107	174	244
45	243	124	109	157	181	38	116	242	147	83	176	240	17	237	131
103	9	148	235	38	168	107	189	24	52	27	187	191	114	247	64
72	156	81	47	59	85	227	192	159	216	211	243	141	177	255	167
220	134	119	215	166	17	251	244	186	146	145	100	131	241	51	239
44	181	178	43	136	209	153	203	140	132	29	20	129	151	113	202
163	139	87	60	130	196	82	92	28	232	160	4	180	133	74	246
84	182	223	12	26	142	222	224	57	252	32	155	36	78	169	152

Table I: Bias words B_2 (first row), B_3 (second row), ... , and B_{21} (last row) used in SAFER++ key schedule.

Table I gives lists all 20 bias words B_2, B_3, ... B_{21} that are actually used in the SAFER++ key schedule. These bias words were computed from the above formulas and are listed in the manner that the 16 bytes of B_2 form the first row, the 16 bytes of B_3 form the second row, ... , and the 16 bytes of B_{21} form the last row.

Details of the key schedule

The manner in which the 16-byte subkeys are generated by the SAFER++ key schedule is diagrammed in Figs. 4 and 5 for the odd-index subkeys (i.e., subkeys $K_1, K_3, K_5, \ldots, K_{19}, K_{21}$) and for the even-index subkeys (i.e., subkeys $K_2, K_4, K_6, \ldots, K_{18}, K_{20}$), respectively. For the 128-bit key-length option in which SAFER++ uses 7 rounds of encryption or decryption, only the 15 subkeys K_1, K_2, ... , K_{15} are used in the encryption and decryption process and hence only these 15 subkeys need be generated. For the 256-bit key-length option in which SAFER++ uses 10 rounds of encryption or decryption, all 21 subkeys $K_1, K_2, \ldots K_{21}$ are used in the encryption and decryption process and hence all must be generated.

It is convenient to think of the operation of the SAFER++ key schedule for the 128-bit key-length option as first expanding the 128-bit user-selected key to a 256-bit "full key" by simple repetition, after which the subkeys are generated in the same manner as for the 256-bit key-length option. As can be seen from Figs. 4 and 5, the two halves of the full key are processed independently by the SAFER++ key schedule.

2.2 System Specification of the SAFER++ Algorithm for the Legacy Block Size of 64 Bits

The encryption and decryption structure of the SAFER++ algorithm is shown in Fig. 6 for the legacy block size of 64 bits. Only one 16-byte subkey is used within each of the eight rounds of encryption. These round subkeys $(K_1, K_2, \ldots, K_8)$ are the same subkeys described in Section 2.1.3, as also is subkey K_9. The first eight bytes of subkey K_9 are "added" to the block produced by the eight rounds of encryption in the manner that bytes 1, 4, 5, and 8 are added bit-by-bit modulo two (the bitwise "exclusive-or" operation) while bytes 2, 3, 6, and 7 are added modulo 256 ("byte addition"). This "addition" of round subkey K_9 constitutes the *"halved" output transformation* for SAFER++ encryption for the legacy block size of 64 bits and produces the ciphertext block of 8 bytes.

As shown in Fig. 6, the input for decryption is the ciphertext block of r bytes. Decryption begins with the *"halved" input transformation* that undoes the "halved" output transformation in the encryption process. In the input transformation, the first eight bytes of subkey K_9 are "subtracted" from the ciphertext block in the manner that the round subkey bytes 1, 4, 5, and 8 are added bit-by-bit modulo two (the bitwise "exclusive-or" operation) to the corresponding ciphertext bytes (because modulo-two addition and subtraction coincide) while round subkey bytes 2, 3, 6, and 7 are subtracted modulo 256 ("byte subtraction") from the corresponding ciphertext bytes. The result of this "subtraction" is the same 8-byte block as was produced from the eight rounds of encryption before the "halved" output transformation was applied. This block then passes through the eight rounds of decryption, round 1 of which undoes round 8 of encryption, round 2 of which undoes round 7 of encryption, ... , and round 8 of which undoes round 1 of encryption to produce the original plaintext block. Note that the subkeys for decryption are the same as those for encryption but are used in reverse order.

Fig. 7 shows the details of one round of SAFER++ encryption for the legacy block size of 64 bits. The input for encryption is the plaintext block of 8 bytes (64 bits). This round consists of a *"halved" key-controlled substitution* followed by the same *invertible linear transformation* used with the standard block size of 128 bits, cf. Fig. 2. The "halved" key-controlled substitution comprises the left half of the key-controlled substitution used with the standard block size of 128 bits. The 8-byte output of the "halved" key-controlled substitution is expanded to 16 bytes in the manner that each half-byte of the 8-byte output becomes, after the expansion, the upper half of a byte whose lower half-byte is 0. The 16-byte result of this expansion then becomes the input to the same invertible linear transformation as used with the standard block size of 128 bits. This invertible linear transformation has the property that all lower half-bytes are also zero in its sixteen-byte output. No changes whatsover in the standard-block-size-of-128-bits implementation of the invertible linear transformation are required to accomodate the legacy block size of 64 bits. The final operation in the SAFER++ encryption round for the legacy block size of 64 bits is the compaction of the 16-byte output of the invertible linear transformation to 8 bytes in the manner that the lower half byte (which is all zero) of each of the 16-bytes is discarded and successive pairs of the upper half bytes of the 16-bytes are merged to form the bytes of the 8-byte round output.

Fig. 8 shows the details of one round of SAFER++ decryption for the legacy block size of 64 bits. The input for decryption is the ciphertext block of 8 bytes (64 bits). The operations in the decryption round simply invert in reverse order the operations in the encryption round. Thus, the first operation in the decryption round is the expansion of the 8-byte round input to 16 bytes in the manner that each half-byte of the 8-byte input becomes, after the expansion, the upper half of a byte whose lower half-byte is 0. The inverse of the invertible linear transformation is then applied to the 16-byte result in exactly the same manner as for the standard-block-size-of-128-bits case, cf. Fig. 3. The next operation in the decryption round for the legacy block size of 64 bits is the compaction of the 16-byte result to 8 bytes in the manner that the lower half byte (which is all zero) of each of the 16-bytes is discarded and successive pairs of the upper half bytes of the 16-bytes are merged to form the bytes of the 8-byte result. This 8-byte result becomes the input to the inverse of the "halved" key-controlled substitution. The eight bytes in the second half of subkey K_{9-i} are now "subtracted" from this 8-byte input in the manner that subkey bytes 9, 12, 13, and 16 are subtracted modulo 256 from bytes 1, 4, 5, and 8 while subkey bytes 10, 11, 14, and 15 are added bit-by-bit modulo 2 to bytes 2, 3, 6, and 7. The 8-byte result of this "subtraction" is then processed nonlinearly in the manner that the value x of byte j is converted to $\log_{45}(x)$ for bytes $j = 1, 4, 5$, and 8 (with the convention that when $x = 0$ then $\log_{45}(0)$ is represented by 128), while the value x of byte j is converted to $45^x \bmod 257$ for bytes $j = 2, 3, 6$, and 7 (with the convention that when $x = 128$ then $45^{128} \bmod 257 = 256$ is represented by 0). The first half of subkey K_{9-i} is then "subtracted" from the 8-byte result in the manner that subkey bytes 1, 4, 5, and 8 are added bit-by-bit modulo 2 to the corresponding input bytes while round subkey bytes 2, 3, 6, and 7 are subtracted modulo 256 from the corresponding input bytes to produce the 8-byte round output.

2.3 Examples for SAFER++

The above description of SAFER++ is intended to be detailed and clear enough for a designer to program this cipher or to implement it in silicon. The data in the following three examples, one for each key-length/block-length pair accomodated by SAFER++, may be helpful to a designer who implements SAFER++ from this description.

Example of SAFER++ for 128-bit key and standard block size of 128 bits:

This is the 16 byte user-selected input key:

187 183 41 209 77 47 126 223 62 75 148 150 109 3 219 183

This is the plaintext block input:

100 73 199 247 219 132 118 11 126 64 147 2 241 36 122 215

This is the ciphertext block output:

73 209 104 215 224 239 100 92 200 9 126 183 19 210 64 160

These are the round subkeys K_1, K_2, ... , K_{15} (one key per row):

187	183	41	209	77	47	126	223	62	75	148	150	109	3	219	183
3	224	63	36	28	170	14	251	31	219	103	52	114	6	105	118
54	31	253	145	6	140	231	223	29	63	81	46	92	201	149	43
45	93	54	133	41	101	204	114	112	153	241	139	5	31	223	15
49	73	121	28	184	37	165	36	248	241	123	246	224	84	222	38
193	160	167	83	215	99	72	177	152	45	193	114	250	28	216	119
214	131	120	11	57	139	139	139	112	128	0	39	184	19	106	255
53	151	133	99	2	235	114	28	195	87	87	158	212	188	2	17
187	248	70	133	47	138	169	44	72	206	185	185	32	123	161	177
26	49	131	20	153	192	129	57	12	41	195	45	188	90	8	54
33	197	157	135	254	209	153	67	75	214	136	201	43	158	214	102
39	229	39	183	137	125	29	40	58	65	252	170	213	80	204	104
238	228	196	255	14	22	31	90	43	159	58	80	72	49	80	126
198	58	47	6	2	23	160	50	49	214	11	150	250	16	248	63
156	209	133	91	123	89	109	169	174	140	210	168	29	99	71	56

Example of SAFER++ for 256-bit key and standard block size of 128 bits:

This is the 32 byte user-selected input key:

116 166 17 250 49 106 16 91 54 27 250 231 246 242 114 195

233 205 53 0 176 246 143 156 1 6 124 182 252 69 23 21

This is the plaintext block input:

205 103 98 71 137 164 156 11 41 57 189 69 229 17 87 82

This is the ciphertext block output:

102 90 103 155 126 97 237 174 178 229 67 19 20 116 37 188

These are the round subkeys K_1, K_2, ... , K_{21} (one key per row):

116	166	17	250	49	106	16	91	54	27	250	231	246	242	114	195
180	64	177	63	90	51	244	18	245	26	104	176	132	224	84	245
48	105	246	96	107	107	229	211	182	147	179	42	2	204	48	90
138	36	197	168	163	235	66	65	176	184	117	2	192	255	59	59
112	253	147	212	56	34	11	57	145	240	229	183	134	224	205	165
165	40	6	180	53	87	88	121	185	203	112	106	17	39	222	143
29	113	88	74	210	208	177	74	202	177	169	10	116	179	14	142
205	240	220	96	6	221	186	131	111	85	29	97	85	130	111	10
179	200	172	214	184	121	64	56	178	135	168	161	73	95	220	67
240	112	132	144	171	154	108	184	126	218	35	159	215	152	145	28
186	25	255	131	164	212	52	114	7	208	210	194	250	3	181	100
103	4	171	46	68	93	121	84	82	158	195	57	248	202	82	222
135	227	46	192	180	162	14	217	189	222	238	106	0	177	77	228
231	216	222	254	25	34	166	74	227	186	147	245	91	110	236	79
246	2	46	62	55	249	17	56	155	211	192	136	92	252	140	94
9	79	209	164	204	168	129	155	9	199	59	123	78	72	150	232
244	16	247	64	53	134	77	208	250	14	238	237	116	167	241	25
43	244	32	215	43	200	119	216	194	194	116	25	106	27	235	151
213	249	112	119	34	213	111	88	82	66	22	209	61	51	97	245
13	139	184	41	161	253	84	104	20	85	153	142	226	175	174	201
3	201	133	13	207	241	143	143	183	107	79	194	96	24	240	2

Example of SAFER++ for 128-bit key and legacy block size of 64 bits:

This is the 16 byte user-selected input key:
244 166 98 166 157 225 71 116 3 130 47 150 74 232 144 2
This is the plaintext block input:
160 88 37 206 208 68 44 153

This is the ciphertext block output:
215 249 17 66 162 215 134 96

These are the round subkeys K_1, K_2, ... , K_9 (one key per row):

244	166	98	166	157	225	71	116	3	130	47	150	74	232	144	2
123	170	230	166	178	241	179	34	217	176	103	27	161	172	188	242
132	84	17	62	56	178	24	173	195	63	136	168	138	92	121	122
215	254	155	23	82	239	59	167	112	83	188	245	154	62	81	237
54	117	6	102	5	153	78	36	198	79	199	155	217	232	205	186
26	4	242	181	115	176	72	32	139	136	231	57	151	148	117	226
250	213	140	232	163	139	255	34	67	170	62	12	116	248	189	64
200	48	108	182	2	135	47	179	13	73	126	124	251	86	12	104
128	47	225	133	12	111	94	119	215	7	168	242	245	203	83	122

3 Design Rationale

The major design principles underlying the SAFER++ algorithm are the following:

- Encrypting Structure
- Byte Orientation
- Group Operation at Round Input
- Use of Two Additive Groups
- Use of the Exponential and Logarithm in the Nonlinear Layer
- Fast Diffusion via the Matrix **M**
- Scalability
- Biases in Key Schedule
- Parity Word and Selections in Key Schedule
- Number of Rounds

Each of these principles will now be explained.

Encrypting Structure

It should be emphasized that SAFER++, unlike the Date Encryption Algorithm (DEA) of DES and many later iterative block ciphers, is **not** a "Feistel" cipher. In a Feistel cipher, the round input is split into its left and right halves, say L_{i-1} and R_{i-1} for the input to round i. Some nonlinear function f from a half–block/round–key pair to a half block is then applied to the right–half input R_{i-1} and the round key K_i. The value of this function is then added bit-by-bit modulo 2 to the left–half input L_{i-1} to create a "new" left half $\mathrm{L}_{i-1} \oplus f(\mathrm{R}_{i-1}, \mathrm{K}_i)$. The left and right halves are then "swapped" to produce the left half L_i and right half R_i of the output of round i as $\mathrm{L}_i = \mathrm{R}_{i-1}$ and $\mathrm{R}_i = \mathrm{L}_{i-1} \oplus f(\mathrm{R}_{i-1}, \mathrm{K}_i)$. The Feistel structure has the feature that the round function is invertible regardless of the choice of the function f. The "swapping of halves" is omitted in the final round of encryption with the result that decryption can be performed by using the encrypting algorithm together with the round keys taken in reverse order. This "encryption/decryption similarity" is an implementation advantage for a Feistel cipher. On the other hand, diffusion in a Feistel cipher tends to be somewhat slow. For instance, it takes two rounds before every bit of the input can be altered by the algorithm. Moreover, the design of the function f, which is the main source of cryptographic strength, poses substantial problems that are generally solved by the use of "random-looking" tables that are difficult to justify analytically to a suspicious cryptanalyst.

A generalization of the Feistel cipher is the kind of iterative block cipher that is sometimes called a "substitution–permutation" cipher. Within a round of such a cipher, an invertible function controlled by the round key is first applied to the round input, then a permutation of coordinates is applied to the result. The substitution–permutation structure gives the designer more freedom than does the Feistel structure so that more rapid diffusion can be obtained, but in general the property of encryption/decryption similarity is lost, i.e., the encrypting and decrypting algorithms differ in more than the ordering of the round keys.

SAFER++ (as is also the case with all prior ciphers in the SAFER family) is neither a Feistel cipher nor a substitution–permutation cipher. There is no fundamental reason to alternate between substitutions and permutations (of coordinates) to create good confusion and diffusion. SAFER++ can be called a *substitution/linear–transformation* cipher, by which we mean that an invertible function controlled by the round key is first applied to the round input, then an invertible linear transformation is applied to the result. Because a permutation of coordinates is also an invertible linear transformation, the substitution/linear-transformation structure is a generalization of the substitution–permutation structure and offers the designer still more freedom. In particular, as will be seen below, extremely fast diffusion can be achieved by the judicious choice of the invertible linear transformation of the cipher.

Byte Orientation

SAFER++ is a *byte–oriented* cipher (as are also all prior ciphers in the SAFER family) in the sense that during encryption, decryption, and execution of the key schedule, only functions from one byte to one byte are used. This byte orientation facilitates both software and hardware realizations of the encrypting and decrypting algorithms. In particular, it causes

SAFER++ to implement extremely well even on 8-bit microprocessors.

During set–up of the SAFER++ algorithm (i.e., during preparation of the tables for exponentiation of 45 modulo 257, for logarithms to the base 45, and for the key biases), some simple integer operations are performed, but these are done once and for all.

Group Operation at Round Input

As can be seen from Figs. 1 and 2, the first operation within the SAFER++ encrypting algorithm is to combine the 16-byte plaintext block with the 16-byte round subkey K_1 by the "addition" operation described in Section 2.1. This "addition" operation is a group operation. Moreover, the 16 bytes of round subkey K_1 are the first 16 bytes of the user–selected key regardless of the key length chosen. If the user selects the key uniformly at random, then, for an arbitrarily chosen plaintext block, the result of the group operation is equally likely to be any of the 2^{128} possible 16 byte blocks. This means that the result of this group operation is statistically independent of the plaintext block and hence that this group operation provides *perfect secrecy* in the sense of Shannon [5] when the key is used only one time. Of course, the SAFER++ key will typically be used to encrypt very many plaintext blocks before being changed, but it is nonetheless good cryptographic practice to incorporate this element of provable perfect secrecy into an encryption algorithm.

Use of Two Additive Groups

The group operation used to insert round subkeys, which was described in Section 2.1.1, is the operation of a "product group" composed of sixteen smaller groups. For eight of these smaller groups, the operation is addition modulo 256 (i.e., usual byte addition) and for the remaining eight smaller groups, the operation is bit-by-bit addition modulo 2 (i.e., the bitwise exclusive-or operation).

Using two different group operations to insert a round subkey enhances the "confusion" created by SAFER++ encryption. Consider for instance the function exptab(.) such that $\text{exptab}(x) = 45^x \bmod 257$ (with the convention that when $x = 128$ then $45^{128} \bmod 257 = 256$ is represented by 0) for $0 \leq x \leq 255$, which is the function corresponding to "exp" in Fig. 2. For this function,

$$\text{exptab}(x_1 + x_2) = \text{exptab}(x_1) \times \text{exptab}(x_2)$$

where "+" denotes addition modulo 256 and "$\times$" denotes multiplication modulo 257. However, in general

$$\text{exptab}(x_1 \oplus x_2) \neq \text{exptab}(x_1) \times \text{exptab}(x_2)$$

where "$\oplus$" denotes bit-by-bit addition modulo 2 and "$\times$" denotes multiplication modulo 257. This difference is important–it shows that although adding two bytes modulo 256 then evaluating exptab(.) of the result is equivalent to evaluating exptab(.) of each byte separately and then multiplying the results, this "morphism" property does **not** hold when the addition is bit-by-bit modulo 2. This is the reason that, as can be seen in Fig. 2, those bytes of round subkey K_{2i-1} that are affected by the "exp" operation are inserted by bit-by-bit addition modulo 2 and **not** by addition modulo 256 as are the other bytes of K_{2i-1}. The insertion of those other bytes by addition modulo 256, however, is important for the strength of SAFER++ against linear cryptanalysis as will be discussed in Section 5.3. Having both types of addition available in SAFER++ makes it possible to exploit the particular cryptographic strengths of each.

Use of the Exponential and Logarithm in the Nonlinear Layer

As was described in Section 2.1.1 above, the 16 bytes within a round of SAFER++ that result from the "addition" of the first of the two round subkeys are used as inputs to either the function exptab(.) or its inverse logtab(.) [where $\text{logtab}(y) = \log_{45}(y)$ for $0 \leq y \leq 255$ (with the convention that $\log_{45}(0) = 128$)], which is the function corresponding to "log" in Fig. 2. The choice of exptab(.) and logtab(.) as the mutually inverse nonlinear functions within the "nonlinear layer" of a round of SAFER++ was motivated by several factors. First of all, these are well-defined mathematical functions and their use obviates the suspicions of intentional weakness that might be raised if mutually inverse nonlinear functions defined only by "random looking" tables were chosen. Further, it was shown in [6] that, for the boolean functions determining single bits of exptab(.) and logtab(.), the number of terms of a given nonlinear order in the algebraic normal form of the function follows virtually the same distribution as for a randomly chosen boolean function. Moreover, Vaudenay [7], as a result of his cryptanalysis of SAFER K–64, concluded that the "choice [of exptab(.) and logtab(.) as the mutually inverse nonlinear functions] is a very good one" because a substantial fraction of all mutually inverse nonlinear functions when used in their place would lead to a known plaintext attack faster than exhaustive search.

Fast Diffusion via the Matrix M

The main difference of SAFER++ from the previous ciphers of the SAFER family, including SAFER+, is in the choice of the invertible linear transformation used in the encryption round, cf. Fig. 2. Indeed, a major feature of SAFER+ had already been the choice of an invertible linear transformation that yielded considerably better diffusion than the invertible linear transformation used in the previous ciphers of the SAFER family. The invertible linear transformation of SAFER++ has been designed such that it not only provides even better difffusion than that of SAFER+, but also runs significantly faster.

The invertible linear transformation of SAFER++ is built on the 4-PHT ("PHT" stands for Pseudo Hadamard Transform), the theory behind which was given in [3]. The matrix of the 4-PHT is

$$\mathbf{H_4} = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

A 4-byte input $[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}]$ to the 4-PHT is transformed to the 4-byte output $[\mathbf{A}\ \mathbf{B}\ \mathbf{C}\ \mathbf{D}]$ in the manner

$$[\mathbf{A}\,\mathbf{B}\,\mathbf{C}\,\mathbf{D}] = [\mathbf{a}\,\mathbf{b}\,\mathbf{c}\,\mathbf{d}]\ \mathbf{H_4}$$

where the arithmethic is byte arithmethic, i.e., arithmetic modulo 256. As shown in Fig. 9, implementation of the 4-PHT requires only six byte additions. The 4-IPHT, i.e., the inverse of the 4-PHT, is the linear transformation whose matrix is

$$\mathbf{H_4}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & 4 \end{bmatrix}.$$

A 4-byte input $[\mathbf{A}\ \mathbf{B}\ \mathbf{C}\ \mathbf{D}]$ to the 4-IPHT is transformed to the 4-byte output $[\mathbf{a}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}]$ in the manner

$$[\mathbf{a}\,\mathbf{b}\,\mathbf{c}\,\mathbf{d}] = [\mathbf{A}\ \mathbf{B}\ \mathbf{C}\ \mathbf{D}]\ \mathbf{H_4}^{-1}$$

where the arithmethic is byte arithmethic, i.e., arithmetic modulo 256. As shown in Fig. 9, implementation of the 4-PHT requires only six byte subtractions.

As shown in Fig. 2, the SAFER++ invertible linear transformation consists of a pre–shuffling of the 16 input bytes, followed by four 4-PHT transformations, followed by a mid–shuffling of the resulting 16 bytes, and ending with four 4-PHT transformations. Both the pre–shuffling and the mid–shuffling are done with the coordinate permutation [9, 6, 3, 16, 1, 14, 11, 8, 5, 2, 15, 12, 13, 10, 7, 4]. This coordinate permutation employs the "Armenian shuffle" principle according to which the i-th input byte ($i = 1, \ldots, 16$) becomes the j-th output byte where i and j are equal modulo 4. It was shown in [3] that this choice results in "optimum transform diffusion". The SAFER++ invertible linear transformation is equivalent to operating with the matrix

$$\mathbf{M} = \begin{bmatrix}
1 & 2 & 2 & 1 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 2 & 1 \\
2 & 1 & 2 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 4 & 1 & 1 \\
1 & 1 & 4 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 2 \\
1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 \\
1 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 4 & 1 & 1 & 2 & 1 & 2 & 1 \\
1 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 2 & 1 & 1 & 4 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 \\
4 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 1 \\
2 & 1 & 2 & 1 & 1 & 4 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \\
2 & 1 & 1 & 2 & 1 & 2 & 4 & 1 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 1 \\
2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 1 & 2 & 1 & 1 & 2 & 1 & 1 & 2 & 2 & 1 & 4 & 1 & 1 & 1 \\
1 & 4 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 \\
2 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 4 & 1 & 2 & 2 & 1 & 1 \\
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1
\end{bmatrix}$$

in the manner that

$$\mathbf{y} = \mathbf{x}\mathbf{M}$$

where

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}]$$

is the sixteen-byte input and

$$\mathbf{y} = [y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{16}]$$

is the the 16-byte output.

One sees that every row of the matrix **M** contains at least ten 1's, which means that every input block with a single non-zero byte will produce an output block with at least ten non-zero bytes. The diffusion provided by the matrix **M** is extremely fast. The matrix **M** has also been designed to make SAFER++ highly resistant to differential cryptanalysis, as will be explained in Section 5.2.

Scalability

SAFER++ has the property, as do all the previous ciphers in the SAFER family, that it can be scaled down to "mini-versions" that permit study of the properties of the algorithm in a simplified setting. A byte in standard SAFER++ can be reduced to 2 bits (and hence the block size reduced to 32 bits) by changing the modulo 256 and modulo 257 arithmetic used in SAFER++ to modulo 4 and modulo 5 arithmetic, respectively. Alternatively, a byte in standard SAFER++ can be reduced to 4 bits (and hence the block size reduced to 64 bits) by changing the modulo 256 and modulo 257 arithmetic used in SAFER++ to modulo 16 and modulo 17 arithmetic, respectively. [It is even possible to make a "maxi-version" of SAFER++ in which a byte in standard SAFER++ is expanded to 16 bits (and hence the block size increased to 256 bits) by changing the modulo 256 and modulo 257 arithmetic used in SAFER++ to modulo 2^{16} and modulo $2^{16} + 1$ arithmetic, respectively. This would in fact be a very strong cipher but would also be rather difficult to implement since the logarithm and exponential tables would each require 2^{16} entries of 16 bits each.] The only "arbitrary" parameter that must be selected for these non-standard versions of SAFER++ is the element of multiplicative order 2^m, modulo the prime number $2^m + 1$ where m is the number of bits in a symbol, used as the base for exponentiation and logarithms. The choice of this parameter (which was chosen as 45 in standard SAFER++ where $m = 8$) is of minor importance–any element of multiplicative order 2^m can be used in the m-bits/symbol mini-version of SAFER++.

The virtue of scalability is that it permits one to perform exhaustive searches for weaknesses on mini-versions that could not be performed on the full SAFER++ algorithm. As one example of this, we were able to show, by exhaustive analysis of a mini-version, that a "potential weakness" that had been pointed out [10] in the PHT layer of the previous SAFER family of ciphers did not in fact exist.

Biases in Key Schedule

The addition of "biases" to shifted versions of the user–selected key in order to generate round subkeys, which was described in Section 2.1.3, is used in SAFER++ to avoid "weak keys"–such as can occur with the DEA of DES and with IDEA [8] where, for instance, if the user–selected key is all-zero then all round subkeys are identical. This "weak-key" property is of minor significance for encryption because of the negligibly small probability of choosing a "weak key", but it can be significant when the algorithm is used for hashing. Essentially, the addition of the biases in the SAFER++ key schedule randomizes the round subkeys generated from a single user–selected key and hence avoids the occurrence of such "weak keys".

Parity Word and Selections in Key Schedule

In SAFER K-64, the first cipher in the SAFER family [2], the key schedule differed from that of SAFER++ as shown in Figs. 4 and 5 in that the same bytes of the key register were always selected for addition to the biases to produce the round subkeys. Knudsen [11] showed that this "stationarity" of the bytes in the key register made it possible, when SAFER K-64 was used in a hashing mode, to create "collisions" much more often than by random guessing. Knudsen suggested a modification of the key schedule to remove this weakness, which Massey then incorporated into the later ciphers (SAFER SK-64, SAFER SK-128, and SAFER SK-40) in the SAFER family [after obtaining from Knudsen a signed statement relinquishing all intellectual rights to this modified key schedule]. Knudsen's key schedule not only introduced "non-stationarity" of the bytes of the key register added to the bias, which is implemented in SAFER++ by the "selection" boxes in Figs. 4 and 5, but also included expansion of the key register by the addition of a "parity byte," i.e., a byte which is the modulo-two sum of all the bytes in the user–selected key. This augmentation of the key register, which is also performed in SAFER++ as can be seen in Figs. 4 and 5, has the very desirable consequence that, if two user–selected keys differ, then they will differ in at least two bytes of the key register where the parity byte is also present. This provides a healthy diversity of the keys produced by the key schedule of SAFER++.

Number of Rounds

The most effective cryptanalytic attack against SAFER++ appears to be Biham and Shamir's differential cryptanalysis [9]. As will be explained in Section 5.2, SAFER++ with six rounds is essentially immune to this attack. The choice of 7 rounds when a 128-bit key is used was made to provide a desirable margin of safety for the security of SAFER++. The choice of 10 rounds for SAFER++ with a 256-bit key was made to provide the super-abundant security to which a user is entitled who opts to use this longer key length.

4 Computational Efficiency of SAFER++ in portable C

The data below specify the computational speed for Portable C implementation of SAFER++.

Platform: Pentium III Processor, 667 MHz, 128 MB RAM, running MS Windows 2000, tools:MS Visual C++ 6.0.

Encryption

- SAFER++ with 128 bit key (7 rounds, 32-bit optimized implementation)–640 cycles per block or 40 cycles per byte.
- SAFER++ with 256 bit key (10 rounds, 32-bit optimized implementation)–912 cycles per block or 57 cycles per byte.
- SAFER++ with 64 bit block size and with 128 bit key (legacy version, 8 rounds, reference implementation)-1352 cycles per block or 169 cycles per byte. [An optimized implementation is not yet available for the legacy version.]

Decryption

- SAFER++ with 128 bit key (7 rounds, 8-bit optimized implementation)–1226 cycles per block or 76 cycles per byte.
- SAFER++ with 256 bit key (10 rounds, 8-bit optimized implementation)–1626 cycles per block, or 101 cycles per byte.
- SAFER++ with 64 bit block size and with 128 bit key (legacy version, 8 rounds, reference implementation)-1280 cycles per block or 160 cycles per byte. [An optimized implementation is not yet available for the legacy version.]

Setting up of Key

- SAFER++ with 128 bit key(7 rounds, reference implementation)–2333 cycles per block or 145 cycles per byte.

- SAFER++ with 256 bit key (10 rounds, reference implementation)–3227 cycles per block or 202 cycles per byte.
- SAFER++ with 64 bit block size and with 128 bit key (legacy version, 8 rounds, reference implementation)-1333 cycles per block or 166 cycles per byte.

Changing Key after Initial Setup

These data are the same as those given above for setting up of a key.

5 Security Considerations

5.1 Cryptographic Strength of SAFER++

We are not aware of any cryptographic weaknesses in SAFER++. In our opinion, for both key lengths (128 bits and 256 bits) accomodated by the algorithm, there is no chosen-plaintext attack more effective than exhaustive key search. Some reasons for this opinion are set forth in the following sections.

5.2 Strength against Differential Cryptanalysis

Differential cryptanalysis is a general attack against an iterative cipher that was introduced by Biham and Shamir in 1990 [9]. Differential cryptanalysis has proved to be the most effective general attack against the previous SAFER family of ciphers, cf. [6], [11] and [12], and appears also to be the most effective general attack against SAFER++. In the below discussion of differential cryptanalysis, we follow the approach and notation of [6] and [8].

5.2.1 Differences and Differentials

As can be seen from Fig.2, at the beginning of a round, SAFER++ combines the 16-byte round input $\mathbf{X} = [X_1, X_2, \ldots, X_{16}]$ byte-wise with the 16-byte vector $\mathbf{Z}_a = [Z_{a1}, Z_{a2}, \ldots, Z_{a16}]$, which is first of the two round subkeys. This produces the 16-byte input $\mathbf{T} = [T_1, T_2, \ldots, T_{16}]$ to the nonlinear layer in the manner that $\mathbf{T} = \mathbf{X} \otimes \mathbf{Z}_a$ where "$\otimes$" is the product–group addition operator

$$\otimes = [\oplus, +, +, \oplus, \oplus, +, +, \oplus, \oplus, +, +, \oplus, \oplus, +, +, \oplus]$$

in which $\oplus$ denotes bit-wise modulo-two addition of bytes and $+$ denotes usual byte addition, i.e., addition modulo 256.

Let the 16-byte vector $\mathbf{S} = [S_1, S_2, \ldots, S_{16}]$ denote the input to the linear layer in Fig. 2 and described by the matrix $\mathbf{M}$ above. Note that S_j is given by

$$S_j = 45^{(X_j \oplus Z_{aj})} + Z_{bj}, \quad j \in \{1, 4, 5, 8, 9, 12, 13, 16\}$$

and

$$S_j = \log_{45}(X_j + Z_{aj}) \oplus \ Z_{bj}, \quad j \in \{2, 3, 6, 7, 10, 11, 14, 15\}$$

where $\mathbf{Z}_b = [Z_{b1}, Z_{b2}, \ldots, Z_{b16}]$ is the second of the two round subkeys. The output $\mathbf{Y} = [Y_1, Y_2, \ldots, Y_{16}]$ of the linear layer, which is also the round output, is given by $\mathbf{Y} = \mathbf{S}\,\mathbf{M}$.

Let "$\oslash$" denote the product–group subtraction for 16-byte vectors, i.e.,

$$\oslash = [\oplus, -, -, \oplus, \oplus, -, -, \oplus, \oplus, -, -, \oplus, \oplus, -, -, \oplus\ ]$$

where "$-$" denotes usual byte subtraction, i.e., subtraction modulo 256. Then the *difference* $\Delta\mathbf{V} = [\Delta V_1, \Delta V_2, \ldots, \Delta V_{16}]$ between the 16-byte words $\mathbf{V}$ and $\mathbf{V}^*$ is the 16-byte vector

$$\Delta\mathbf{V} = \mathbf{V} \oslash \mathbf{V}^*.$$

Equivalently,

$$\Delta\mathbf{V} = \begin{array}{llll} [V_1 \oplus V_1^*, & V_2 - V_2^*, & V_3 - V_3^*, & V_4 \oplus V_4^*, \\ \ V_5 \oplus V_5^*, & V_6 - V_6^*, & V_7 - V_7^*, & V_8 \oplus V_8^*, \\ \ V_9 \oplus V_9, & V_{10} - V_{10}^*, & V_{11} - V_{11}^*, & V_{12} \oplus V_{12}^*, \\ \ V_{13} \oplus V_{13}^*, & V_{14} - V_{14}^*, & V_{15} - V_{15}^*, & V_{16} \oplus\ V_{16}^*]. \end{array}$$

Let $\mathbf{X}(i)$ and $\mathbf{Y}(i)$ denote the 16-byte input and output, respectively, of the i-th round of SAFER++. As usual in differential cryptanalysis, we assume that all round subkeys are chosen independently and uniformly at random (rather than actually produced by the SAFER++ key schedule from a random user–selected key). A pair (α, β) of *non-zero* 16-byte vectors, considered as the value of $(\Delta\mathbf{X}(1), \Delta\mathbf{Y}(i))$ is an *i-round differential* for SAFER++.

It follows from Proposition 1 in [6] that SAFER++ is a *Markov cipher* as defined in [8], i.e., that the conditional probability

$$P(\Delta\mathbf{Y}(i) = \beta \mid \Delta\mathbf{X}(1) = \alpha,\ \mathbf{X}(1) = \gamma)$$

is independent of γ. This has the consequence that the sequence

$$\Delta\mathbf{X}(1),\ \Delta\mathbf{Y}(1),\ \Delta\mathbf{Y}(2), \ldots, \Delta\mathbf{Y}(i)$$

is a *Markov chain* and that the uniform distribution over non–zero differences is a stationary distribution for this Markov chain, cf. [8].

A sequence $(\alpha, \beta_1, \beta_2, \ldots, \beta_i)$, considered as the value of

$$(\Delta\mathbf{X}(1), \Delta\mathbf{Y}(1), \Delta\mathbf{Y}(2), \ldots, \Delta\mathbf{Y}(i))$$

is called an *i-round characteristic*. The probability

$$P(\Delta\mathbf{Y}(i) = \beta \mid \Delta\mathbf{X}(1) = \alpha)$$

of an i-round differential is just the sum of the probabilities

$$P(\Delta\mathbf{Y}(1) = \beta_1, \ldots, \Delta\mathbf{Y}(i-1) = \beta_{i-1}, \mathbf{Y}(i) = \beta \mid \Delta\mathbf{X}(1) = \alpha)$$

of the i-round characteristics $(\alpha, \beta_1, \ldots, \beta_{i-1}, \beta)$ for all non-zero values of β_1, β_2, …, β_{i-1}. We will refer to these characteristics $(\alpha, \beta_1, \ldots, \beta_{i-1}, \beta)$ as *belonging to the i-round differential* (α, β).

5.2.2 Differential Cryptanalysis of SAFER++

The attack by differential cryptanalysis on an r-round cipher relies on being able to find an $(r-1)$-round differential whose probability is substantially greater than the average probability of such a differential, which is $\frac{1}{2^{128}-1} \approx 2^{-128}$ for a 16-byte block length. The existence of such a differential is generally characterized by the existence of a characteristic belonging to it that itself has a probability greater than this average differential probability. The task of showing the security of r-round SAFER++ with a 128-bit key against differential cryptanalysis is essentially that of showing that there is no $(r-1)$-round characteristic with probability greater than 2^{-128}. An exhaustive study of SAFER++ has shown that all 5-round characteristics have probability significantly smaller than 2^{-128} (but that this is not the case for only 4 rounds). The conclusion is that SAFER++ with six or more rounds (but not fewer) is secure against differential cryptanalysis.

An exhaustive study of SAFER++ for the legacy block size of 64 bits and key size of 128 bits has shown that all 6-round characteristics have probability smaller than 2^{-128} (but that this is not the case for only 5 rounds). The conclusion is that SAFER++ 64-bit legacy version with seven or more rounds (but not fewer) is secure against differential cryptanalysis. The reason for this difference between the 64-bit legacy block size version and 128-bit standard block size version, both of which have a 128-bit user-selected key and thus for security also require that there be no characteristic with probability greater than 2^{-128}, is that the 64-bit legacy block size version has only 8 byte paths (rather than 16) in the key–controlled substitution portion of the encryption round–cf. Figs. 7 and 2. To get the required probability of less than 2^{-128} for every characteristic requires one more round of encryption for the 64-bit legacy block size version.

The invertible linear transformation of the SAFER++ encryption round, which was chosen to ensure that SAFER++ enjoys good diffusion (i.e., to ensure that small changes in round inputs cause large changes in round outputs) also ensures that "differences" similarly propagate and is a main source of the strength of SAFER++ against differential cryptanalysis.

5.3 Strength against Linear Cryptanalysis

Linear cryptanalysis is a general attack against an iterative cipher that was introduced by Matsui in 1993 [13]. Linear cryptanalysis has proved to be a very effective general attack against ciphers in which the round subkeys are inserted by modulo-two addition. However, linear cryptanalysis is in general a weak attack against ciphers in which the round subkeys are inserted by addition modulo a larger modulus [15]. In particular, linear cryptanalysis is weak against all the SAFER-K and SAFER-SK ciphers. With only three rounds, cf. pp. 41-47 in [14], these ciphers have been shown to be secure against linear cryptanalysis, cf. also [15] and [16].

Because SAFER++ employs the same structural properties that make the SAFER-K and SAFER-SK ciphers strongly secure against linear cryptanalysis, it is to be expected that SAFER++ likewise enjoys such security. In fact, SAFER++ is even stronger against linear cryptanalysis than are the SAFER-K and SAFER-SK ciphers. With only two-and-one-half rounds, i.e., up to the input of the invertible linear transformation in the third encryption round, SAFER++ is already secure against linear cryptanalysis for the reasons that we now sketch. We follow here the terminology and notation in [14] and [15].

5.3.1 Imbalance and Homomorphisms

A binary-valued function is *balanced* if it takes on the value 0 for exactly half of its possible arguments and the value 1 otherwise. An *I/O sum* $S^{(i)}$ for the i-th round of an iterative cipher is a modulo-two sum of a balanced binary-valued function f_i of the round input $X(i) = Y(i-1)$ and a balanced binary-valued function g_i of the round output $Y(i)$, i.e.,

$$S^{(i)} = f_i(X(i)) \oplus g_i(Y(i)) = f_i(Y(i-1)) \oplus g_i(Y(i))$$

where $\oplus$ denotes modulo-two addition, i.e., the XOR operation. I/O sums for successive rounds are said to be *linked* if the output function of each I/O sum, except the last, coincides with the input function of the following I/O sum (i.e., $g_i = f_{i+1}$). When $S^{(1)}$, $S^{(2)}$,..., $S^{(\rho)}$ are linked, then their sum is also an I/O sum, namely

$$S^{(1\ldots\rho)} = \bigoplus_{i=1}^{\rho} S^{(i)} = f_1(X(1)) \oplus g_\rho(Y(\rho)),$$

which is called a ρ-round I/O sum .

A homomorphism from a group $(B^n, \oplus)$ onto the group $(B, \oplus)$ is called a *binary-valued homomorphism* for $\otimes$. The binary-valued function f is such a homomorphism if f is not identically zero and if, for all U and V in B^n, $f(U \otimes V) = f(U) \oplus f(V)$. An I/O sum for rounds i to $j (j \geq i)$ is *homomorphic* if the input function is a binary-valued homomorphism for $\otimes_i$ and the output function is a binary-valued homomorphism for $\otimes_{j+1}$.

5.3.2 Linear Cryptanalysis of SAFER++

The *imbalance* $I(V)$ of binary random variable V is the non-negative real number $|2P[V = 0] - 1|$ where $P[V = 0]$ is the probability that V takes on the value 0. The success of the attack by linear cryptanalysis on an r-round iterative cipher depends on being able to find an $r-1$ round homomorphic I/O sum with substantial imbalance. We have been able to prove that Harpes' procedure [14] for finding effective homomorphic I/O sums, *which is the only effective procedure known for finding such sums*, cannot find an I/O sum with non-zero imbalance for one-and-one-half rounds of SAFER++. This strongly suggests that there is no homomorphic I/O sum whatsoever with non-negligible imbalance after only one-and-one-half rounds of SAFER++, i.e., that SAFER++ is already secure against linear cryptanalysis after only two-and-one-half rounds.

5.4 Other Analyses of SAFER++

5.4.1 Weak Keys

It is virtually certain that there are no "weak keys" in SAFER++ for any reasonable definition of this somewhat elusive cryptanalytic concept. The reason for this is the use of biases as described in Section 2.1.3 to "randomize" the round subkeys produced from the user-selected key. In particular, there are no pairs of keys (such as binary complements) that have similar encrypting or decrypting properties. As a result, no precautions must be taken when selecting the input key for SAFER++; the "best" method is simply to choose the key uniformly at random, i.e., by coin-tossing.

5.4.2 Trap Doors

There are no "trap doors" built into SAFER++ that would allow one privy to knowledge of such a trap door to read encrypted traffic illicitly. The best "proof" of this is given in the remark labelled "Transparency" in Section 6 below.

5.5 Previous Cryptanalyses

Although SAFER++ is a new cipher and has not yet been subject to cryptanalysis except by its designers, there have been several cryptanalyses of the previous SAFER family of ciphers. These are given in [3], [6], [7], [10], [11], [12], [14], [15], [16] and [20]. These cryptanalyses are all referred to within this document. The conclusion that one may draw from these cryptanalyses is that the design principles of SAFER++ are sound and ensure that this cipher is cryptographically strong.

6 Advantages and Limitations of SAFER++

The major advantages that would be offered by the choice of SAFER++ for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) are:

- **A proven track record of security**

  Many cryptanalysts in the public arena have attacked the previous ciphers in the SAFER family since their announcement in 1993. The only real weakness found to date, as described above in Section 3, was in the original key schedule which led to a significantly improved key schedule for the SAFER-SK ciphers. In [19], a minor weakness in the 256 bit key schedule of SAFER+ was found that permitted the key to be found (in principle) with slightly less computation than an exhaustive key search. The key schedule for SAFER++ eliminates this minor weakness.

- **Speed and simplicity**

  The simple structure of SAFER++, which is based on byte operations only, makes it easy to implement correctly even by inexperienced programmers. It also makes possible a host of "tricks" that experienced programmers can use to speed up encryption and decryption. We expect that the "optimized" Portable C implementations of SAFER++ included with this report will be significantly accelerated as other programmers concern themselves with the implementation of SAFER++.

- **Transparency**

  The fact that the SAFER++ algorithm uses only well-defined mathematical functions (rather than "random-looking" tables) should convince any fair-minded person that there are no "trapdoors" (i.e., hidden weaknesses) built into this cipher for the benefit of those "in the know." There are simply no suspicious steps anywhere in the algorithm and every step has a clear cryptographic justification.

- **Flexibility of Use**

  SAFER++ offers the same flexibility for usage as does the Data Encryption Algorithm (DEA) of the Data Encryption Standard (DES) [4]. In particular, it can be used in Electronic CodeBook (ECB) mode, in

Cipher Block Chaining (CBC) mode, in Cipher FeedBack (CFB) mode and in Output FeedBack (OFB) mode in exactly the same manner as the DEA. It can also be used in hashing in accordance with the schemes described in [17]. SAFER++ can also be used to generate a Message Authentication Code (MAC) and to produce pseudo–random numbers in essentially the same manner that the DEA can be used for these purposes.

If it were to become desirable, it would be easy to accomodate other key lengths into SAFER++ in addition to those considered in this document (16 bytes and 32 bytes). However, it would not possible to change the block length from its present 16 bytes (or 8 bytes in the legacy version) without a substantial redesign of the algorithm.

- **Flexibility of Environment**

  SAFER++ was designed to implement easily on any platform from an 8-bit processor upwards. The computational efficiency data given in Section 4 are evidence of this flexibility. The facts that only byte operations are performed within the algorithm and that its memory requirements are small make it attractive for use on smart cards with 8-bit processors, in ATM, for HDTV, in B-ISDN, in voice applications, aboard satellites, and in virtually any other environment where encryption might be desirable.

The limitations of SAFER+ are:

- **No proof of complete security**

  As for every cipher today in which the key is used more than once, it must be said and remembered that there is **no proof** that SAFER++ is not vulnerable to some attack of a kind not as yet known.

- **Encryption/Decryption Dissimilarity**

  The fact that SAFER++ is a general substitution/linear–transformation cipher, as was explained in Section 3 above, implies that its decrypting algorithm differs from the encrypting algorithm by more than just a change in the key schedule. However, it should be clear from Sections 2.1.1 and 2.1.2, as well as from Figs. 2 and 3, that there are substantial similarities between the encrypting and decrypting algorithms of

SAFER++ so that one need pay only a small price in software or hardware for implementing both. Moreover, as was discussed in Section 3 above, this dissimilarity of encryption and decryption in SAFER++ was a source of desirable additional freedom in designing this cipher.

7 Intellectual Property Rights

SAFER++ is a completely non-proprietary algorithm. Neither Cylink Corporation nor any of the inventors have retained any intellectual property rights. The C programs included in this report and the submitted diskettes are not copyrighted.

8 Diskettes Submitted with This Proposal

- Discette #1 of 6—Postscript file of cover sheet of this document
- Diskette # 2 of 6—Postscript files of this document.
- Diskette #3 of 6—Portable C Reference Code for SAFER++.
- Diskette #4 of 6—Test vectors for SAFER++; computational efficiency estimates for SAFER++
- Diskette #5 of 6—Optimized 32-bit and 8-bit implementations of SAFER++ for Pentium III platform
- Diskette #6 of 6—Postscript file of Intellectual property statement

References

[1] BLUETOOTH SPECIFICATION Version 1.0B, 29 Nov. 1999, [http://www.bluetooth.com/link/spec/bluetooth_b.pdf].

[2] J. L. Massey, "SAFER K–64: A Byte-Oriented Block-Ciphering Algorithm," pp. 1–17 in *Fast Software Encryption* (Ed. R. Anderson), Lecture Notes in Computer Science No. 809. New York: Springer, 1994.

[3] J. L. Massey, "On The Optimality of SAFER++ diffusion," pp.215–229 in The Second Encryption Standard Candidate Conference,March 22–23,1999,Rome,ITALY.

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

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

[6] J. L. Massey, "SAFER K–64: One Year Later," pp. 212–241 in *Fast Software Encryption II* (Ed. B. Preneel), Lecture Notes in Computer Science No. 1008. New York, Springer 1995.

[7] S. Vaudenay, "On the Need for Multipermutations Cryptanalysis of MD4 and SAFER," pp. 286–297 in *Fast Software Encryption II* (Ed. B. Preneel), Lecture Notes in Computer Science No. 1008. New York, Springer 1995.

[8] 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, 1991.

[9] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," pp. 212–241 in *Advances in Cryptology - CRYPTO '90* (Eds. A. J. Menezes and S. A. Vanstone), Lecture Notes in Computer Science No. 537. Heidelberg and New York: Springer 1990.

[10] S. Murphy, "An Analysis of SAFER," *J. Cryptology*, Vol. 11, No. 4,pp. 235-251, Autumn 1998.

[11] L. R. Knudsen, "A Key-Schedule Weakness in SAFER K–64," in *Advances in Cryptology - CRYPTO '95* (Ed. D. Coppersmith), Lecture Notes in Computer Science No. 963. Heidelberg and New York: Springer, 1995.

[12] L. R. Knudsen and T. A. Berson, "Truncated Differentials of SAFER," pp. 15–26 in *Fast Software Encryption* (Ed. D. Gollmann), Lecture Notes in Computer Science, No. 1039. Heidelberg and New York: Springer, 1996.

[13] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," pp. 386–397 in *Advances in Cryptology-EUROCRYPT '93* (Ed. T. Helleseth), Lecture Notes in Computer Science No. 765. New York: Springer, 1994.

[14] C. Harpes, "Cryptanalysis of Iterated Block Ciphers," ETH Series in Inform. Processing. Konstanz: Hartung-Gorre Verlag, 1996.

[15] C. Harpes, G. C. Kramer and J. L. Massey, "A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-Up Lemma," pp. 24–38 in *Advances in Cryptology-EUROCRYPT'95* (Eds. L. C. Guillou and J.-J. Quisquater), Lecture Notes in Computer Science, No. 921. Heidelberg and New York: Springer, 1995.

[16] S. R. Perkins, "Linear Cryptanalysis of the SAFER K-64 Block Cipher," Semesterarbeit No. 9415, Institut für Signal- und Informationsverarbeitung, ETH Zürich, 15 July 1994.

[17] X. Lai and J. L. Massey, "Hash Functions Based on Block Ciphers," pp. 55–70 in *Advances in Cryptology-EUROCRYPT'92* (Ed. R. A. Rueppel), Lecture Notes in Computer Science, No. 658. Heidelberg and New York: Springer, 1993.

[18] J. L. Massey, G. H. Khachatrian and M. K. Kuregian, Nomination of SAFER+ as Candidate algorithm for the Advanced Encryption Standard (AES), Submission document from Cylink Corporation to NIST, June 1998.

[19] J. Kelsey, B. Schneier and D. Wagner, "Key Schedule Weaknesses in SAFER+" pp.155–167 in The Second Encryption Standard Candidate Conference, March 22-23, 1999, Rome, ITALY.

[20] J. Nakahara, Jr., B. Preneel, and J. Vandewalle, "Linear Cryptanalysis of Reduced-Round Versions of the SAFER Block Cipher Family," Fast software Encryption Workshop 2000, New York, 10–12 April 2000.