MARS - a candidate cipher for AES

Carolynn Burwick Don Coppersmith Edward D'Avignon
Rosario Gennaro Shai Halevi Charanjit Jutla
Stephen M. Matyas Jr. Luke O'Connor Mohammad Peyravian
David Safford Nevenko Zunic

IBM Corporation

June 10, 1998

Abstract

We describe MARS, a shared-key (symmetric) block cipher supporting 128-bit blocks and variable key size. MARS is designed to take advantage of the powerful operations supported in today's computers, resulting in a much improved security/performance tradeoff over existing ciphers. As a result, MARS offers better security than triple DES while running significantly faster than single DES. The current C implementation runs at rates of about 65 Mbit/sec. on a 200 MHz Pentium-Pro, and 85 Mbit/sec. on a 200 MHz PowerPC. In hardware, MARS can achieve a $10\times$ speedup factor. Still, both hardware and software implementations of MARS are remarkably compact, and easily fit on a smartcard and in other limited-resource environments. The combination of high security, high speed, and flexibility, makes MARS an excellent choice for the encryption needs of the information world well into the next century.

AES submission checklist

Items	Addressed in	On page(s)
21-25	Section 2	8-23
26-47	Section 3	23-28
174-188	Section 4	28-55
190,195	Subsection 1.1	4
191,194	Section 5	55 - 58
192	Subsection 3.4	27

Contents

List of Figures

List of Tables

1 Introduction

Shared-key (symmetric) encryption is perhaps the most fundamental cryptographic task. It is used in a large variety of applications, including protection of the secrecy of login passwords, ATM PINS, e-mail messages, video transmissions (such as pay-per-view movies), stored data files, and Internet-distributed digital content. It is also used to protect the integrity of banking and point-of-sale transactions, in key distribution protocols (such as Kerberos), and many other applications.

The current standard for shared-key encryption is the DES cipher, which was developed by IBM in the early 70's [4]. Although DES has provided a secure encryption algorithm for the past 25 years, its block-length and key-length limitations – combined with the advances in computing technology – necessitate the design of a new cipher for use in the next 25 years. In this document we describe a design for a new cipher, MARS, which is well suited for this job.

1.1 The MARS cipher

MARS is a shared-key block cipher, with a block size of 128 bits and a variable key size, ranging from 128 to 1248 bits.[1] It was designed to meet and exceed the requirements for a standard for shared-key encryption in the next few decades. The main theme behind the design of MARS is to get the best security/performance tradeoff by utilizing the strongest tools and techniques available today for designing block ciphers. As a result, MARS provides a very high level of security, combined with much better performance than other existing ciphers.

We estimate that MARS offers better security than triple-DES. In particular, we estimate that all the known cryptanalytical attacks (including linear and differential cryptanalysis) require more data than is available (2^{128}), and hence these attacks are impossible against MARS. Also, the design principles of MARS make it likely that MARS would remain resilient even in the face of new cryptanalytical techniques.

As for efficiency, we estimate that a fully optimized software implementation of MARS can be made to run at rates exceeding 100 Mbit/sec. on the high-end computers available today. We currently have a C implementation which runs at 65 Mbit/sec. on a 200 MHz Pentium-Pro and 85 Mbit/sec. on a 200 MHz PowerPC, and dedicated hardware can achieve an additional 10$\times$ speedup factor.

1.2 Rationale and design choices

Below, we explain the rationale behind the design of MARS and discuss various choices made in this design. Throughout the design process we capitalized on the following principles:

Choice of operations. MARS is designed to be used in the computer environments of today and tomorrow. We thus use the full menu of "strong operations" supported in modern computers to achieve better security properties. This approach enables us to get a much better security-

[1] The main objective in allowing key-lengths beyond 256 bits is convenience (rather than security). For example, a key which is derived from a Diffie-Hellman exchange is usually much longer than 256 bits.

per-instruction ratio for our software implementation of MARS than is possible for existing ciphers.

In particular, the design of MARS takes full advantage of the ability of today's computers to perform fast multiplications and data-dependent rotations. We discuss these operations (and their combination) in more detail in Subsection 1.2.1.

The cipher's structure. Two decades of experience in cryptanalysis has taught us that different parts in a cipher play very different roles in assuring the security of the cipher. In particular, it appears that the top and bottom rounds in the cipher usually have a different role than the middle rounds in protecting against cryptanalytical attacks.

We therefore designed MARS using a mixed structure, where the top and bottom rounds are designed differently than the middle ones. More on that in Subsection 1.2.2.

Analysis. An important aspect of MARS is that its components are designed to permit extensive analysis. In every step of the design, we refrained from using operations and structures which seemed "too hard to analyze". Instead, we insisted on providing an analysis for every aspect of the cipher, and we used this analysis to guide us through many of the design choices.

Some choices that we made in the design of MARS include:

Working with 32 bit words. Since most computers today (and in the near future) use word-size of 32 bits, all the operations in MARS are applied to 32-bit words. At the current state of the technology, this choice provides a good tradeoff between the ability to run the algorithm on computers which are available today (as well as on legacy systems and even 8-bit processors), and the ability to take advantage of larger word-size in future architectures.

Type-3 Feistel network. Since MARS has a block length of 128 bits and word-size of 32 bits, it follows that each block consists of four words. Among the various network-structures which are capable of handling four words in a block, it seems that a type-3 Feistel network provides the best tradeoff between speed, strength and suitability for analysis.

A type-3 Feistel network consists of many rounds, where in each round one data word (and a few key words) are used to modify all the other data words. Compared with a type-1 Feistel network (where in each round one data word is used to modify one other data word), this construct provides much better diffusion properties with only a slightly added cost. Hence, fewer rounds can be used to achieve the same strength.

Additionally, a type-3 Feistel network has advantages over structures in which several data words are used "at once" to modify other data words, in that these structures are typically much harder to analyze (and hence, much more prone to design errors). The reason is that in such structures the analysis must take into account all the possible combinations of values for the input data words, which quickly leads to unmanageable complexity.

Symmetry of encryption and decryption. We designed MARS to be as secure against chosen ciphertext attacks as against chosen plaintext attacks. This dictates making the cipher very symmetric, so the last half of the rounds are almost a "mirror image" of the first half.

1.2.1 Choice of operations

As we explained above, the MARS cipher uses a variety of operations (on 32-bit words). Specifically, it combines exclusive-ors (xors), additions, subtractions, table look-ups, multiplications, and both fixed and data-dependent rotations. We discuss these operations and their use in MARS below.

Additions, subtractions and xors. These are the simplest operations, which are used to "mix together" data values (and key values). These operations are very fast in either software or hardware, and typically are not meant to provide much "cryptographic strength". Throughout the cipher we interleave xors with additions and subtractions to ensure that the operations in the cipher do not commute with each other.

Table look-up. Table look-up operations provide the basis for the security of DES, as well as of many other ciphers (e.g. [1]). MARS uses a single table of 512 32-bit words, called *the S-box*. Sometimes the S-box is viewed as two tables, each of 256 entries.

In principle, a carefully chosen S-box can provide good resistance against linear and differential attacks, as well as good avalanche of data and key bits. A drawback of using S-box lookups, however, is that it is relatively slow for software implementations. In a word-oriented cipher like MARS, a typical S-box lookup operation takes three instructions (one to copy the source word into an index register, one to mask out the high order bits of the index, and one to access the table itself). Also, a large S-box may take up a considerable amount of space in hardware implementations.

Another problem is that the index into the table consists of just a few bits (otherwise the table would be too large). Hence, in order to use all the bits of a data word, one needs to do several S-box lookups, which slows the cipher even further.

Therefore, S-box lookups are used in MARS only in places where fast avalanche of the key bits is needed, or in places where it suffices to use only a few bits of the data word (since other bits are "already taken care of" by other means).

Fixed rotations. Rotations by fixed amounts are mainly used in conjunction with the software implementation to get the data bits to places where we can use them (e.g., in order to use the high order bits of a data word as an index to the S-box).

Data-dependent rotations. Data-dependent rotations were first used for encryption in a cipher developed by Becker in IBM in the late 1970's [2] (and later were used by Madryiga [8] in his cipher). This operation gained recognition in recent years after it was used by Rivest as the main building block for the RC5 cipher [12].

Data dependent rotations can be performed quickly in software and hardware. Combined with arithmetic operations (such as addition), this operation is very effective against linear cryptanalysis. Also, when carefully used in a cipher it can be made effective against differential cryptanalysis.

One problem with data-dependent rotations is that specifying a rotation amount for a w-bit word only takes $\log w$ bits. Hence, while the result of this operation depends on all the bits

in one operand, it only depends on very few bits in the other. This may lead to differential weaknesses, as was recently demonstrated by Biryukov and Kushilevitz [5].

In MARS we make extensive use of data-dependent rotations, but we solve the problem mentioned above by combining these operations with multiplications, as described next.

Multiplications. Multiplications were used for encryption in the IDEA cipher and its variants [7]. However, until recently multiplications were considered prohibitively expensive for fast encryption. This was true since old machines took many cycles to perform a single multiplication operation.[2] Today, this is no longer the case, as essentially all modern architectures (including PowerPC, Pentium-Pro, Alpha, Ultra-SPARC, and others) support a multiply instruction which takes about two cycles to complete.[3]

Another reason that multiplications were considered so expensive is that IDEA and its variants insisted on performing multiplications in the field of integers modulo $2^{16}+1$. Hence, each multiplication operation had to be coded in software as a sequence of operations, including a "native multiplication" modulo 2^{16} and a few additional operations.

In MARS, we use "native multiplications" modulo 2^{32}, in conjunction with data-dependent rotations, to obtain very high security. The main cryptographic strength of multiplication modulo 2^{32} is in the high-order bits of the product, as each of these bits depends on almost all the bits in the operands in a non-linear fashion. Also, these bits have excellent differential properties. Therefore, in MARS we use the high order bits of the product to specify the rotation amounts in the data-dependent rotation operations. This novel combination is what gives MARS its good resistance to differential cryptanalysis.

It should be noted that multiplication is still a rather expensive operation: even on modern processors it takes about twice the time of other operations, and in hardware it is even more costly. Hence we use this operation in moderation: in the entire cipher we only perform 16 multiplications (compared to 32 multiplications in IDEA). As a result, we estimate that the multiplications only take about 30% of the time and less than 20% of the area in a typical hardware implementation of MARS, and they take less than 10% of the time in our software implementation.

A final point about the usage of multiplications in MARS has to do with our ability to analyze them: Analyzing a multiplication of two data words turns out to be a very hard task. As a result, in MARS we only multiply data words by key words. In addition, in the key expansion process we check the key words used for multiplication to avoid some "obviously weak" words (such as $1, -1$, or even integers). Restricting ourselves to data-key multiplications enables us to provide a substantial analysis for this operation, which we use to analyze the security of the cipher.

[2] Multiplication took at least 50 cycles in the original SPARC architecture, about 40 in the Intel 486, and about 10 in the Intel Pentium.

[3] On some architectures, the multiplication instruction takes longer by itself, but it can be pipelined with other instructions, resulting in an effective time of two cycles per operation.

1.2.2 Using a mixed structure

Many cryptanalytical techniques (including linear and differential cryptanalysis) treat the top and bottom rounds of the cipher differently than the middle rounds. Typically, these techniques begin by guessing several key bits, hence "stripping out" some of the top/bottom rounds of the cipher, and then mounting the cryptanalytical attack against the remaining rounds. This suggests that the top and bottom rounds of the cipher play a different role than the middle rounds in protecting against cryptanalytical attacks. Specifically, for these rounds we care more about fast avalanche of the key bits (which is a combinatorial property) than about resistance to cryptanalysis. Theoretical evidence for the different role played by the top and bottom rounds can be found in the Naor-Reingold constructions [11], in which a "cryptographic core" is wrapped with some non-cryptographic mixing.

Therefore, in the design of MARS the middle rounds are viewed as the "cryptographic core" and are designed differently than the top and bottom rounds, which are viewed as "wrapper layers". Specifically, the wrapper layers consist of first adding in key words, and then performing several rounds of (unkeyed) S-box based mixing, providing rapid avalanche of key bits. The core layer consists of several rounds of keyed transformation which involves a combination of S-box lookups, multiplications and data-dependent rotations to get good resistance to cryptanalytical attacks.

Another advantage of this mixed structure is that it is likely to provide better resistance against new (yet undiscovered) cryptanalytical techniques. Namely, a cipher consisting of two radically different structures is more likely to be resilient to new attacks than a homogeneous cipher, since in order to take advantage of a weakness in one structure one has to propagate this weakness through the other structure. Viewed in this light, the mixed structure can be thought of as an "insurance policy" to protect the cipher against future advances in cryptanalytical techniques.

1.3 Organization

The rest of the document is organized as follows: In Section 2, we describe the cipher using text, figures and pseudo-code. This section covers the requirements in Section 2.B.1 in the checklist (Items 21 through 25). Section 3 describes the computational efficiency of the cipher, and describe speed measurements and speed estimates for various implementations. This section covers the requirements in Section 2.B.2 in the checklist (Items 26 through 47). Section 4 contains a statement of the expected strength and analysis of the algorithm, to meet the requirements in Sections 2.B.4 and 2.B.5 in the checklist (Items 174 through 188). Finally, in Section 5 we discuss some other issues related to the cipher, such as its usage in standard modes and possible extensions. This section covers the requirements in Items 191 and 193 in Section 2.B.6 of the checklist.

2 Algorithm Specifications (2.B.1)

MARS takes as input (and produces as output) four 32-bit data words. The cipher itself is word-oriented, in that all the internal operations are performed on 32-bit words, and hence the internal structure is endian-neutral (i.e., the same code works on both little-endian and big-endian ma-

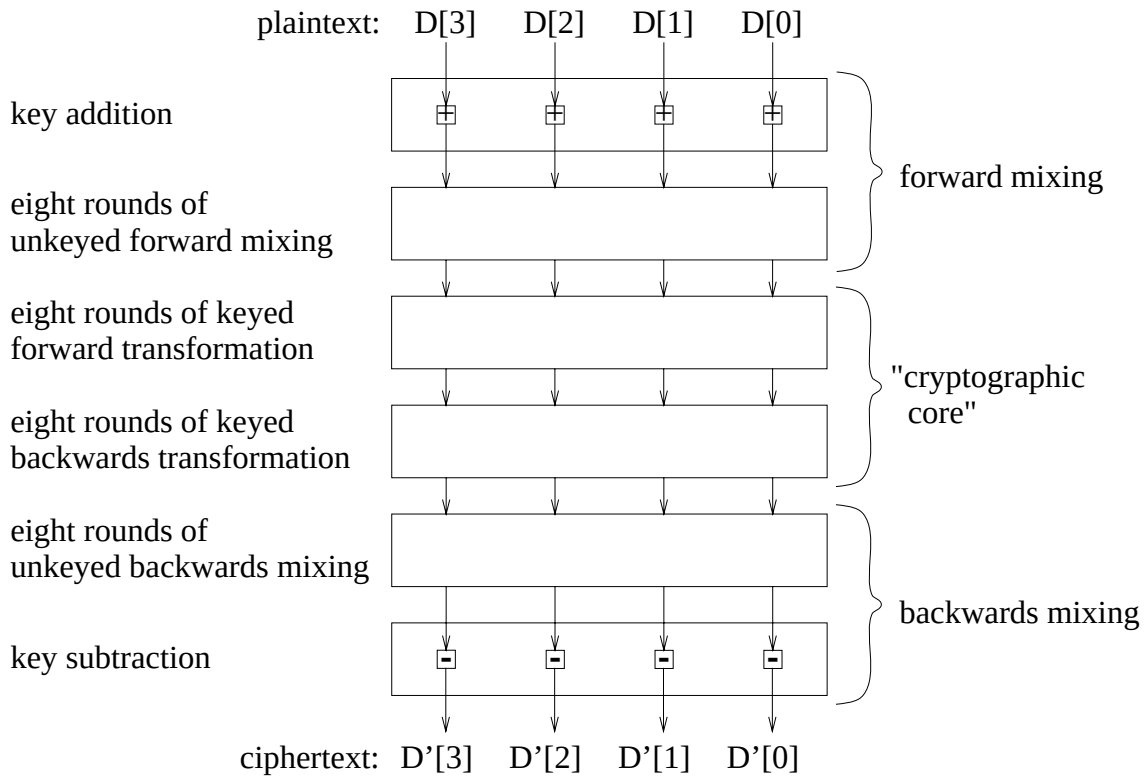

Figure 1: High-level structure of the cipher

chines). When the input (or output) of the cipher is a byte stream, we use *little endian* byte ordering to interpret each four bytes as one 32-bit word.

2.1 High level structure

The general structure of the cipher is depicted in Figure 1. The cipher consists of a "cryptographic core" of keyed transformation, which is wrapped with two layers providing rapid key avalanche.

- The first phase provides rapid mixing and key avalanche, to frustrate chosen-plaintext attacks, and to make it harder to "strip out" rounds of the cryptographic core in linear and differential attacks. It consists of addition of key words to the data words, followed by eight rounds of S-box based, unkeyed type-3 Feistel mixing (in "forward mode").
- The second phase is the "cryptographic core" of the cipher, consisting of sixteen rounds of keyed type-3 Feistel transformation. To ensure that encryption and decryption have the same strength, we perform the first eight rounds in "forward mode" while the last eight rounds are performed in "backwards mode".
- The last phase again provides rapid mixing and key avalanche, this time to protect against chosen-ciphertext attacks. This phase is essentially the inverse of the first phase, consisting of eight rounds of the same type-3 Feistel mixing as in the first phase (except in "backwards mode"), followed by subtraction of key words from the data words.

Below we describe the cipher in details. In this description we use the following notations:

- $D[\,]$ is an array of 4 32-bit data words. Initially D contains the plaintext words, and at the end of the encryption process it contains the ciphertext words.
- $K[\,]$ is the expanded key array, consisting of 40 32-bit words.
- $S[\,]$ is an S-box, consisting of 512 32-bit words. Below we also denote the first 256 entries in S by $S0$ and the last 256 entries by $S1$.

All the arrays below are 0-based (which means, for example, that the four words in $D[\,]$ are indexed $D[0]$ through $D[3]$).

2.2 Phase one: forward mixing

In this phase we first add a key word to each data word, and then perform eight rounds of unkeyed type-3 Feistel mixing, combined with some additional mixing operations. In each round we use one data word (called the source word) to modify the other three data words (called the target words). We view the four bytes of the source word as indices into two S-boxes, $S0$ and $S1$, each consisting of 256 32-bit words, and xor or add the corresponding S-box entries into the other three data words.

If we denote the four bytes of the source words by $b0, b1, b2, b3$ (where $b0$ is the lowest byte and $b3$ is the highest byte), then we use $b0, b2$ as indices into the S-box $S0$ and $b1, b3$ as indices into the S-box $S1$. We first xor $S0[b0]$ into the first target word, and then add $S1[b1]$ to the same word. We also add $S0[b2]$ to the second target word and xor $S1[b3]$ to the third target word. Finally, we rotate the source word by 24 positions to the right.

For the next round we rotate the four words, so that the current first target word becomes the next source word, the current second target word becomes the next first target word, the current third target word becomes the next second target word, and the current source word become the next third target word.

In addition, after each of four specific rounds we add one of the target words back into the source word. Specifically, after the first and fifth rounds we add the third target word back into the source word, and after the second and sixth round we add the first target word back into the source word. The reasons for these extra mixing operations are to eliminate some easy differential attacks against the mixing phase (see Subsection 4.2.4), to break the symmetry in the mixing phase and to get faster avalanche. The forward mixing phase is depicted in Figure 2.

2.3 Main keyed transformation

The "cryptographic core" of the MARS cipher is a type-3 Feistel network, consisting of sixteen rounds. In each round we use a keyed E-function (E for expansion) which is based on a novel combination of multiplication, data-dependent rotations, and an S-box lookup. This function takes

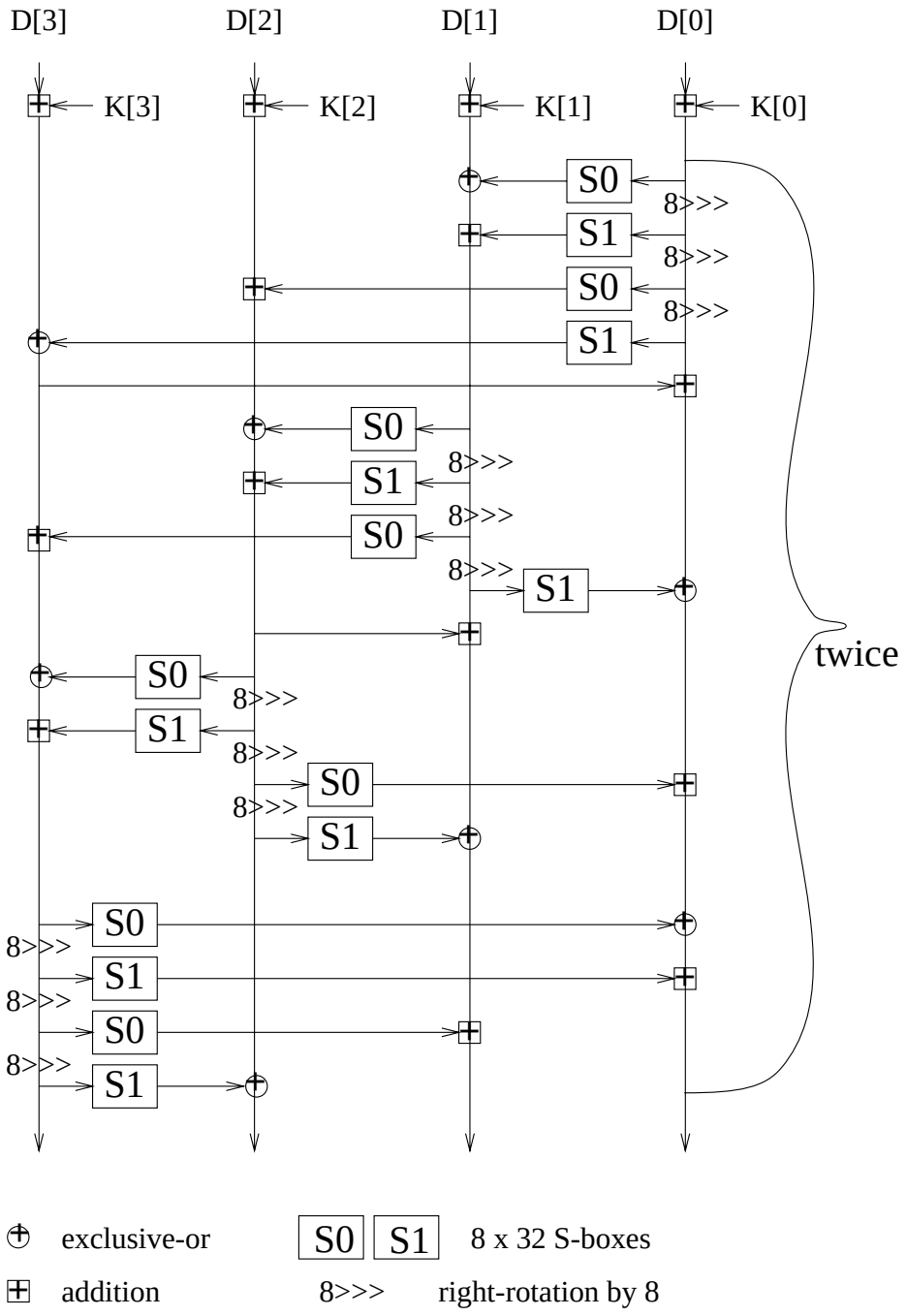

Figure 2: Structure of the forward mixing phase

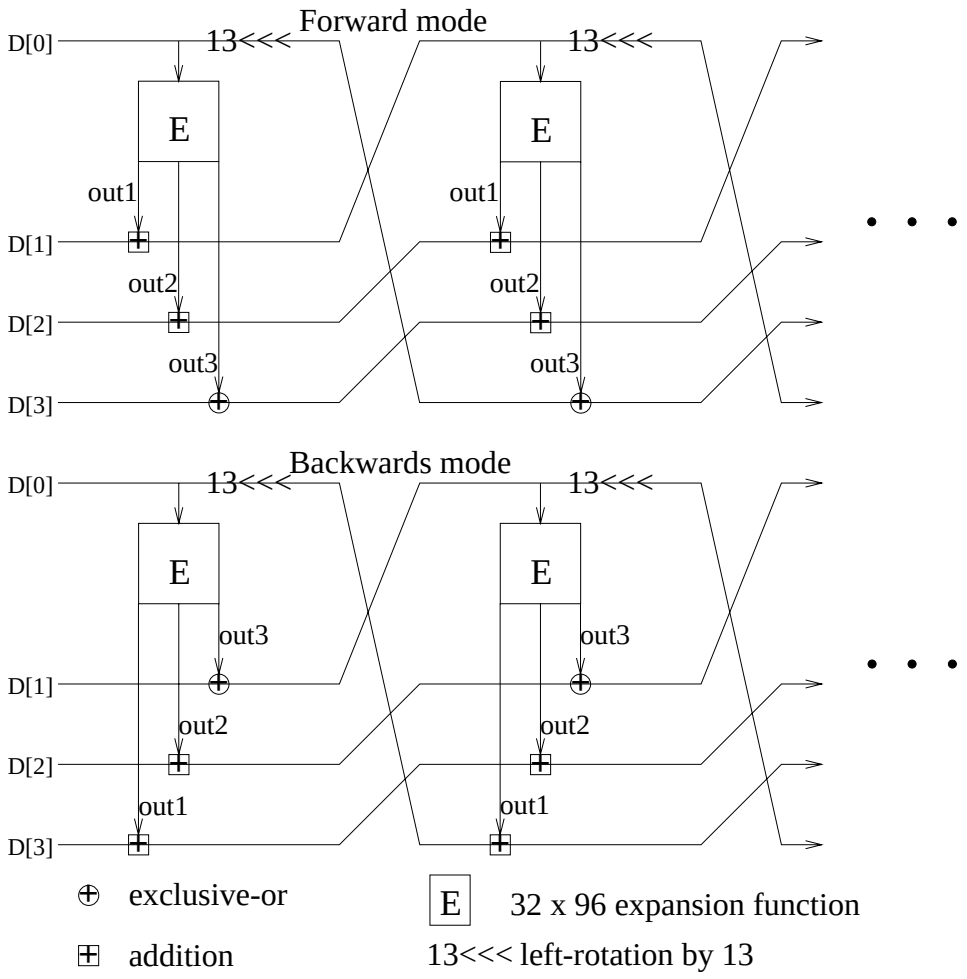

Figure 3: The type-3 Feistel network of the main keyed transformation.

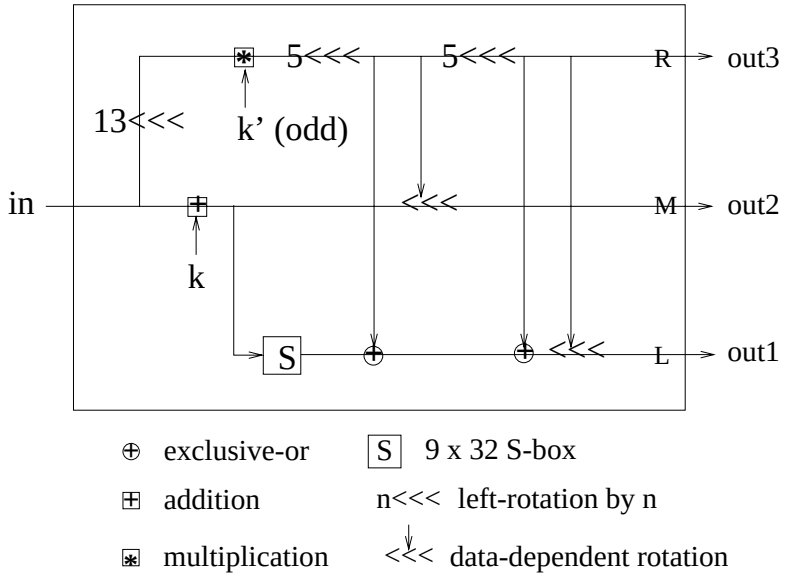

Figure 4: The E-function of the main keyed transformation

as input one data word and returns three data words as output. The structure of the Feistel network is depicted in Figure 3 (for a different picture see also Figure 8), and the E-function itself is diagrammed in Figure 4. In each round we use one data word as the input to the E-function, and the three output words from the E-function are added or xored to the other three data words. In addition, the source word is rotated by 13 positions to the left.

To ensure that the cipher has the same resistance to chosen ciphertext attacks as it has for chosen plaintext attacks, the three outputs from the E-function are used in a different order in the first eight rounds than in the last eight rounds. Namely, in the first eight rounds we add the first and second outputs of the E-function to the first and second target words, respectively, and xor the third output into the third target word. In the last eight rounds, we add the first and second outputs of the E-function to the third and second target words, respectively, and xor the third output into the first target word.

The E-function. The E-function takes as input one data word and uses two more key words to produce three output words. In this function we use three temporary variables, denoted below by L, M and R (for left, middle and right). Below we also refer to these variables as the three "lines" in the function.

Initially, we set R to hold the value of the source word rotated by 13 positions to the left, and we set M to hold the sum of the source word and the first key word. We then view the lowest nine bits of M as an index to a 512-entry S-box S (which is obtained by concatenating $S0$ and $S1$ from the mixing phase), and set L to hold the value of the corresponding S-box entry.

We then multiply the second key word (constrained to contain an odd integer) into R and then rotate R by 5 positions to the left (so the 5 highest bits of the product becomes the 5 lowest bits of R after the rotation). Then we xor R into L, and also view the five lowest bits of R as a rotation amount between 0 and 31, and rotate M to the left by this amount. Next, we rotate R by 5 more positions to the left and xor it into L. Finally, we again view the five lowest bits of R as a rotation amount and rotate L to the left by this amount. The first output word of the E-function is L, the second is M and the third is R.

Design rationale. In the design of the E-function we tried to combine the different operations in a way that would maximize the advantages from each. Some properties of this function which are worth noting are the following:

- Recall that when we multiply two words, the lower bits of the input word have larger effect on the product than the higher bits. Thus, we arrange it so that bits which are *not fed as input to the S-box* will be the lowest bits in the data word which is being multiplied. The amount of rotation (13 bits) was set to maximize the resistance of the E-function to differential attacks. See Subsection 4.2 for details.

  Also, since the internal structure of the E-function is very sensitive to the location of the input bits, it makes sense to apply a constant rotation to the data lines, so as to make it hard for an attacker to maintain a consistent behavior across rounds. Since we use a rotation of the source word by 13 inside the E-function, we can get a rotation by 13 of the corresponding data line "for free".

- Recall also that when we multiply two words, the most significant bits in the product are the "stronger bits" since they are affected by almost all the input bits. In the combination of the multiplication and the data-dependent rotation, we therefore arrange it so that these "strong bits" are used to determine the amount of the data-dependent rotation.

- Since the E-function is supposed to approximate a pseudo-random function, we would like to make the three lines of the function as "independent of each other" as possible. We thus use very little interaction between the data in the three lines. This also helps to avoid unwanted cancellations and makes it harder to obtain a linear approximation of one line in terms of another.

  Where we do mix the lines – in the xors of Line R into Line L – we xor the input word twice and have a fixed rotation by five between these two operations (so, for example, the effects of these xor operations on the parity of Line L cancel each other).

- Still trying to guarantee some measure of "independence" between the data lines, we make sure that the value of one line never completely determines the value of another line. Indeed, the relative entropy of any two lines is at least 9 bits (of lines L, R), and gets as high as 32 bits (of lines R, M).

- Since we view Line M as the weakest output of the E-function (as it only carries the sum of the input and a key word, rotated by some amount), we put it as the middle output line. This way, it never affects the next data line which is used as a source, but rather a data line which is used further down in the encryption process.

2.4 Phase three: backwards mixing

The backwards mixing phase is the same as the decryption of the forward mixing phase, except that the data words are processed in different order. Namely, if we fed the output from the forward unkeyed mixing into the input of the backwards unkeyed mixing in reverse order (i.e., output $D[3]$ goes to input $D[0]$, output $D[2]$ to input $D[1]$, etc.) then these two phases would cancel each other.

As in the forward mixing, here too we use in each round one source word to modify the other three target words. Denote the four bytes of the source words by $b0, b1, b2, b3$ as before. We use $b0, b2$ as indices into the S-box $S1$ and $b1, b3$ as indices into the S-box $S0$. We xor $S1[b0]$ into the first target word, subtract $S0[b3]$ from the second data word, subtract $S1[b2]$ from the third target word and then xor $S0[b1]$ also into the third target word. Finally, we rotate the source word by 24 positions to the left.

For the next round we rotate the four words, so that the current first target word becomes the next source word, the current second target word becomes the next first target word, the current third target word becomes the next second target word, and the current source word become the next third target word.

Also, before each of four specific rounds we subtract one of the target words from the source word: before the fourth and eighth rounds we subtract the first target word from the source word, and before the third and seventh round we subtract the third target word from the source word. The backwards mixing phase is depicted in Figure 5.

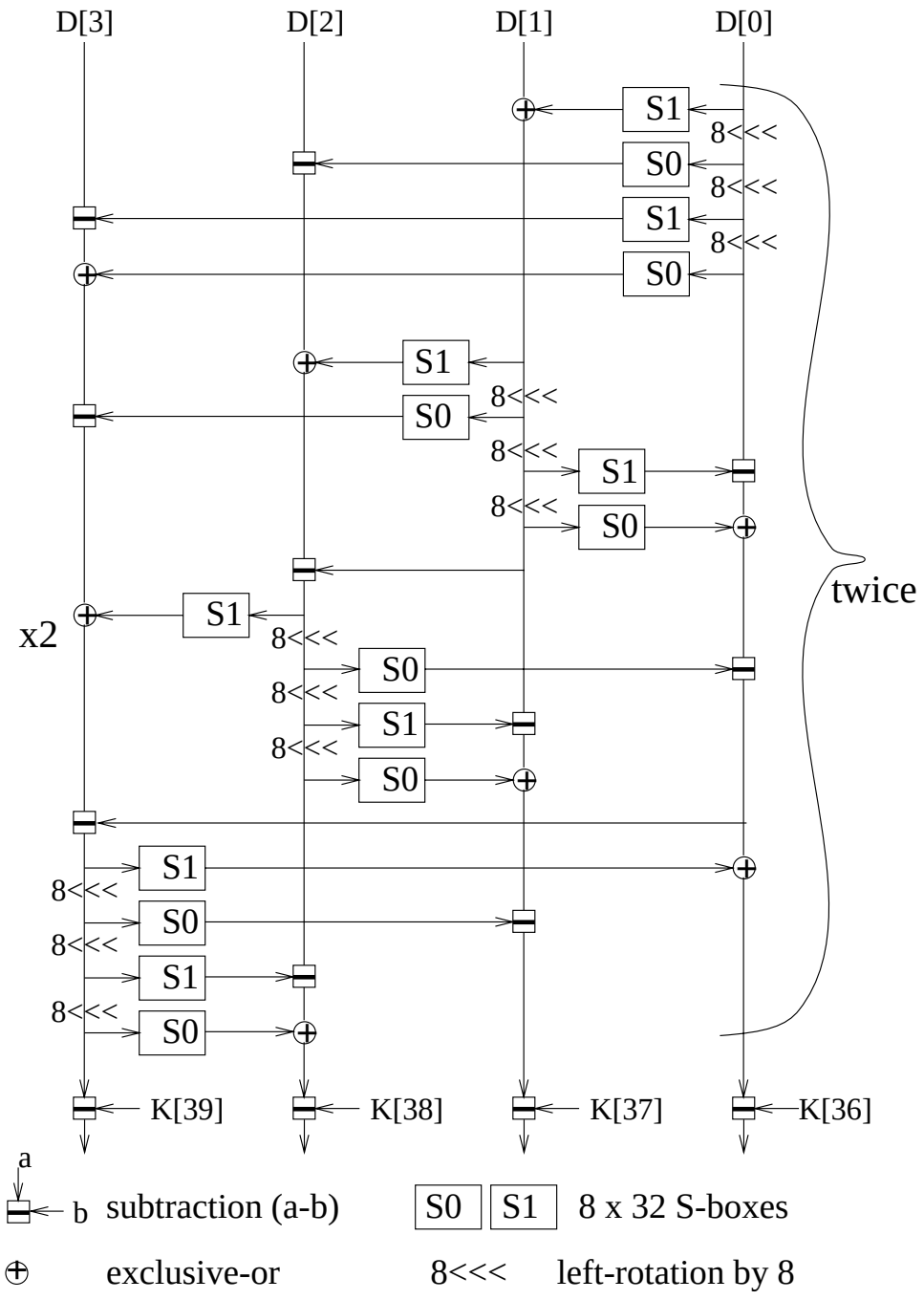

Figure 5: Structure of the backwards mixing phase

2.5 Pseudo-code

Below we describe the cipher using pseudo-code. In this description we use the following notations: The operations used in the cipher are applied to 32-bit words, which are viewed as unsigned integers. We number the bits in each word from 0 to 31, where bit 0 is the least significant (or lowest) bit, and bit 31 is the most significant (or highest) bit.

We denote by $a \oplus b$ a bitwise exclusive-or of the two words a and b, $a \vee b$ and $a \wedge b$ denote bitwise OR and bitwise AND, respectively. We denote by $a + b$ addition modulo 2^{32}, by $a - b$ subtraction modulo 2^{32}, and by $a \times b$ multiplication modulo 2^{32}.

Also, $a \lll b$ and $a \ggg b$, respectively, denote cyclic rotations of the 32-bit word a by b positions to the left and right. In a left rotation by b, the bit in location i is moved to location $i + b \mod 32$ (for example, the lowest bit is moved from location 0 to location b). Similarly, in a right rotation by b the bit in location i is moved to location $i - b \mod 32$.

Finally, if $x_1, \ldots x_n$ are 32-bit words, we use the notation $(x_n, \ldots, x_2, x_1) \leftarrow (x_1, \ldots, x_3, x_2)$ for n-wise swap operation. For example, $(D[3], D[2], D[1], D[0]) \leftarrow (D[0], D[3], D[2], D[1])$ denotes a rotation of the 4-word array $D[\,]$ by one word to the right.

Remark. Notice that the pseudo-code below follows a somewhat different style than Figures 2 and 5. Specifically, to make the pseudo-code shorter we implement the eight mixing rounds in a loop.

E-function(input: $in, key1, key2$)

1. // we use three temporary variables, L, M, R
2. $M = in + key1$ //add first key word
3. $R = (in \lll 13) \times key2$ // multiply by 2nd key word, *which must be odd*
4. $i =$ lowest 9 bits of M
5. $L = S[i]$ // S-box lookup
6. $R = R \lll 5$
7. $r =$ lowest 5 bits of R // these bits specify rotation amount
8. $M = M \lll r$ // 1st data-dependent rotation
9. $L = L \oplus R$
10. $R = R \lll 5$
11. $L = L \oplus R$
12. $r =$ lowest 5 bits of R // these bits specify rotation amount
13. $L = L \lll r$ // 2nd data-dependent rotation
14. output (L, M, R)

MARS-encrypt(input: $D[\,], K[\,]$)

Phase (I): Forward mixing

1. // First add subkeys to data
2. for $i = 0$ to 3 do
3. $\quad D[i] = D[i] + K[i]$
4. // Then do eight rounds of forward mixing
5. for $i = 0$ to 7 do // use $D[0]$ to modify $D[1], D[2], D[3]$
6. $\quad$ // four S-box look-ups
7. $\quad D[1] = D[1] \oplus S0[$ low byte of $D[0]\ ]$
8. $\quad D[1] = D[1] + S1[$ 2nd byte of $D[0]\ ]$
9. $\quad D[2] = D[2] + S0[$ 3rd byte of $D[0]\ ]$
10. $\quad D[3] = D[3] \oplus S1[$ high byte of $D[0]\ ]$
11. $\quad$ // and rotation of the source word to the right
12. $\quad D[0] = D[0] \gg 24$
13. $\quad$ // followed by additional mixing operations
14. $\quad$ if $i = 0$ or 4 then
15. $\quad\quad D[0] = D[0] + D[3]$ // add $D[3]$ back to the source word
16. $\quad$ if $i = 1$ or 5 then
17. $\quad\quad D[0] = D[0] + D[1]$ // add $D[1]$ back to the source word
18. $\quad$ // rotate $D[\ ]$ by one word to the right for next round
19. $\quad (D[3], D[2], D[1], D[0]) \leftarrow (D[0], D[3], D[2], D[1])$
20. end-for

Phase (II): Keyed transformation

21. // Do 16 rounds of keyed transformation
22. for $i = 0$ to 15 do
23. $\quad (out1, out2, out3) =$ E-function$(D[0], K[2i+4], K[2i+5])$
24. $\quad D[0] = D[0] \ll 13$
25. $\quad D[2] = D[2] + out2$
26. $\quad$ if $i < 8$ then // first 8 rounds in forward mode
27. $\quad\quad D[1] = D[1] + out1$
28. $\quad\quad D[3] = D[3] \oplus out3$
29. $\quad$ else // last 8 rounds in backwards mode
30. $\quad\quad D[3] = D[3] + out1$
31. $\quad\quad D[1] = D[1] \oplus out3$
32. $\quad$ end-if
33. $\quad$ // rotate $D[\ ]$ by one word to the right for next round
34. $\quad (D[3], D[2], D[1], D[0]) \leftarrow (D[0], D[3], D[2], D[1])$
35. end-for

Phase (III): Backwards mixing

36. // Do eight rounds of backwards mixing
37. for $i = 0$ to 7 do
38. $\quad$ // additional mixing operations
39. $\quad$ if $i = 2$ or 6 then

40. $D[0] = D[0] - D[3]$ // subtract $D[3]$ from source word
41. if $i = 3$ or 7 then
42. $D[0] = D[0] - D[1]$ // subtract $D[1]$ from source word
43. // four S-box look-ups
44. $D[1] = D[1] \oplus S1[$ low byte of $D[0]\ ]$
45. $D[2] = D[2] - S0[$ high byte of $D[0]\ ]$
46. $D[3] = D[3] - S1[$ 3rd byte of $D[0]\ ]$
47. $D[3] = D[3] \oplus S0[$ 2nd byte of $D[0]\ ]$
48. // and rotation of the source word to the left
49. $D[0] = D[0] \lll 24$
50. // rotate $D[\ ]$ by one word to the right for next round
51. $(D[3], D[2], D[1], D[0]) \leftarrow (D[0], D[3], D[2], D[1])$
52. end-for
53. // Then subtract subkeys from data
54. for $i = 0$ to 3 do
55. $D[i] = D[i] - K[36 + i]$

2.5.1 Decryption

The decryption process is the inverse of the encryption process. The code for decryption is similar (but not identical) to the code for encryption. We provide a pseudo-code for decryption in Appendix B.

2.6 S-box design

In the design of the S-box S, we generated the entries of S in a "pseudorandom fashion" and tested that the resulting S-box has good differential and linear properties. The "pseudorandom" S-boxes were generated by setting for $i = 0 \ldots 102,\ j = 0 \ldots 4$, $S[5i + j] = \text{SHA-1}(i \mid c1 \mid c2 \mid c3)_j$ (where SHA-1$(\cdot)_j$ is the j'th word in the output of SHA-1). Here we view i as a 32-bit unsigned integer, and $c1, c2, c3$ are some fixed constants. In our implementation we set $c1 =$ `0xb7e15162`, $c2 =$ `0x243f6a88` (which are the binary expansions of the fractional parts in e, π, respectively) and we varied $c3$ until we found an S-box with good properties. We view SHA-1 as an operation on byte-streams, and use little-endian convention to translate between words and bytes.

The properties of the S-box which we tested are the following:

Differential properties. We require that the S-box has the following properties:

(1) The S-box does not contain the all-zero or the all-one word.

(2) Within each of the two S-boxes $S0, S1$, every two entries differ in at least three of the four bytes. (We note that it is very unlikely that a random S-box will have this property, and so we first "fix" the S-box by modifying one of the entries in each pair that violates this condition).

(3) S does not contain two entries $S[i], S[j] (i \neq j)$ such that $S[i] = S[j], S[i] = \neg S[j]$ or $S[i] = -S[j]$.

(4) S has $\binom{512}{2}$ distinct xor-differences and $2 \times \binom{512}{2}$ distinct subtraction-differences.

(5) Every two entries in S differ by at least four bits.

Linear properties. We try to minimize the following quantities:

(6) Parity bias: $\left|\Pr_x[\text{parity}(S[x]) = 0] - \frac{1}{2}\right|$. We require that the parity bias of S be at most $1/32$.

(7) Single-bit bias: $\forall\ j,\ \left|\Pr_x[S[x]_j = 0] - \frac{1}{2}\right|$. We require that the single-bit bias of S be at most $1/30$.

(8) Two consecutive bits bias: $\forall\ j,\ \left|\Pr_x[S[x]_j \oplus S[x]_{j+1} = 0] - \frac{1}{2}\right|$. We require that the two-bit bias of S be at most $1/30$.

(9) Single-bit correlation: $\forall\ i, j,\ \left|\Pr_x[S[x]_j = x_i] - \frac{1}{2}\right|$. We minimize this quantity over all the S-boxes that satisfy the conditions 1-8.

The threshold values in Conditions 6-8 above were set experimentally. The reason for the different treatment of the single-bit correlation is that its value is usually larger than the other quantities.

We generated the S-box as follows: We went over possible values of $c3$ in increasing order, starting from $c3 = 0$. For each value, we generated the S-box, and then "fixed it" by going over all the pairs (i, j) of entries in $S0, S1$ in lexicographic order, and checking if the difference $S[i] \oplus S[j]$ has two or more zero bytes. Whenever we found a difference with two or more zero bytes, we replaced $S[i]$ with $3 \cdot S[i]$ and moved on to the next i. After the "fixing", we tested the S-box again to verify that it satisfies all the Conditions (1)-(8) above, and we calculated the single-bit correlation bias (from Item (9) above). Our program ran for about a week, going over roughly 2^{26} possible values for $c3$. The value of $c3$ which minimized the single-bit correlation bias was $c3 =$ `0x02917d59`. The resulting S-box is presented in Appendix A. This S-box has parity bias 2^{-7}, single-bit bias at most 1/30, two consecutive bit bias at most 1/32, and single-bit correlation bias less than 1/22.

2.7 Key expansion

The high-level structure of the key expansion routine is depicted in Figure 6. This procedure expands a given key array $k[\,]$ (which consists of n 32-bit words, where n is any number between 4 and 39) into an array $K[\,]$ of 40 words. We note that the original key is not required to have any structure (in particular, the key does not include any parity bits). In addition, the key expansion procedure also guarantees that the key words which are used for multiplication in the encryption procedure have the following properties

- The two lowest bits in a key word which is used for multiplication are set to 1.
- None of these key words contains either ten consecutive 0's or ten consecutive 1's.

The procedure consists of four steps

1. Initially, the original key material is expanded using a simple linear transformation. Specifically, given an n-word array $k[\,]$, we first initialize a 47-word temporary array $T[\,]$, which we

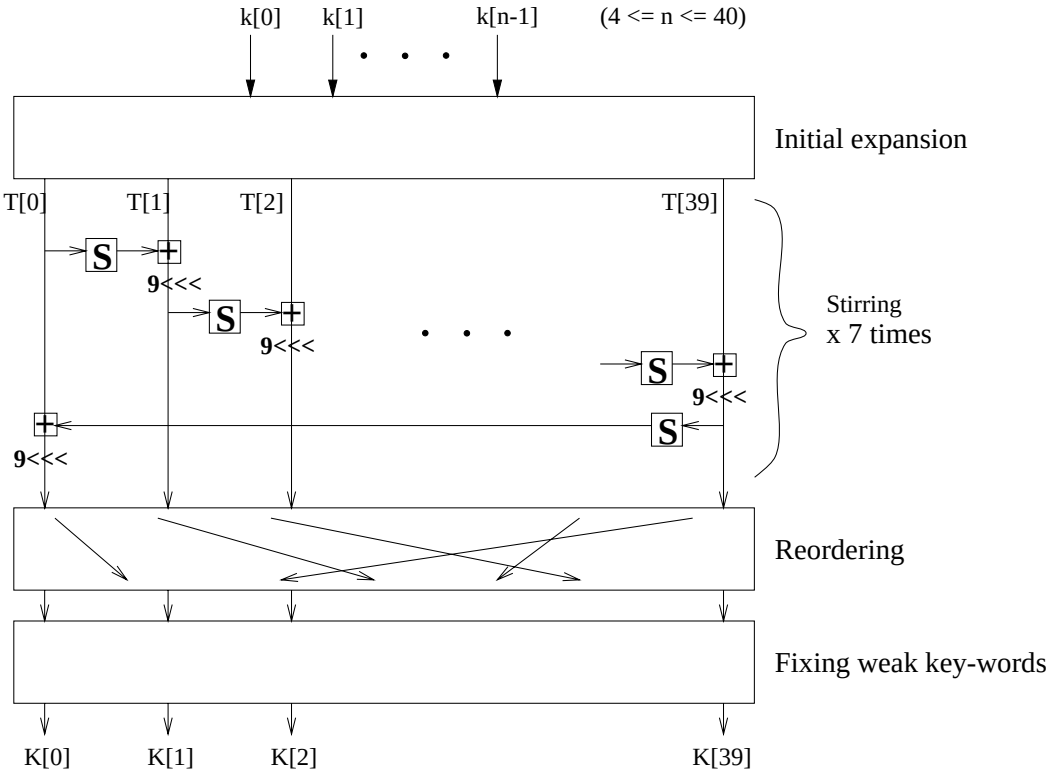

Figure 6: The key expansion procedure

view as indexed from $T[-7]$ through $T[39]$. The first 7 entries in $T[\,]$ are initialized with the first seven entries in the S-box S, by setting

$$\text{for } i = -7 \ldots -1, \quad T[i] = S[i+7]$$

Entries 0 through 38 in $T[\,]$ are filled using the formula

$$\text{for } i = 0 \ldots 38, \quad T[i] = ((T[i-7] \oplus T[i-2]) \lll 3) \oplus k[i \mod n] \oplus i$$

and the last entry is set to hold the length of the original key, $T[39] = n$. The reason for this last setting is to eliminate the possibility that two keys of different lengths generate the same 40-word expanded key (for example, this way we guarantee that the all-zero 4-word key results in a different expanded key than the all-zero 6-word key).

2. Next we stir the temporary array using seven rounds of type-1 Feistel network. Specifically, we repeat the operation

$$T[i] = (T[i] + S[\text{low 9 bits of } T[i-1]]) \lll 9, \quad i = 1, 2, \ldots 39$$
$$T[0] = (T[0] + S[\text{low 9 bits of } T[39]]) \lll 9$$

for seven rounds.

3. Then we reorder the words into the expanded key array, $K[\,]$, so that words which were adjacent in the stirring phase will be far apart in the resulting array. This is done by setting

$$K[7i \mod 40] = T[i], \; i = 0, 1, \ldots, 39$$

4. Finally, we go over the sixteen words which are used in the cipher for multiplication (these are words $K[5], K[7], \ldots, K[35]$), and check that none of them is "weak": We say that a word w is weak if $(w \vee 3)$ (i.e., the word w with the two lowest bits set to 1) contains either ten consecutive 0's or ten consecutive 1's. See Subsection 4.2 for the reasoning behind this choice. We note that the probability that a randomly chosen word is weak is about $1/41$. We process each of the words $K[5], K[7], \ldots, K[35]$ as follows:

   (a) We record the two lowest bits of $K[i]$, by setting $j = K[i] \wedge 3$, and then consider the word with these two bits set to 1, $w = K[i] \vee 3$.

   (b) We construct a mask M of the bits in w which belong to a sequence of ten (or more) consecutive 0's or 1's. Namely, we have $M_\ell = 1$ if and only if w_ℓ belongs to a sequence of ten consecutive 0's or 1's. Then we reset to 0 the 1's in M which correspond to the two end-points of every run of 0's or 1's in w. The two low order bits of M are also reset to 0.

   For example, assume that we have $w = 0^3 1^{13} 0^{12} 1001$ (where by $0^i, 1^i$ we denote i consecutive 0's or 1's, respectively). In this case we first set $M = 0^3 1^{25} 0^4$, and then we reset the 1's in bit positions 4, 15,16 and 28 to get $M = 0^4 1^{11} 001^{10} 0^5$.

   (c) Next we use a fixed four-word table B to "fix w", where the four entries in B are chosen so that they (and their cyclic shifts) do not contain any seven consecutive 0's or ten consecutive 1's. Specifically, we use the table `B[ ] = {0xa4a8d57b, 0x5b5d193b, 0xc8a8309b, 0x73f9a978}`, (these are entries 265 through 268 in the S-box). The reason we chose these entries is that there are only 14 8-bit patterns which appear twice in these entries (and their cyclic shifts), and no pattern appears more than twice.

   We use the two recorded bits j (from Step (a)) to select an entry from B, and use the lowest five bits of $K[i+3]$ to rotate this entry, $p = B[j] \ll$ (lowest 5 bits of $K[i+3]$).

   (d) Finally, we xor the pattern p into w under the control of the mask M, and store the result in $K[i]$

   $$K[i] = w \oplus (p \wedge M)$$

   Since the lowest two bits of M are 0's, then the lowest two bits of $K[i]$ will be 1's (since those in w are). Also, the choice of B guarantees that $K[i]$ will not be weak.

We note that this procedure not only guarantees that the words $K[5], K[7] \ldots K[35]$ are not weak, but also keeps these words "random", in the sense that no single word has probability much larger than in the uniform distribution. In particular, an exhaustive search confirmed that no 20-bit pattern occurs in these words with probability of more than 1.23×2^{-20}. Similarly, no 10-bit pattern appears with probability larger than 1.06×2^{-10}. We use these facts in the analysis of the cipher. A pseudo-code for the key expansion procedure is given below.

Key-Expansion(input: $k[\,], n$; output: $K[\,]$)

1. // n is the number of words in the key buffer $k[\,]$, $(4 \leq n \leq 39)$
2. // $K[\,]$ is the expanded key array, consisting of 40 words

3. // $T[\,]$ is a temporary array, consisting of 47 words, $T[-7], T[-6], \ldots, T[39]$
4. // $B[\,]$ is a fixed table of four words

5. // Initialize $B[\,]$
6. $B[\,] = \{$`0xa4a8d57b`, `0x5b5d193b`, `0xc8a8309b`, `0x73f9a978`$\}$

7. // Initialize $T[\,]$ with seven constants, then key data
8. $T[-7 \ldots -1] = S[0 \ldots 6]$
9. for $i = 0$ to 38 do
10. $\quad T[i] = ((T[i-7] \oplus T[i-2]) \lll 3) \oplus k[i \bmod n] \oplus i$
11. $T[39] = n$

12. // Do seven rounds of stirring
13. repeat 7 times
14. $\quad$ for $i = 1$ to 39 do
15. $\quad\quad T[i] = (T[i] + S[\text{low 9 bits of } T[i-1]]) \lll 9$
16. $\quad T[0] = (T[0] + S[\text{low 9 bits of } T[39]]) \lll 9$ $\quad$ // wrap around end
17. end-repeat

18. for $i = 0$ to 39 do $\quad$ // reorder the key words into $K[\,]$
19. $\quad K[7i \mod 40] = T[i]$

20. // Fix "weak" key-words (w is weak if $(w \vee 3)$ contains ten consecutive 0's or 1's)
21. for $i = 5, 7, \ldots 35$ do
22. $\quad j =$least two bits of $K[i]$
23. $\quad w = K[i]$ with both of the least two bits set to 1

24. $\quad$ // Generate a bit-mask M (if $K[i]$ is not weak then $M = 0$)
25. $\quad M_\ell = 1$ iff w_ℓ belongs to a sequence of ten consecutive 0's or 1's in w
26. $\quad\quad$ and also $\ell \geq 2$ and $w_{\ell-1} = w_\ell = w_{\ell+1}$

27. $\quad$ // Select a pattern from the fixed table and rotate it
28. $\quad r =$least five bits of $K[i+3]$ $\quad$ // rotation amount
29. $\quad p = B[j] \lll r$

30. $\quad$ // Modify $K[i]$ with p under the control of the mask M
31. $\quad K[i] = w \oplus (p \wedge M)$
32. end-for

3 Computational Efficiency (2.B.2)

Due to the structure of the key expansion procedure, the performance of MARS is essentially independent of the key-length used. Hence, below we only provide a single figure for the performance of MARS on any given platform, and these figures do not change with the key size.

3.1 Software implementation

Since MARS was designed to take full advantage of the powerful operation available on today's computers, it can achieve a very high speed in software. We estimate that a fully optimized assembly implementation of the cipher (on most of the platforms available today) requires about 450-650 machine instructions for encryption of a single 128-bit block. Most of these instructions can be paired to take advantage of super-scalar architectures, leading to an estimate of about 250-400 cycles for encryption of a single block. On a machine with clock-rate of 200MHz, this estimate yields encryption rates from 65 to 100 Mbit/sec.

3.1.1 C implementation

We currently have a C implementation of MARS running at rates of 65-85 Mbit/sec on machines with clock rate of 200MHz. We measured the performance of this implementation of MARS on the following platforms:

	Pentium-Pro Borland C++ 5.0	Pentium-Pro DJGPP (+ pgcc101)	PowerPC 604e C Set ++ 3.1.1
encryption	920 $\frac{\text{cycles}}{\text{block}}$ (28 mbps)	390 $\frac{\text{cycles}}{\text{block}}$ (65 mbps)	300 $\frac{\text{cycles}}{\text{block}}$ (85 mbps)
decryption	920 $\frac{\text{cycles}}{\text{block}}$ (28 mbps)	390 $\frac{\text{cycles}}{\text{block}}$ (65 mbps)	300 $\frac{\text{cycles}}{\text{block}}$ (85 mbps)
key-setup	9200 $\frac{\text{cycles}}{\text{key}}$	3950 $\frac{\text{cycles}}{\text{key}}$	2050 $\frac{\text{cycles}}{\text{key}}$
algorithm-setup	0 cycles	0 cycles	0 cycles
key-change	9200 $\frac{\text{cycles}}{\text{key}}$	3950 $\frac{\text{cycles}}{\text{key}}$	2050 $\frac{\text{cycles}}{\text{key}}$

Table 1: Timing measurements for the C implementation of MARS (mbps stands for Mbit/sec).

- We measured the performance on the reference platform, which is an IBM-compatible PC, with a 200MHz Pentium Pro processor and 64MB RAM, running Windows95. On this machine we used two different compilers to compile the C code. One is the NIST reference compiler, Borland C++ version 5.0. Unfortunately, the Borland compiler does a very poor job in taking advantage of the speed potential of MARS. In particular, *the Borland compiler penalizes algorithms which use data rotations much more than other algorithms*, since it implements every rotation operation as three machine instructions (two shifts and an OR) instead of using the rotate operation which is available in the Intel architecture.

	Borland C++ 5.0	DJGPP (+ pgcc101)
MARS	28 Mbit/sec	65 Mbit/sec
DES (RSAREF)	10.6 Mbit/sec	16.7 Mbit/sec
Triple-DES	4.4 Mbit/sec	7.3 Mbit/sec

Table 2: Encryption speed of several ciphers using two different compilers.

We therefore also compiled the C implementation using the Pentium-optimizing version of the Gnu-C compiler (pgcc) version 1.0. This compiler is freely available over the Internet from `http://www.gcc.ml.org`, and can be used with most of the Unix variants running on Intel. In addition it was ported to DOS (under the DJGPP compiler) so it can also be used under Windows. It is this port (DJGPP version 2.01) that we used for our timing measurements. The speed of MARS using the two compilers is described in Table 1.

We remark that on the Intel platform there is some tradeoff between the speeds of key generation and encryption: We can store the S-box in the key schedule itself, thereby saving one pointer during the encryption process (since the same pointer can be used to point to both the key and the S-box). This results in a speedup of about 5% in the encryption/decryption, making it run at about 67 Mbit/sec, but at the same time it implies a 50% slowdown in the key setup.

- We also measured the speed of our C implementation on an RS/6000 43P workstation model 140, with a 200 MHz PowerPC 604e processor and 64MB RAM, running AIX. On this platform we used the xlC compiler (included in C Set ++ for AIX, version 3.1.1). The running time of MARS on this platform is also described in Table 1.

To demonstrate the fact that the Borland compiler penalizes algorithms which use data rotations much more than it penalizes other algorithms, we compare in Table 2 the encryption speed of MARS to that of DES and triple-DES under the two compilers. It can be seen in that table that the speed of DES degrades by only about 35% by switching from DJGPP to Borland, while the speed of MARS is cut by more than a factor of two.

3.1.2 Java implementation

We tested the java implementation on the same platforms as the C implementation. On the Intel platform we used the javac compiler and java interpreter from JDK1.1.6 and the symjit just-in-time compiler that comes with JDK1.1.6 for Windows. On the PowerPC we used the javac compiler and java interpreter from JDK1.1.4 and the jitc just-in-time compiler that comes with JDK1.1.4 for AIX. The running time of our implementation is given in Table 3.[4] In can be seen that the optimized Java code runs only about 4 times slower than the C code (and is roughly equivalent in speed to the C implementation of DES).

[4]The results in Table 3 represent the speed of the low-level word-oriented routines for encryption, decryption and key-setup. These results do not include the time for byte-to-word conversion, endianess conversion or memory allocation.

	Pentium-Pro	PowerPC 604e
encryption	1760 $\frac{\text{cycles}}{\text{block}}$ (14.5 Mbit/sec)	1280 $\frac{\text{cycles}}{\text{block}}$ (19.9 Mbit/sec)
decryption	1480 $\frac{\text{cycles}}{\text{block}}$ (17.3 Mbit/sec)	1240 $\frac{\text{cycles}}{\text{block}}$ (20.6 Mbit/sec)
key-setup	7100 $\frac{\text{cycles}}{\text{key}}$	8480 $\frac{\text{cycles}}{\text{key}}$
algorithm-setup	0 cycles	0 cycles
key-change	7100 $\frac{\text{cycles}}{\text{key}}$	8480 $\frac{\text{cycles}}{\text{key}}$

Table 3: Timing measurements for the Java implementation of MARS.

3.1.3 Memory requirements

Implementations of MARS need 2Kbyte of memory to store the S-box, 160 bytes to store the expanded key and a few more words to carry the operations of the cipher. This small amount of memory fits easily in the first-level cache of any modern processor.

3.2 Implementation on 8-bit processors

We estimate that a software implementation of MARS on a simple 8-bit processor would take about 5000 cycles for encryption/decryption of a single block, and about 15000 cycles for key-setup.

The processor model that we use for these estimates has a few general purpose 8-bit registers (we assume four registers in our estimates). We assume that most of the logic and arithmetic operations (add, xor, shift, etc.) take a single cycle, either between two registers or between a register and a memory location. We also estimate that the processor has a multiplication operation which multiplies two 8-bit values and returns the 16-bit result in two registers, and that this operation takes four cycles.

With these assumptions, we get the following estimates for the basic operations of MARS:

- A multiplication of two 32-bit words can be implemented using six $8 \times 8 \rightarrow 16$ multiplications, four $8 \times 8 \rightarrow 8$ multiplications and 33 other operations. If each multiplication takes four cycles and the other operations take a single cycle, then we get 73 cycles for a single $32 \times 32 \rightarrow 32$ multiplication.
- A data-dependent-rotation operation on a 32-bit word can be implemented using 8 shifts, 4 or's and 22 other operations. Hence, we can perform it in 34 cycles.
- A fixed rotation by 8, 16 or 24 bit positions does not take any time, since it only involves renaming the variables. A rotation by other fixed amounts takes 12 cycles.
- An S-box lookup with 8-bit index takes 8 cycles, and an S-box lookup with 9-bit index takes 12 cycles.
- The other operations on 32-bit words (move, add, subtract, and, or, not, xor) each take 8 cycles to implement on an 8-bit processor.

Operation	# of operations	cycles/operation	# of cycles
Multiplication	16	73	1168
Data-dependent rotation	32	34	1088
Fixed rotation	48	12	576
8-bit $S[\cdot]$	64	8	512
9-bit $S[\cdot]$	16	12	192
others	184	8	1472
Total encryption/decryption			5008

Operation	# of operations	cycles/operation	# of cycles
Data-dependent rotation	16	34	544
Fixed rotation/shift	480	12	5760
9-bit $S[\cdot]$	296	12	3552
others	740	8	5920
Total key setup			15776

Table 4: Estimated speed of MARS on an 8-bit processor

Our estimates for the speed MARS on an 8-bit processor are summarized in Table 4. We note that on a smartcard with clock rate of 20MHz, these estimates imply an encryption rate of about 500 Kbit/sec. However, it is not clear what is the meaning of this last estimate, since smartcards that are used for encryption are typically equipped with a dedicated crypto unit, and so can execute MARS much faster. (We show below that a hardware implementation of MARS can easily fit on a smartcard). Moreover, even without a dedicated crypto chip, modern smartcard controllers have much more capabilities than our simple 8-bit processor model. For instance, the Intel 80251 controller can operate on 16-bit words (and even 32-bits words). It is likely that our estimated speed can be improved by a factor of at leat four on such a processor.

3.3 Hardware implementation

The MARS algorithm lends itself very well to a hardware based implementation. The MARS algorithm, even in a non-optimal implementation, provides significant performance gains over software implementations. We estimate the performance advantage at $10\times$ versus the software implementation.

Our analysis shows that the forward mixing phase (including the key addition and the unkeyed mixing) can be completed within 9 cycles. The same analysis applies to the backwards mixing phase, which can also be completed within 9 cycles. For the keyed transformation phase, we've included only one multiplier in our initial estimates. We've designed one E-function and are using it for each successive iteration. Therefore, our estimate is that it will take 2 cycles to complete each E-function, and 32 cycles to complete the sixteen rounds. In total, we estimate that an encryption of one block takes 50 cycles.

One of our goals in performing a hardware assessment was to get a reasonable combination of size

an speed. There is considerable reuse of S-boxes,adders, exclusive-or functions, and multipliers in order to minimize the cell counts. With only one multiplier in our hardware based design, our performance estimate for MARS is 80MByte/sec, or 640 Mbit/sec. The cell count for this implementation is approximately 70,000 cells. This count includes circuitry for encryption, decryption and key generation (but does not include the registers for the sub-keys). The majority of the cell usage is devoted to the S-boxes, adders, and the multiplier.[5] As a basis for comparison, a typical DES implementation is approximately 28,000 cells.

A count of approximately 70,000 cells is not extraordinary. This cell count will easily fit on all chips, including smart cards. This small size provides the algorithm with great flexibility and the ability to be utilized in many varied applications.

Modes of operation that allow pipelining (such as ECB mode, counter mode, or decryption in CBC mode) can be implemented much faster. In particular, a hardware implementation consisting of four copies of the mixing rounds and the E-function can produce a throughput of one block every 8 cycles, resulting in an encryption/decryption rate of 4Gbit/sec. It is even possible to use four copies of the mixing rounds and eight copies of the E-function to get a throughput of one block every 4 cycles. The cell count of this last implementation is about 393,000 (which is still reasonable), and it achieves overall performance of 8 Gbit/sec. It follows that for applications that only need to decrypt (such as DVD players), we can build a hardware chip of MARS with a decryption rate of 8 Gbit/sec.

3.4 Other implementations

The MARS algorithm is suitable for implementation in a variety of environments. We previously demonstrated that the algorithm can be implemented efficiently in both software and hardware. This flexibility is extremely important since it provides us with an implementation choice for differing environments which may be constrained either by physical silicon space or memory application space. Environments such as smart cards possess both physical and application space constraints. However, the MARS algorithm can be implemented in silicon which will easily fit within the smart card specifications and still leave plenty of room for the processor and other logic functions. If silicon space needs to be conserved, then the algorithm can be executed on the native 8-bit processor, or a combination of a minimal hardware implementation plus the native processor can be used.

MARS' characteristics (flexibility, high-speed, security, efficient implementations, etc.) and implementation options are attractive and applicable to Asynchronous Transfer Mode products, High Definition Television, B-ISDN, voice applications, satellite applications, and many other networked applications. It will provide robust, high speed encryption and decryption capabilities to every solution. MARS is highly suitable for all of these varied applications.

[5] Still, less than 20% of this count is due to the multiplier.

4 Analysis and Expected Strength (2.B.4–2.B.5)

We use the following terminology when talking about the resistance of MARS to certain attacks:

data complexity. The data complexity of an attack is the number of (plaintext,ciphertext) pairs that an attacker must see (or choose, in the case of chosen plaintext/ciphertext attacks) before it can distinguish between the cipher and a random permutation.

work load. The work load of an attack is the number of operations it takes. This is always at least as large as the data complexity, but can sometimes be larger. For example, exhaustive key search has very low data complexity (typically two or three pairs are enough), but its work load is exponential in the key length.

key probability. Some of the attacks described in the sequel can only proceed when some of the key words have special properties. In this case, the key probability of an attack is the probability that a random key has these special properties. In computing this probability, we assume that each entry in the expanded key array is chosen independently at random (subject to the constraints imposed by the key-setup process).

security level. The security level of a cipher relative to a certain attack (or class of attacks) is the ratio between the work load and success probability of the attack. The success probability is the probabilistic advantage that the attacker gains in distinguishing between the cipher and a random permutation. The probability is taken over both the choice of the key and the randomness used in the attack itself. (For example, if an attack has work load of 2^{20} and key probability of 2^{-30}, then the security level of the cipher relative to this attack is 2^{50}.)

The (conjectured) security level of a cipher is its security level relative to the (conjectured) best possible attack. We remark that a cipher with key length of n bits cannot have security level of more than 2^n.

Expected strength of MARS. We expect the security level of MARS with an n-bit key to be 2^n for key lengths up to 256 bits. We do not expect the security level to grow as rapidly beyond 2^{256}. In particular, there may be attacks with work-load of about 2^{300} even when all the key words are chosen independently. Hence the main reason for using keys longer than 256 bits is convenience, not security.

We estimate that any linear or differential attacks against MARS must have data complexity of more than 2^{128}, which means that for block-length of 128 bits these attacks are impossible. Below we justify this estimate by providing crude (though conservative) bounds on the complexity of such attacks. For these bounds we consider only the "cryptographic core" of MARS (which is equivalent to analyzing 16R-attacks in the sense of [3], since it entails ignoring the 16 rounds of mixing in the cipher).

For linear attacks, we argue in Subsection 4.1 that no "constructible" linear approximation of the keyed transformation has a bias of more than 2^{-69}, which implies data complexity of more than 2^{128}. By "constructible" approximation we mean an approximation which is obtained by combining approximations for the internal operations of the cipher, computing the bias using the

Piling-up lemma [9]. Although in principle a cipher can also have linear properties which result from some "global cancellations", we do not know of any such properties for MARS[6].

For differential attacks [3] we provide two arguments: We first present a heuristic argument explaining why it is unlikely that one would be able to construct a characteristic of the keyed transformation with probability more than 2^{-240}, taken over both the key and the data. We then also devise a more conservative (and very crude) bound of 2^{-156} on the probability of any characteristic of the keyed transformation, where the probability is again taken over both the key and the data.

4.1 Linear analysis

In linear analysis [9] one tries to find a subset of the bit positions in the plaintext, ciphertext and expanded keys, so that for a uniformly chosen plaintext and expanded key, the probability that the sum of the bits in these positions is equal to zero modulo 2, will be bounded away from $1/2$. Such a subset is called a *linear approximation* of the cipher, and the difference between the obtained probability and $1/2$ is called the *bias* of the approximation. In general, the goal of linear analysis is to find approximations with large bias, since an approximation with bias ε typically corresponds to an attack with work-load and data-complexity of about $(1/\varepsilon)^2$.

Notations. Below, a linear approximation of an operation involving the words $w_1 \ldots w_n$ is specified via a set of *masks* $X_1 \ldots X_n$, such that a certain bit-position in w_i belongs to the approximation if and only if the corresponding bit of X_i is '1' (in this writeup all the words are of length 32-bits). We describe this approximation by the formula

$$A(w_1, \ldots, w_n) \stackrel{\text{def}}{=} (w_1 \circ X_1) \oplus \ldots \oplus (w_n \circ X_n) \tag{1}$$

where $\oplus$ denotes exclusive-or (i.e. addition modulo 2) and $\circ$ denotes the inner product operation modulo 2. The bias of this approximation is then

$$\left| \Pr[A(w_1, \ldots, w_n) = 0] - \frac{1}{2} \right| \tag{2}$$

where the probability is taken over the uniform choice of all the words which are considered *the inputs* to this operation (hence to define the bias we must specify which words are the inputs of the operation and which are the outputs).

Local and Global approximations. The standard way to devise linear approximations for a complex operation (such as a cipher) is to combine approximations for some of the internal "basic operations". Combining several approximations $A_1 \ldots A_l$ is done by simply "adding them modulo 2". Namely, the resulting approximation is $A = A_1 \oplus \ldots \oplus A_l$, and it consists of all the bit positions that appear *an odd number of times* in all the approximations $A_1 \ldots A_l$. Below, when we combine several linear approximations to obtain a new one, we informally say that the *local approximations*

[6] For example, the data-dependent rotation operations in two consecutive rounds never cancel each other, and so we have to approximate each of them separately.

$A_1 \ldots A_l$ are combined to obtain a *global approximation A*. Clearly, to be of any use, a global approximation of a cipher must only include bit-positions in the plaintext, ciphertext and key. Namely, the occurrences of bit-positions in internal variables of the cipher (which appear in the local approximations) must all cancel out in the global approximation.

We use the Piling-up lemma [9] to compute the bias of the global approximation from the bias of the local approximations. If the bias of the approximations $A_1 \ldots A_l$ is denoted by $b_1 \ldots b_l$, then the bias b of the combined approximation A is computed as

$$b = \frac{1}{2} \cdot \prod_{i=1}^{l} (2b_i) \tag{3}$$

We note that this formula assumes that the inputs to the different operations are chosen independently (which is usually not the case), and so the formula represents only a heuristic evaluation of the bias of A.

Treatment of key bits in linear approximations. In principle, one can treat the key bits in a linear approximation differently than the data bits. For example, we can use an approximation involving only the data bits, and take the *expected value*, over the random choice of the key, of the absolute value of the bias of this approximation. We are leaving this for future research.

Organization. Below we only discuss linear approximations of the keyed transformation of MARS. The rest of this section is organized as follows: We start by discussing the linear approximations of the operations used in the E-function, and then analyze linear approximations of the E-function itself. Then, we use this analysis to provide a conservative bound on the bias of every linear approximation of the keyed transformation.

4.1.1 Linear approximation of the basic operations

The basic operations used in the E-function are addition modulo 2^{32} $(+)$; 9-to-32-bit table look-up $(S[\cdot])$; exclusive-or $(\oplus)$; multiplication modulo 2^{32} $(\times)$; and also data rotation by fixed and varying amounts $(\ll)$. Below we briefly discuss some properties of the linear approximations of these operations.

Exclusive-or. The exclusive-or operation, $w_3 = w_1 \oplus w_2$ (with inputs w_1, w_2 and output w_3), is approximated by $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$. This approximation has bias $1/2$ if $X_a = X_b = X_c$, and it has zero bias otherwise.

Addition. The addition operation, $w_3 = w_1 + w_2$ (inputs w_1, w_2, output w_3), can be viewed as $w_3 = w_2 \oplus w_1 \oplus c$ where c is the carry-bit pattern. The following probabilities are useful in computing the bias of any particular approximation for this operation (below c_i denotes the carry into

bit position i):

$$\Pr[c_i = 1] = \frac{1}{2} - \frac{1}{2^{i+1}}, \quad \text{for } i = 0 \ldots 31$$
$$\Pr[c_i = 1 | c_{i-j} = 1] = \frac{1}{2} + \frac{1}{2^{j+1}}, \quad \text{for } i = 0 \ldots 31,\ j = 1 \ldots i$$
Given c_{i-j}, c_i is independent of $c_{i-j-1} \ldots c_0$

The following facts are also useful:

Assertion 1

- *The LSB-approximation ($X_1 = X_2 = X_3 = 0^{31}1$) is the only approximation for $+$ with bias $1/2$.*
- *The parity-approximation ($X_1 = X_2 = X_3 = 1^{32}$) has bias 2^{-17}.*
- *A necessary (but not sufficient) condition for the approximation $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$ to have nonzero bias, is that the most significant bit in X_1, X_2, X_3 is in the same position.*
- *If the Hamming-weight of either X_1, X_2 or X_3 is h, then the bias of the approximation $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$ is at most $2^{-1-\lfloor h/2 \rfloor}$.*

Multiplication. The multiplication operation, $w_3 = w_1 \times w_2$ (inputs w_1, w_2, output w_3) is approximated by $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$. In MARS, we force the lowest two bits of w_2 (which is the key word) to be '1', and so they need not be present in an any approximation. With this restriction, the multiplication operation has exactly three approximations with bias $1/2$, all involving only the two lowest bits in the inputs and output (and hence using $X_2 = 0$). These approximations are

(1) The LSB-approximation, $X_1 = X_3 = 0^{31}1$,
(2) The second-bit approximation, $X_1 = 0^{30}11$, $X_3 = 0^{30}10$, and
(3) The sum of the two lowest bits, $X_1 = 0^{30}10, X_3 = 0^{30}11$.

Although we do not have a rigorous analysis of the linear properties of $\times$, it seems that linear approximations for the high-order bits in the inputs and output of this operation have only very small bias.

S-box lookup. This unary operation, $w_2 = S[\text{lowest 9 bits of } w_1]$ (input w_1, output w_2) is approximated by $(X_1 \circ w_1) \oplus (X_2 \circ w_2)$, where X_1 is zero everywhere except in the lowest 9 bits. We picked the S-box so that approximations involving very few bits will have only a small bias. Specifically, S was chosen so that any approximation consisting only of one output bit (i.e. $X_1 = 0$ and X_2 has a single '1') has bias of at most $1/30$, each approximation consisting of exactly one input bit and one output bit has bias of less than $1/22$, and the parity approximation has bias 2^{-7}. We conjecture that there are no approximations of the S-box with bias of more than 2^{-3}.

Data-dependent rotation. The data-dependent rotation operation $w_3 = w_1 \lll w_2$ (inputs w_1, w_2, output w_3), is approximated via $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$, where X_2 is zero everywhere except in the lowest 5 bits (as only the lowest 5 bits of w_2 affect this operation). This operation can be approximated as either a binary or unary operation, depending on whether the rotation amount is included or excluded from the approximation. Approximations of data-dependent rotation were investigated by Moriai, Aoki and Ohta in [10], where the following is proven:

Theorem 2 ([10]) *For two masks X_1, X_3, denote by $\rho(X_1, X_3)$ the number of different rotation amounts $n < 32$ such that $X_3 = X_1 \lll n$.*[7] *Then, the approximation $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$ has bias of $\rho(X_1, X_2)/64$ provided that $X_2 < 32/\rho(X_1, X_3)$, and it has zero bias otherwise.*[8]

A useful corollary of this assertion provides a connection between the Hamming weight of X_1, X_3 and the bias of the approximation.

Corollary 3 *Let $(X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3)$ be an approximation of the operation $w_3 = w_1 \lll w_2$. If the Hamming weight of X_1 or X_3 is in the range $[2^i, 2^{i+1} - 1]$ (for some $i \leq 5$), then $\rho(X_1, X_3)$ is at most 2^i and thus the bias of the approximation is at most 2^{i-6}.*

Combining rotations with additions. In two of the three output lines of the E-function the output of the data-dependent rotation is used as input to an addition operation. It is therefore useful to analyze the linear properties of this combined operation.

Assertion 4 *Consider the ternary operation $w_4 = (w_1 \lll w_2) + w_3$ (inputs w_1, w_2, w_3, output w_4), and let $A \stackrel{\text{def}}{=} (X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_3 \circ w_3) \oplus (X_4 \circ w_4)$ be a linear approximation of this operation. Then A has bias of at most 2^{-6}.*

Reasoning: An approximation as above is obtained by adding the local approximations for the $\lll$ and $+$ operations. Namely, we have $A = A_{\lll} \oplus A_+$ where $A_{\lll}, A_+$ are approximations for $\lll, +$, respectively

$$A_{\lll} \stackrel{\text{def}}{=} (X_1 \circ w_1) \oplus (X_2 \circ w_2) \oplus (X_{\lll} \circ w_{\lll})$$
$$A_+ \stackrel{\text{def}}{=} (X_{\lll} \circ w_{\lll}) \oplus (X_3 \circ w_3) \oplus (X_4 \circ w_4)$$

where $w_{\lll}$ is the internal variable describing the output of the data-dependent-rotation (which is also an input to the addition). Notice that the same mask $X_{\lll}$ appears in both $A_{\lll}$ and A_+, since it must cancel in the global approximation A. Denote the Hamming weight of the mask $X_{\lll}$ by h. Then,

(a) By Corollary 3, if $2^i \leq h < 2^{i+1}$ (for some $i \leq 5$), then the approximation $A_{\lll}$ has bias of at most $2^i/64$.

(b) By Assertion 1, the approximation A_+ has bias of at most $\frac{1}{2} \cdot 2^{-h/2}$.

Combining these two facts, and using the Pilinp-up lemma, we conclude that the combined bias of the approximations A is at most 2^{-6}. □

[7] It follows that for 32-bit words, $\rho(X_1, X_3)$ must be either zero or a power of two.

[8] The "mysterious" expression $X_2 < 32/\rho(X_1, X_3)$, in which X_2 is viewed as the binary representation of an integer, simply means that the only bit-positions of w_2 in the approximations are the ones which are relevant for the operation. For the special case that $\rho(X_1, X_3) = 1$, this condition means that X_2 is zero everywhere except in the lowest 5 bits.

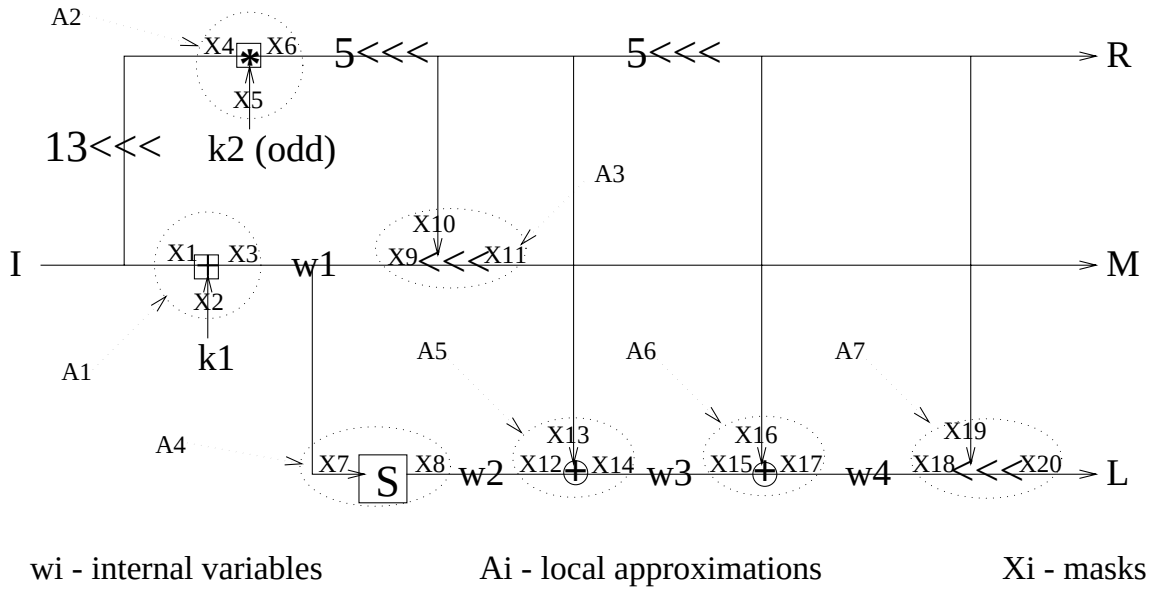

Figure 7: Labeling of the lines in the E-function.

4.1.2 Linear approximations of the E-function

We now consider approximating the E-function, which is one of the main sources of nonlinearity in MARS. Figure 7 contains labeling of the internal lines of the E-function which we use below. The w_i's represent variables which are internal to the E-function, and are thus created and used only within the E-function. The E-function has seven internal operations (other than the fixed rotations): one $+$, two $\oplus$'s, one $\times$, one S-box lookup and two data-dependent rotations. Table 5 shows the labeling of the inputs and outputs in each of these operations, and the linear approximation to the operation.

Operation	Input(s)	Output	Approximation
$+$	in, k_1	w_1	$A_1 \stackrel{\text{def}}{=} (X_1 \circ I) \oplus (X_2 \circ k_1) \oplus (X_3 \circ w_1)$
$\times$	$in \lll 13, k2$	$R \ggg 10$	$A_2 \stackrel{\text{def}}{=} (X_4 \circ I) \oplus (X_5 \circ k_2) \oplus (X_6 \circ R)$
$\lll$	$w_1, R \ggg 5$	M	$A_3 \stackrel{\text{def}}{=} (X_{12} \circ w_1) \oplus (X_{13} \circ R) \oplus (X_{14} \circ M)$
$S[\cdot]$	w_1	w_2	$A_4 \stackrel{\text{def}}{=} (X_7 \circ w_1) \oplus (X_8 \circ w_2)$
$\oplus$	$w_2, R \ggg 5$	w_3	$A_5 \stackrel{\text{def}}{=} (X_9 \circ w_2) \oplus (X_{10} \circ R) \oplus (X_{11} \circ w_3)$
$\oplus$	w_3, R	w_4	$A_6 \stackrel{\text{def}}{=} (X_{15} \circ w_3) \oplus (X_{16} \circ R) \oplus (X_{17} \circ w_4)$
$\lll$	w_4, R	L	$A_7 \stackrel{\text{def}}{=} (X_{18} \circ w_4) \oplus (X_{19} \circ R) \oplus (X_{20} \circ L)$

Table 5: Local approximations of the operations in the E-function.

	Local approximations	Operations
1	$\{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$	$\{+, \lll, \times, S[\cdot], \oplus, \oplus, \lll\}$
2	$\{A_1, A_2, A_4, A_5, A_6, A_7\}$	$\{+, \times, S[\cdot], \oplus, \oplus, \lll\}$
3	$\{A_1, A_2, A_3\}$	$\{+, \times, \lll\}$
4	$\{A_1, A_3, A_4, A_5, A_6, A_7\}$	$\{+, \lll, S[\cdot], \oplus, \oplus, \lll\}$
5	$\{A_1, A_4, A_5, A_6, A_7\}$	$\{+, S[\cdot], \oplus, \oplus, \lll\}$
6	$\{A_1, A_3\}$	$\{+, \lll\}$
7	$\{A_2\}$	$\{\times\}$
8	$\{A_2, A_4, A_5, A_6, A_7\}$	$\{\times, S[\cdot], \oplus, \oplus, \lll\}$
9	$\{A_2, A_3, A_4, A_5, A_6, A_7\}$	$\{\times, \lll, S[\cdot], \oplus, \oplus, \lll\}$
10	$\{A_2, A_3\}$	$\{\times, \lll\}$
11	$\{A_3, A_4, A_5, A_6, A_7\}$	$\{\lll, S[\cdot], \oplus, \oplus, \lll\}$
12	$\{A_4, A_5, A_6, A_7\}$	$\{S[\cdot], \oplus, \oplus, \lll\}$

Table 6: Global approximations of the E-function.

A global approximation of the E-function is of the form

$$A \stackrel{\text{def}}{=} (X_I \circ I) \oplus (X_L \circ L) \oplus (X_M \circ M) \oplus (X_R \circ R) \oplus (X_{k_1} \circ k_1) \oplus (X_{k_2} \circ k_2) \tag{4}$$

where at least one of the output masks X_L, X_M, X_R is nonzero.

A global approximation to the E-function is devised by selecting a subset of the local approximations $A_1 \ldots A_7$ and assigning values to the masks in these approximations. We note that the masks used in different approximations must be related in order to get a useful global approximation of the E-function. In particular, an approximation is only useful if it does not include the intermediate values w_i. However since local approximations to the operations of the E-function necessarily involve w_i values, the occurrences of these values must cancel each other modulo 2. Also, an approximation is only useful if it has non-zero bias, hence, for example, all the masks which are adjacent to an $\oplus$ operation must be the same.

Below we say that local approximation A_i is included in a global approximation if at least one of the masks of this approximation is non-zero. Table 6 lists all the useful global approximations of the E-function, according to which local approximations are included in them. We remark that a certain subset of the local approximations can give rise to many different global approximations, depending on the setting of the relevant masks.

Example 1. Consider an approximation of the E-function which only uses local approximations A_1, A_2 and A_3 (Line 3 in Table 6). A conceivable way to devise such approximation is to assign non-zero values only to the masks $X_1, X_2, X_3, X_4, X_6, X_9, X_{10}$ and X_{11} (and possibly also to X_5), in such a way that $X_4 = X_1 \lll 13$, $X_3 = X_9$ and $X_{10} = X_6 \lll 5$. Then, the intermediate values on input line I and output line R (as well as the intermediate value w_1) cancels modulo 2, and the resulting approximation is of the form

$$(X_2 \circ k_1) \oplus (X_5 \circ k_2) \oplus (X_{11} \circ M) \tag{5}$$

which is a valid global approximation of the E-function. Clearly, such an approximation is only useful if there is a way to assign values to these masks so that the resulting local approximations have non-zero bias. In particular, X_{10} must be zero in all but the lowest five bits, which implies that X_6 is zero in all but the highest five bits. Hence, one must use a linear approximation for the highest bits of the multiplication output, and such approximations seem to have only a very small bias. We conjecture that no approximation of the form (5) has bias of more than 2^{-15}. Also, since the value of M is then added into the data line, to use such an approximation one must also approximate this last addition operation. Hence, we conjecture that (including the approximation of the addition) the bias cannot be more than 2^{-20}.

Example 2. Consider an approximation of the E-function which uses local approximations $A_1, A_2, A_4, A_5, A_6, A_7$ (Line 2 in Table 6). Again, it is conceivable that such an approximation can set values for the involved masks so that $X_4 = X_1 \lll 13$, $X_6 \oplus (X_{13} \lll 5) \oplus (X_{16} \lll 10) \oplus (X_{19} \lll 10) = 0$, and $X_3 = X_7$, in which case the resulting global approximation is of the form

$$(X_2 \circ k_1) \oplus (X_5 \circ k_2) \oplus (X_{20} \circ L) \tag{6}$$

A similar global approximation can be obtained from Line 12 in Table 6, except that in that case we also get $X_2 = X_5 = 0$. As in the previous example, the problem here too is to assign values to the masks so as to get an approximation with non-zero bias. This implies that X_{19} is zero everywhere except in the lower five bits, and that $X_8 = X_{12} = X_{13} = X_{14} = X_{15} = X_{16} = X_{17} = X_{18}$. One such approximation is obtained from Line 12 by using the parity approximation for the S-box (i.e., $X_7 = 0^{32}, X_8 = 1^{32}$). This approximation has bias of 2^{-5}, but it interacts very badly with the final addition of Line L into the data line (as the bias of the parity approximation for addition is only 2^{-17}). In general, in every approximation of the form (6) we must have either of two cases:

1. The masks $X_{13} \lll 5$ and X_{16} cancel each other everywhere except in the lowest five bits. Since we must have $X_{13} = X_{16}$ then it means X_{16} includes a 5-periodic non-zero 30-bit sub-word. Hence X_{20} must also include such a 30-bit sub-word. This, in turn, means that the bias of the approximation of the final addition cannot be more than 2^{-7}. Also, it means that X_{20} is either 1^{32}, or else it is non-periodic. In the first case, the addition approximation has bias of 2^{-17}, and in the second case the rotation approximation has bias of 2^{-6}. In any case, approximating the two operation has bias of at most 2^{-12}. Finally, we conjecture that the approximation of the S-box has bias of at most 2^{-4}, so the total bias of the E-function approximation is at most 2^{-15}.

2. The masks $X_{13} \lll 5$ and X_{16} do not cancel each other in the higher bits. In this case the approximation (6) must use local approximations A_1, A_2 (for the $+, \times$ operations). Here we must have $X_3 = X_7 \neq 0$, and so X_3 must be zero everywhere except in the lowest 9 bits. This implies that X_1 must be zero everywhere except in the lowest 9 bits, and since $X_4 = X_1 \lll 13$ then X_4 is zero everywhere except in bit positions 13..21. Hence we must approximate the "middle bits" of the multiplication input, and such approximations again seem to have only a very small bias. Here too we conjecture that the total bias of the E-function approximation is at most 2^{-15}.

I/O values	Largest bias	Comments
L	2^{-15}	Example 2 above
M	2^{-20}	Example 1 above
L,M	2^{-20}	2^{-15} as in Example 2 and 2^{-6} for the $\ll +$ on Line M
L,R I,L I,L,R	2^{-8}	2^{-6} for the $\ll +$ on Line L and 2^{-3} for the S-box
M,R	2^{-7}	2^{-6} for the $\ll +$ on Line M and 2^{-2} for the initial $+$
L,M,R I,L,M I,L,M,R	2^{-13}	2^{-6} for each of the $\ll +$ on lines L,M and 2^{-3} for the S-box
I,M I,M,R	2^{-6}	2^{-6} for the $\ll +$ on Line M
I,R	$1/2$	lower-bits of the multiplication

Table 7: Bias of approximations for the E-function

We note that approximations of the form (5) and (6) are the only approximations of the E-function (with non zero bias) that include only a single value from $\{I,L,M,R\}$. The other approximations of the E-function can be analyzed similarly to these two examples. In Table 7 we list the approximations of the E-function by the subset of the values $\{I,L,M,R\}$ which they include. With each subset, we list our estimate for the highest possible bias which can be obtained with this subset.

Approximating combinations of the basic operations. One way to refine the analysis above is to approximate several basic operations together, *taking into account the fact that the inputs to these operations are not independent.* For instance, one may try to combine approximations A_5, A_6, A_7, using the fact that a "self rotate" (i.e., $w_2 = w_1 \ll w_1$) operation has some small bias. We note however, that in the E-function one has to also take into account the value from approximation A_5. The best approximation of this kind requires the mask X_{20} to be periodic with Hamming weight at least 6, and this approximation has bias at most 2^{-5}.

4.1.3 Linear approximations of the keyed transformation

Below we provide a conservative bound (not a proof), showing that the data complexity of linear attacks against the keyed transformation phase of MARS exceeds 2^{128}. For this estimate we ignore most of the fine structure of the cipher, and only consider its graph structure. It is likely that taking into consideration more of the fine structure will improve these bounds considerably. In the analysis it will be convenient to consider four consecutive rounds at a time. We refer to four consecutive rounds as a "super-round" of the keyed transformation. Namely, in this terminology the keyed transformation consists of four super-rounds, each consisting of four rounds.

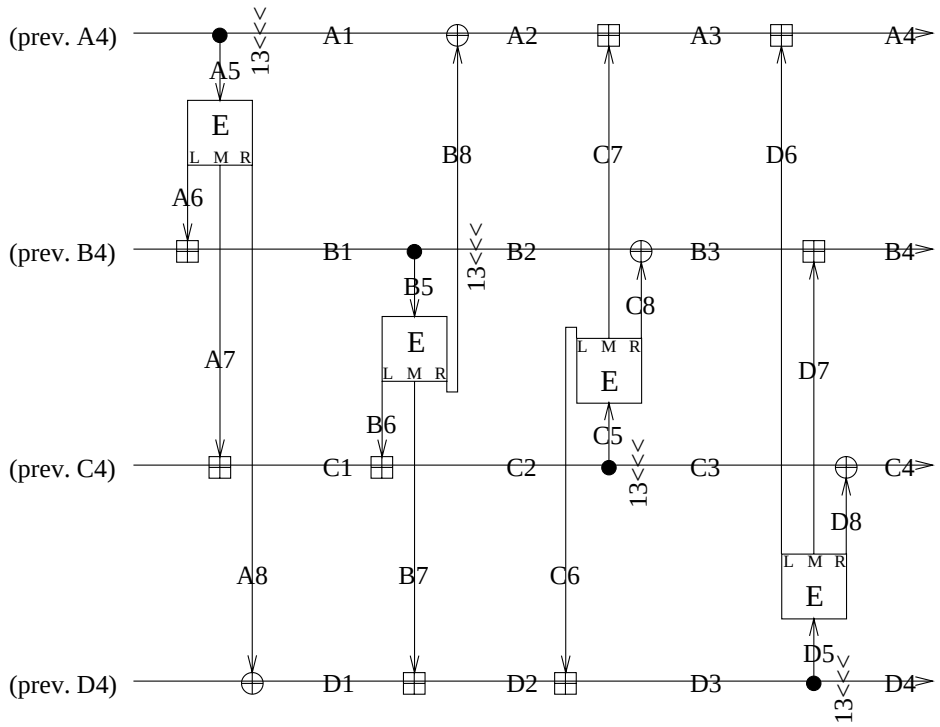

Figure 8: Labeling of the lines in the keyed transformation: $\oplus$ denotes exclusive-or and $\boxplus$ denotes addition.

The graph structure of the keyed transformation. The graph structure of one super-round is depicted in Figure 8. In the analysis we use a labeling of the lines of the keyed transformation, and this labeling is also given in Figure 8. Within each super-round, each data line consists of four segments (where a segment represents the value of this data line between two operations). The four segments of the first line are denoted by $A1 \ldots A4$, those of the second line by $B1 \ldots B4$, etc.

The graph structure of the keyed transformation consists of four copies of this super-round graph. In the last two copies, outputs L and R of every E-function are swapped (so, for example, Line $A6$ is added to Line $D1$ and Line $A8$ is xored into Line $B1$). In the description below, we refer to lines in the different super-rounds using subscripts. For example, the input to the second E-function in the first super-round will be denoted $B5_1$, and the value of the fourth data line at the end of the last super-round will be denoted $D4_4$. With this notation, the four input words to the keyed transformation are denoted $A4_0 \ldots D4_0$ and the four output words are $A4_4 \ldots D4_4$.

A little more formally, we have a graph[9] whose vertices are the various operations in the keyed transformation (inputs, outputs, copy operations, additions, xors and E-functions), and with edges that are labeled by

$$\{A4_0, B4_0, C4_0, D4_0\} \cup \{Ai_j, Bi_j, Ci_j, Di_j \ : \ 1 \le i \le 8, \ 1 \le j \le 4\}$$

In the analysis we consider global approximations for the keyed transformation phase which consist of local approximations for the various operations. Such approximations correspond in a natu-

[9] For the analysis below it is convenient to ignore the edge directions and think of the graph as undirected.

ral way to subsets of the edges in the graph above: A global approximation corresponds to the set of all edges whose values are used in its local approximations (where we say that the value of an edge is used in an approximation if the mask of this line in non zero).[10]

Clearly, not every subset of edges correspond to an approximation with non-zero bias. For example, any approximation which contains the edge $A1_j$ must also contain the edges $B8_j, A2_j$ (since approximations which only consider one or two of the three values incident to an exclusive-or always have zero bias). Also, we are only interested in approximations which contain at least one input value and at least one output value. Hence we have the following definition:

Definition 5 *We say that a subset S of the edges is* valid, *if it satisfies the following:*

1. *S contains at least one input edge (either $A4_0, B4_0, C4_0$ or $D4_0$) and at least one output edge (either $A4_4, B4_4, C4_4$ or $D4_4$).*
2. *If S contains an edge which is incident to an xor or an addition operation, then it also contains the other two edges incident to this operation.*
3. *If S contains an edge which is incident to a copy operation, then it contains at least one of the other two edges incident to this operation.*

   *(Notice that it may contain both. For example, an approximation which uses two bits from B*1 *can approximate one of them using a bit in B*2 *and the other using a bit in B*5.*)*
4. *If S contains either input edge I or output edge R of an E-function, then it contains at least one other edge incident to this E-function.*

To devise a bound, we identify with each approximation a valid set of edges, and then consider the edges incident to the E-functions in this set and use Table 7 to bound the bias of this approximation. In particular, we consider the edges *L* and *M* of the E-functions in the graph (these edges correspond to approximations of the combination of rotation followed by addition). We call these edges the *rotation edges*. We argue the following:

Assertion 6 *For any approximation A of the keyed transformation phase, the bias of A, as computed from the Piling-up lemma, is at most* 2^{-69}.

Reasoning: Let *S* be the valid subset of edges corresponding to the approximation *A*. We consider two cases:

1. For each E-function in the graph structure of MARS, *S* contains either zero or at least two edges incident to this E-function. A search of the graph structure of the keyed transformation verifies that in this case *S* must contain at least three rotation edges in every super-round, and that at least one of these rotation edges must be an *L* edge. From Table 7 we see that every occurrence of an *M* edge has bias at most 2^{-6} and every occurrence of an *L* edge has bias at most 2^{-8}. Using the Piling-up lemma, the bias of approximating one super-round is at most 2^{-18} and the bias of approximating the keyed transformation is at most 2^{-69}.

[10] Of course, there are many different approximations which correspond to the same subset of the edges, depending on the actual values of the masks.

2. There are E-function for which S contains a single edge. From Table 7 it follows that the corresponding local approximations must be of the form (6) or (5), which have bias of only 2^{-15} or 2^{-20}, respectively. Moreover, a search of the graph structure of the keyed transformation verifies each E-function like this only "saves" at most one occurrence of a rotation edge, hence the resulting bias is even smaller than 2^{-69}.

□

4.2 Differential analysis

One of the main considerations in the design of the E-function was to combine the data-key multiplication, S-box lookup and data-dependent rotations so as to maximize the resistance to differential attacks. Below we start by analyzing the differential behavior of the data-key multiplication operation, then use this to analyze the differential behavior of the E-function, and then provide analysis for the entire keyed transformation phase of MARS. Finally, we also provide some analysis of the differential properties of the mixing phase.

4.2.1 Analysis of the data-key multiplication

Conventions. In the description below we view 32-bit words as integers between 0 and $2^{32}-1$. All the arithmetic operations are considered modulo 2^{32}. We identify positive integers with their binary representation. If w is a word, then we denote by $w_{j..i}$ the sub-word consisting of the bits in positions i through j in w.

Assertion 7 (data-key multiplication) *Let $d \neq d'$ be two fixed 32-bit data words such that the lowest bit in which d, d' differ is in position i. Assume without loss of generality that $d' > d$ and denote $d' - d = st10^i$, where t is a single bit and s is a $(30-i)$-bit word.*

Let k be a 32-bit key word, which is chosen at random subject to the constraint that its two lowest bits are set to one. Then the difference in the product is of the form

$$\Delta p = (d' \cdot k) - (d \cdot k) \;=\; u\bar{t}10^i$$

where $\bar{t}$ is the complement of the bit t, and u ranges uniformly over all possible $(30-i)$-bit words.

Proof: We can write $d' - d = 2^i + t2^{i+1} + s2^{i+2}$ and also $k = 3 + 4x$, where x is a uniformly distributed 30-bit integer. Then we have

$$\begin{aligned}(d' \cdot k) - (d \cdot k) &= (2^i + t2^{i+1} + s2^{i+2}) \cdot (1 + 2 + 4x) \\ &= 2^i \;+\; 2^{i+1}(1+t) \;+\; 2^{i+2}(t + 3s + (1 + 2t + 4s)x)\end{aligned}$$

and the proof follows since s, t are fixed and x is random. ■

Corollary 8 *Let $d \neq d'$ be two fixed 32-bit data words such that the lowest bit in which d, d' differ is in position i, and let k be a 32-bit key word, which is chosen at random subject to the constraint*

that its two lowest bits are set to one. Also, let l, m be two indices such that $i+2 \le l \le m \le 31$, and denote $n = m-l+1$. Then for every n-bit word s, we get

$$\Pr_k\left[(d'\cdot k)_{m..l} - (d\cdot k)_{m..l} = s \pmod{2^n}\right] \le 2^{-n+1}$$

Proof: This follows immediately from Assertion 7, since the expression $(d'\cdot k)_{m..l} - (d\cdot k)_{m..l}$ always equal either $[(d'\cdot k)-(d\cdot k)]_{m..l}$ or $[(d'\cdot k)-(d\cdot k)+1]_{m..l}$ (depending on the carry into the l'th bit position). ■

Corollary 9 *Let $d \ne d'$ be any two fixed 32-bit data words, and denote by i the least significant bit in which d, d' differ. Then*

$$\Pr_k[(k\cdot d)_{31..22} = (k\cdot d')_{31..22}] \le \begin{cases} 2^{-9} & \text{if } i \in \{0\ldots20\} \\ 2^{-8} & \text{if } i = 21 \\ 0 & \text{if } i \in \{22\ldots31\} \end{cases}$$

where the probability is taken over the choice of k as a 32-bit word with the two least significant bits set to 1.

Corollary 9 explains the usage of the top ten bits of the product as the source-bits for the data-dependent rotation: If we feed two different data words into the data-key multiplication, then with probability of at least $(1-2^{-8})$ (taken over the choice of the key) the top ten bits will not agree, in which case we get rotations by different amounts in the E-function.

Xor-differences. The behavior of the data-key multiplication with respect to xor-differences is more involved than its behavior with respect to subtraction. Still, we can prove the following bound:

Assertion 10 *Let $d \ne d'$ be two fixed 32-bit data words such that the lowest bit in which d, d' differ is in position i, and let k be a 32-bit key word, which is chosen at random subject to the constraint that its two lowest bits are set to one. Also, let l, m be two indices such that $i+2 \le l \le m \le 31$, and denote $n = m-l+1$. Then for every n-bit word s, we get*

$$\Pr_k\left[(d'\cdot k)_{m..l} \oplus (d\cdot k)_{m..l} = s\right] \le 2^{w(s)-n+1}$$

where $w(s)$ is the Hamming weight of s, not including the most significant bit *(e.g., $w(10110) = w(00110) = 2$).*

Proof: The proof follows from Corollary 8 since there are only $2^{w(s)}$ words s' such that $(d'\cdot k)_{m..l} - (d\cdot k)_{m..l} = s' \pmod{2^n}$ is consistent with $(d'\cdot k)_{m..l} \oplus (d\cdot k)_{m..l} = s$. The reason that we do not count the most significant bit is that 2^{n-1} and -2^{n-1} are equal modulo 2^n. ■

Assertion 10 gives a good bound on the probability of output xor-differences which has very few 1's, but it only gives an upper bound of a 1 on differences which are all 1's. To some extent, this is the best bound possible, since for $d = 1, d' = -1$ we get $\Delta p = \Delta d$ with probability 1. Although we still do not have a comprehensive analysis for the differential behavior of the multiplication with respect to xor-differences, below we provide partial analysis for some special cases.

Case 1. The data words d, d' differ in the least significant bit. Here we show that as the key k varies, the 30 higher bits in the output difference assume every 30-bit value exactly once. For this, we prove that once the bits $k_{i-1} \ldots k_0$ are fixed, varying the bit k_i varies bit i in the output difference without affecting any of the lower bits in the difference: Fix bits $k_{i-1} \ldots k_0$ to any value, and denote $\hat{k} = k_{i-1} \ldots k_0$ and $p = \hat{k}d, p' = \hat{k}d'$. Consider now what happens when we vary the value of k_i. If we set $k_i = 0$ then p, p' will not change, and therefore bit i in the output difference will remain $p_i \oplus p'_i$. On the other hand, if we set $k_i = 1$ then we add d, d' (shifted by i) to p, p' respectively, as shown below.

$$\begin{array}{cccccccc} & p_{30+i} & \cdots & p_i & p_{i-1} & \cdots & p_0 \\ + \; d_{31} & d_{30} & \ldots & d_0 & 0 & \ldots & 0 \end{array} \qquad \begin{array}{cccccccc} & p'_{30+i} & \cdots & p'_i & p'_{i-1} & \cdots & p'_0 \\ + \; d'_{31} & d'_{30} & \ldots & d'_0 & 0 & \ldots & 0 \end{array}$$

Since we only add zeros to positions $i-1 \ldots 0$, then nothing changes in these positions. In position i, however, we add d_0, d'_0, respectively, and since $d_0 \neq d'_0$, then output bit i is necessarily flipped.

Case 2. d is even and $\Delta = \star\star\star\ldots10$. In this case d' is also even, so we can apply the analysis above to the 31-bit integers $d/2, d'/2$ (which differ in the l.s.b.). Hence we get the same result as above for the high 29 bits of the input difference. Similar analysis can be used when d is a multiple of 2^i and Δ is of the form $\Delta = \star\star\star\ldots10^i$

Case 3. d is odd and $\Delta = 1^{31}0$. In this case, $d' = d \oplus \Delta = -d$ and so $kd' = -kd = kd \oplus \Delta$. Similarly, when d is odd and $\Delta = 01^{30}0$, we get $d' = 2^{31} - d$ and so $kd' = 2^{31} - kd = kd \oplus \Delta$.

We extended the above analysis using experimental results. In our experiments we worked with word sizes up to 14 bits. In each experiment we fixed the input xor difference, and then went over all possible keys and all possible data pairs with this xor difference, measuring the probabilities of the various output differences. These experiments suggest the following behavior:

- When the input difference is of the form $\Delta d = x01^i0$ with $|x| = n$ (i.e. the l.s.b. is 0, then some 1's, then a 0, and then n don't-cares), the most likely output differences are all the differences of the form $\Delta p = u01^i0$, where u ranges over all possible n-bit values. Each of these output difference has probability of 2^{-n-1} (so their total probability is $1/2$).

  Notice that for 32-bit words and $i = 30$, this matches exactly the analysis in Case 3 above, since when d is odd (which happens with probability $1/2$), we get $\Delta p = \Delta d$ with probability 1.

- As we add more zeros in the low-order bits of the input difference Δd, we get similar patterns with probabilities that are close (but not equal) to a factor of $1/2$ for each additional zero. Namely, when the input difference is of the form $\Delta d = x01^i0^j$ with $|x| = n$, the most likely output differences are all the differences of the form $\Delta p = u01^i0^j$, and each one occurs with probability close to (but slightly larger than) 2^{-n-j}.

  This pattern is only maintained as long as i, the number of 1's, is "large enough". As i decreases, the deviations from this pattern increase. In our experiments with 14-bit words, as i decreased below 7 or 8, the pattern itself disappeared and we could not recognize any pattern in the output differences.

A comment about the key words. In the analysis above we assumed that the key word is chosen uniformly at random with the lowest two bits set to one. In fact, in the key generation process we also impose the condition that the key word does not contain ten consecutive 0's or 1's. The effects of this condition on our analysis are as follows.

- This condition ensures that a single-bit difference in the input to multiplication *always causes some difference in the top ten bits* of the output. Hence, we are guaranteed that if we have a single-bit difference in the input to the E-function, we get a different rotation amount on at least one of the output lines.
- Recall that in the key expansion process, the probability of any 20-bit pattern grows by at most a factor of 1.23, and for 10-bit patterns the factor is about 1.06. Since our analysis depends only on short patterns in the product, the probabilities which were calculated above cannot grow more than by a factor of 1.06 (or 1.23). In the rest of the analysis we ignore these small factors.

Key probability vs. data-probability. The analysis above assumes that the data words are fixed and the key is chosen at random (subject to the given constraints). In a differential attack, however, it is the key that is fixed and the data words are chosen at random (with a fixed difference pattern). We therefore would like to say something about the probability of a certain pair of input and output differences, when the key is fixed and the probability is taken over the data.

For the subtraction difference, $\Delta = d - d'$, once the key k and the input difference Δ_{in} are fixed, this completely determines the output difference, $\Delta_{\text{out}} = k\Delta_{\text{in}}$ with probability 1. For the xor difference $\Delta = d \oplus d'$ this is not the case. Assuming that the lowest '1' in Δ_{in} is not in the top ten bits (which is the interesting case for MARS), there are only two pairs of input and output xor-differences with probability $1/2$ for a fixed key (specifically $\Delta_{\text{out}} = \Delta_{\text{in}} = 1^{31}0$ and $\Delta_{\text{out}} = \Delta_{\text{in}} = 01^{30}0$). All the other pairs have probability of $1/4$ or less. It also seems that the probability of a pair further decreases when either Δ_{out} or Δ_{in} contains more 0's, although we still do not have a rigorous analysis of this behavior.

4.2.2 Analysis of the E-function

We analyze the behavior of the E-function with respect to xor differences. The structure of the E-function is depicted again in Figure 9. In this figure we also label the lines, so that in the analysis below we can refer to the differences on specific lines. There are three cases to consider, depending on the position of the lowest '1' in the input difference of the multiplication:

1. If lowest '1' in the input difference to the data-key multiplication (Δ_{13}) is in positions 31..22, then we are guaranteed to get a different rotation amount on at least one of the rotation lines (L or M). Even in this case, we may get a characteristic with non-trivial probability if we assume that the actual values in the E-function after the key addition are periodic.

   Specifically, assume that the input difference to the E-function is $\Delta_{\text{in}} = 1 \ll 18$. Hence, after rotating it by 13 the input difference to the multiplication is $\Delta_{13} = 1 \ll 31$, and so the output

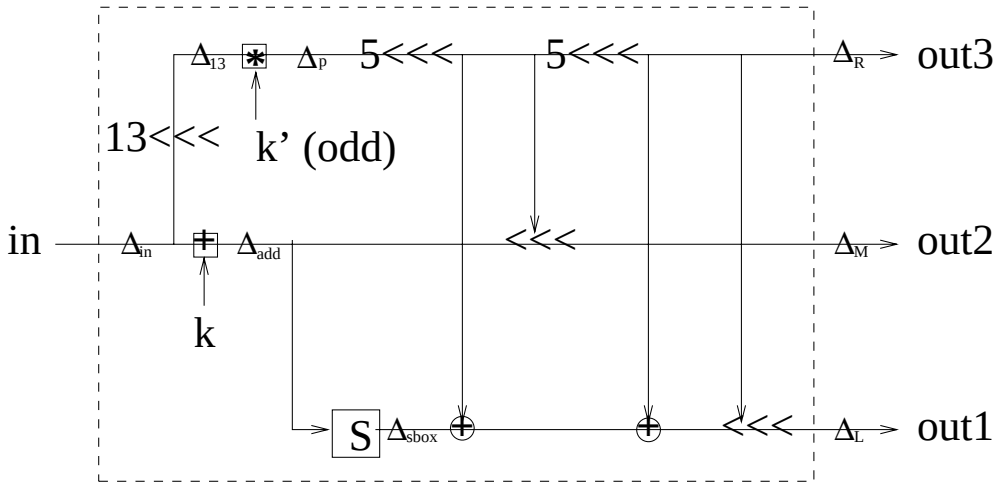

Figure 9: Another labeling of the lines in the E-function

difference is also $\Delta_p = 1 \ll 31$. Therefore there is a difference of 16 in the rotation amount on Line M between the two invocations of the E-function.

With probability $1/2$, the difference after the key addition is still $\Delta_{\text{add}} = 1 \ll 18$. With additional probability 2^{-15}, one of the two actual values is 16-periodic (and the other is 16-periodic except for the bit in position 18). It is easy to verify that if this happens, then the output differences of the E-function will be $\Delta_R = 1 \ll 9$, $\Delta_M = 1 \ll r_2$ and $\Delta_L = 100001 \ll r_1$ for random rotation amounts r_1, r_2. This characteristic is shown in the first column of Table 8.

We remark that although in principle one can also consider values with smaller periods, the probability of those is so small that the characteristics obtained this way are irrelevant (for example, the probability of obtaining an 8-periodic value is 2^{-24}).

2. If lowest '1' in Δ_{13} is in positions 21..13, then there is a difference in the low 9 bits of the input difference Δ_{in}, so there must be a difference in the input to the S-box. Also, recall that if Δ_{in} has just a single '1' that we are guaranteed to get different rotation amounts. Hence, we assume that it contains at least two '1's, which means that any specific pattern can go through the key addition with probability at most $1/4$. Below we denote the difference after the key addition by Δ_{add} and the S-box difference by Δ_{sbox}.

   If we denote the input difference to the multiplication by $\Delta_{13} = t0^{13}$ where $t = t'10^i$, then the product difference will be $\Delta_p = \$..\$10^{13+i}$ where the $(18-i)$-bit string $\$..\$$ is distributed by the differential behavior of the multiplication operation. In particular, with probability of at least $1-2^{-8}$ the top ten bits will not be all zero, and then we will have different rotation amounts on at least one of the lines L, M.

   With probability $\leq 2^{-9}$, the top ten bits will be all zero, namely $\Delta_p = 0^{10}s10^{13+i}$ (where s is of length $8-i$ and it is distributed by the differential behavior of the multiplication operation). If this happens, then the output differences will be $\Delta_R = s10^{13+i}$, $\Delta_M = \Delta_{\text{add}} \ll r_2$ and $\Delta_L = (\Delta_{\text{sbox}} \oplus \Delta_R \oplus (\Delta_R \ll 5)) \ll r_1$ for random rotation amounts r_1, r_2. This characteristic is shown in the second column of Table 8.

3. If the lowest '1' in Δ_{13} is in positions 12..0, then we denote $\Delta_{\text{in}} = vwu, \Delta_{13} = wuv$ with $|v| = 13, |w| = 10, |u| = 9$. The case where $u \neq 0$ is similar to the previous case (except that the probabilities are lower). In the case where $u = 0$, the product difference will be $\Delta_p = \$..\$10^i$ where i is the position of the lowest '1' in v and the $(31-i)$-bit string \$..\$ is distributed by the differential behavior of the multiplication operation. With probability of at least $1 - 2^{-9}$ the top ten bits are not all zero, and then we have different rotation amounts on at least one of the lines L, M. With probability $\leq 2^{-9}$, the top ten bits are all zero, namely $\Delta_p = 0^{10}s10^i =$, where s is of length $21 - i \geq 9$ and it is distributed by the differential behavior of the multiplication operation.

   Since the lowest 9 bits of Δ_{in} are zero, then there is no difference in the input to the S-box. Since we assume that Δ_{in} contains at least two '1's, then any specific pattern can go through the key addition with probability at most $1/4$. The differences in the output of the E-function is therefore $\Delta_R = s10^{10+i}$, $\Delta_M = \Delta_{\text{add}} \lll r_2$ and $\Delta_L = (\Delta_R \oplus (\Delta_R \ggg 5)) \lll r_1$ for random rotation amounts r_1, r_2. This characteristic is shown in the third column of Table 8.

Key probability vs. data-probability. The probabilities quoted in Table 8 are taken over the random choice of both the key and the data. It is also useful to know how this probability can be broken to key vs. data probability, since in general it is the data-probability that corresponds to the data-complexity of an attack.

In the table we list our estimate for the largest possible data-probability, and the corresponding key-probability. For example, in the third column we list key probability of 2^{-7}, and data probability of 2^{-2}. This means that there may be a property of keys that holds with probability 2^{-7}, such that if the key has this property then one out of four data pairs satisfies the characteristic in this column. However, there is no property of keys (with any probability) that causes a larger fraction of the data pairs to satisfy this characteristic. We also note that the random rotation amounts are completely data-dependent. Namely, for any fixed key and fixed input difference, when you vary the data pairs, the rotation amounts vary uniformly between 0 and 31.

4.2.3 Analysis of the keyed transformation phase

Using the results in Table 8 we now proceed to analyze the differential behavior of the keyed transformation phase. We first describe a few attempts to devise high-probability characteristics of the keyed transformation. Then we use the intuition gained in these attempts to make a heuristic argument suggesting that there are no high-probability characteristics, and finally we devise a crude bound on the probability of any characteristic. As with linear analysis, here too we consider "super-rounds" consisting of four consecutive rounds of the keyed transformation.

Active and passive rounds. Since the characteristics of the E-function have rather low probabilities (at most 2^{-9}, with two random rotation amounts and a few random carry bits), we would like to have as few rounds with non-zero input difference as possible. Below we say that a round is *active* if it has non-zero input difference, and is *passive* otherwise. Since every active round produces three non-zero output differences, it is not possible to maintain a characteristic with only one

	Type-1	Type-2	Type-3
$\Delta_{\text{in}} =$	$1 \ll 18$	$0^{13}wu$ $(\lvert w\rvert = 10, \lvert u\rvert = 9)$	$vw0^9$ $(\lvert v\rvert = 13, \lvert w\rvert = 10)$
where		Δ_{in} has at least two '1's $u = u'10^i$	Δ_{in} has at least two '1's $v = v'10^i$
Probability	2^{-16} (key : 1, data : 2^{-16})	2^{-8} (key : 2^{-6}, data : 2^{-2})	2^{-9} (key : 2^{-7}, data : 2^{-2})
$\Delta_{\text{add}} =$	Δ_{in}	similar to Δ_{in}	similar to Δ_{in}
$\Delta_{\text{sbox}} =$	0	$S[\Delta_{\text{add}}\lvert_{8..0}]$	0
$\Delta_{13} =$	$1 \ll 31$	$uw0^{13}$	$w0^9v$
$\Delta_p =$	$1 \ll 31$	$0^{10}s10^{13+i}$	$0^{10}s10^i$
$\Delta_R =$	$1 \ll 9$	$s10^{23+i}$	$s10^{10+i}$
$\Delta_M =$	$\Delta_{\text{add}} \ll r_1$	$\Delta_{\text{add}} \ll r_1$	$\Delta_{\text{add}} \ll r_1$
$\Delta_L =$	$100001 \ll r_2$	$\left(\begin{array}{c}\Delta_{\text{sbox}} \oplus \Delta_R \\ \oplus(\Delta_R \gg 5)\end{array}\right) \ll r_2$	$(\Delta_R \oplus (\Delta_R \gg 5)) \ll r_2$
Comments	input is periodic	difference in the S-box	most probable pattern

r_1, r_2 – random and independent rotation amounts, s – a random word

Table 8: The differential behavior of the E-function

active round per super-round. In the attempts below we therefore try to maintain characteristics with two active rounds per super-round.

First attempt: two adjacent active rounds. In the first attempt we try to maintain the invariant that only the first two rounds in each super-round are active. We try this using a Type-3 characteristic of the E-function (third column in Table 8). This attempt is depicted in Figure 10. Assume that the input difference to the keyed transformation phase is $(a, b, 0, 0)$, where $a = vw0^9$ with $|v| = 13, |w| = 10$, and b is an arbitrary input difference. The characteristic proceeds as follows:

1. The input difference to the first E-function is a, which matches the Type-3 characteristic in the third column of Table 8 (where i, which is the bit-position of the lowest '1' in v, is probably no more than one or two). With probability at most 2^{-9} we get characteristic of Type-3. Namely, the output difference on Line R is $x = s0^{10}$, the difference on Line M is $y \approx a \ll r_2$ for a random rotation amount r_2 (but probably $y \neq a \ll r_2$ because of the carry bits in the key addition), and the difference on Line L is $z = x \oplus (x \ll 5)) \ll r_1$ for a random rotation amount r_1.

   Also, the difference on Line A after the first round is $a \ll 13 = w0^9v$.

2. With some probability (denoted p) the lowest 9 bits of the difference x on Line L cancel the low 9 bits of the difference b on Line B. Hence, the input difference into the next round becomes $b' = v'w'0^9$ with $|v'| = 13, |w'| = 10$.

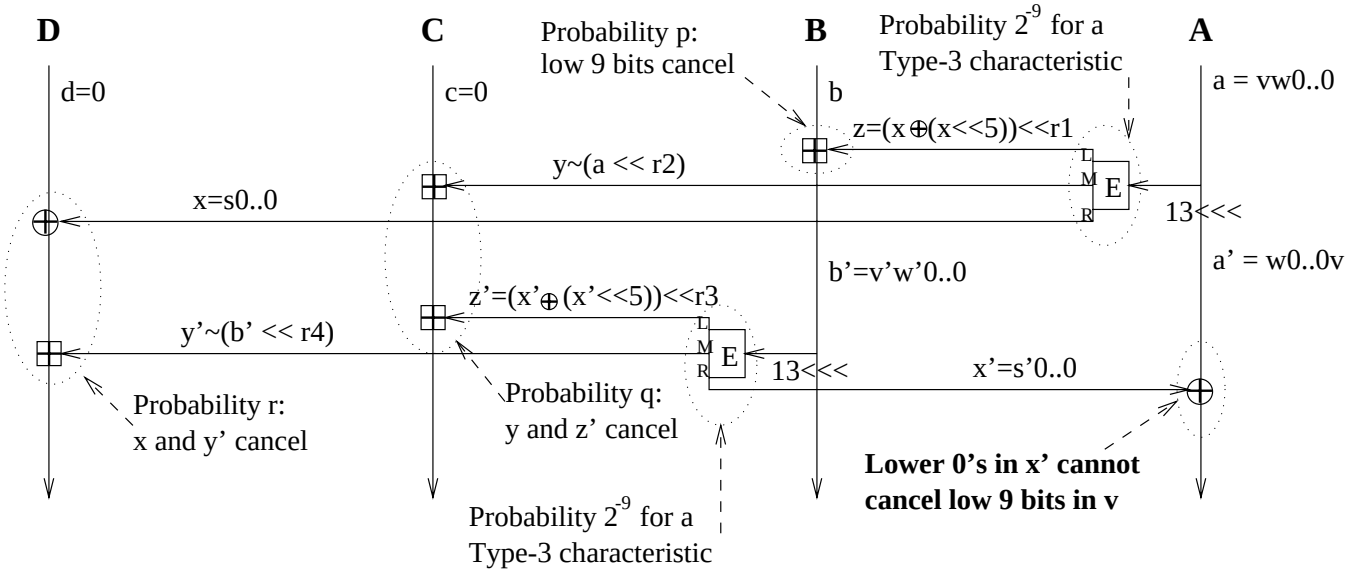

Figure 10: First attempt to devise a characteristic of the keyed transformation.

3. With another 2^{-9} probability we again have a Type-3 characteristic: the output difference on Line R is $x' = s'0^{10}$, the difference on Line M is $y' \approx b' \lll r_4$ for a random rotation amount r_4, and the difference on Line L is $z' = x' \oplus (x' \lll 5)) \lll r_3$ for a random rotation amount r_3.

4. With some small probability, the differences y and z' cancel each other, and the differences x and $\approx y'$ also cancel each other.

   However, even if this happens, the output difference x' on Line R of the second E-function has ten or more '0's in the lowest bits, and so it cannot cancel the low bits of the difference $w0^9v$ on Line A. Hence, some of the low nine bits in A remain non-zero, and so this characteristic cannot be maintained. (Of course, we could make the assumption that the low 9 bits in v are also '0', but then we could not maintain these bits as '0's).

One problem with the above attempt is that the difference on the R output line will always have the lowest ten bits set to zero (or else there will be a different rotation amount on one of the other lines), and hence it cannot be used to counter the effect of the fixed rotation by 13 on the source line. Hence, in the attempts below we try to maintain characteristics in which the active rounds are not adjacent (e.g., lines A and C).

Second attempt: Using a Type-1 characteristic of the E-function. In the next attempt we try and keep active only the first and third rounds in each super-round, this time using a Type-1 characteristic of the E-function (i.e., relying on periodic inputs). Assume that the input difference to the keyed transformation phase is $(a, b, 0, 0)$, where $a = 1 \lll 18$ and $b = 100001$. The characteristic is depicted in Figure 11 and it proceeds as follows:

1. With probability 2^{-16} we get a Type-1 characteristic, which means that the output difference on Line R is $1 \lll 9$, the output difference on Line M is $1 \lll r_2$ and the output difference on Line L is $100001 \lll r_1$, where r_1, r_2 are random rotation amounts.

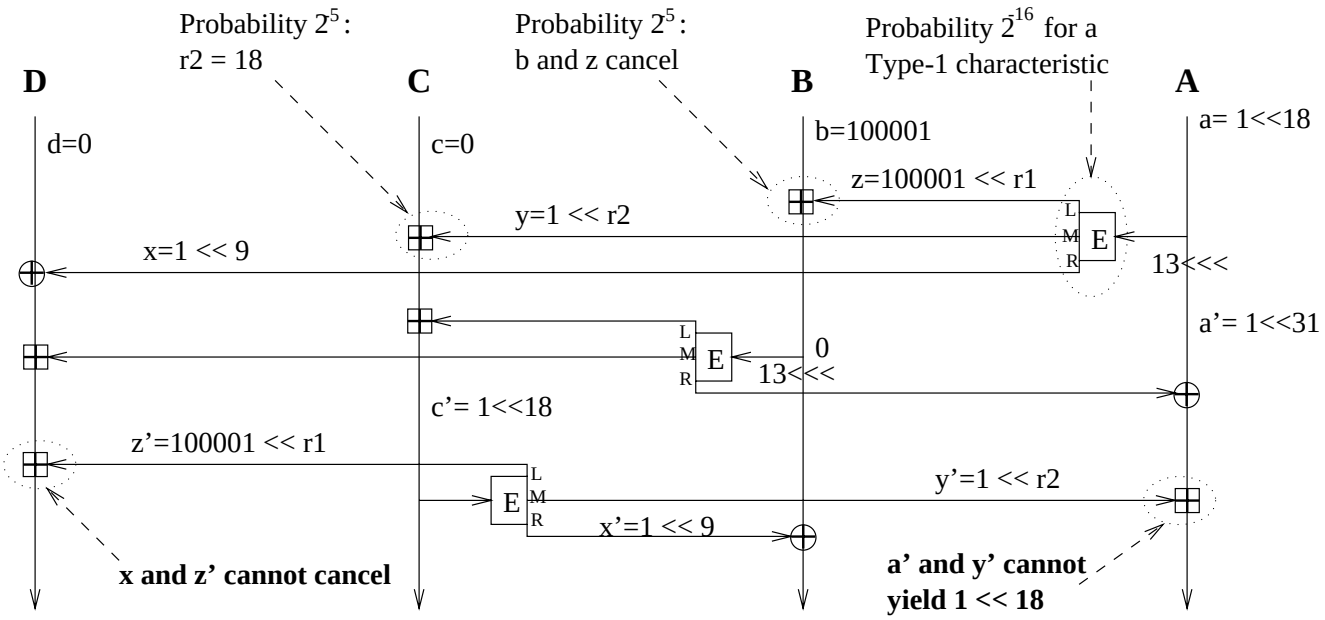

Figure 11: Second attempt to devise a characteristic of the keyed transformation.

The difference on Line A after the first round is $1 \lll 31$.

2. With probability 2^{-5} we have $r_1 = 0$, so the differences on lines L and B cancel each other (assuming the carry bits behave correctly), and the second round becomes passive.

3. With probability 2^{-5} we have $r_2 = 18$, and then the input difference to the third round again matches the Type-1 characteristic of the E-function.

4. With probability 2^{-16} we again get a Type-1 characteristic, so the output difference on Line R is $1 \lll 9$, the output difference on Line M is $1 \lll r_4$ and the output difference on Line L is $100001 \lll r_3$, where r_3, r_4 are random rotation amounts.

   However, the differences on lines R of the first round ($1 \lll 9$) and Line L of the third round ($100001 \lll r_3$) cannot cancel each other, so the fourth round must also be active. Similarly, the differences on lines A ($1 \lll 31$) and Line M of the third round ($1 \lll r_4$) cannot be combined to yield $1 \lll 18$, which is what needed for a Type-1 characteristic of the E-function.

This attempt demonstrates the difficulty of working with the Type-1 characteristic of the E-function: not only does this characteristic have low probability (2^{-16}), it is also very fragile in that it requires that the input difference contains a single '1' in a fixed location. This does not fare well with the fixed rotation on the data lines.

Third attempt: Type-3 characteristics in non-adjacent rounds. Below we try to keep only the first and third rounds active, but with the Type-3 characteristic of the E-function. Assume that the input difference to the keyed transformation phase is $(a, b, c, 0)$, where $a = vw0^9$ with $|v| = 13, |w| = 10$, $b = t0^{10}$ (with $|t| = 22$), and $c = w'0^9v'$ with $|v'| = 13, |w'| = 10$ (c is essentially a rotated version of a). The characteristic, depicted in Figure 12, proceeds as follows:

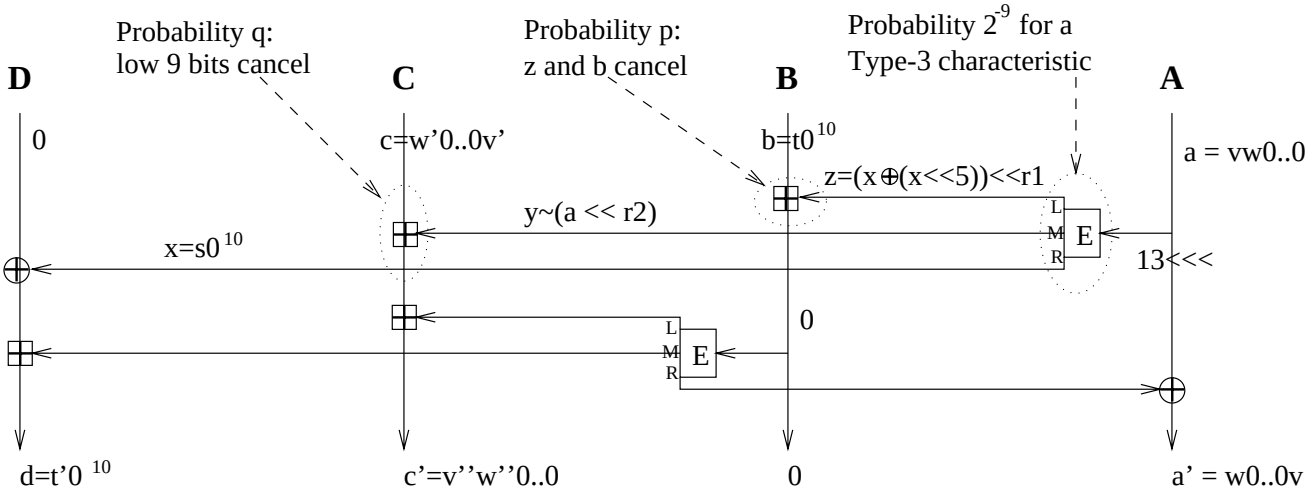

Figure 12: First attempt to devise a characteristic of the keyed transformation.

1. The input difference on Line *A* matches the Type-3 characteristic in the third column of Table 8 (where *i*, which is the bit-position of the lowest '1' in *v*, is probably no more than one or two).

   With probability at most 2^{-9} we get a Type-3 characteristic of the E-function: the output difference on Line *R* is $x = s0^{10}$, the difference on Line *M* is $y \approx a \lll r_2$ for a random rotation amount r_2, and the difference on Line *L* is $z = x \oplus (x \lll 5)) \lll r_1$ for a random rotation amount r_1.

   Also, the difference on Line *A* after the first round is $a \lll 13 = w0^9v$.

2. With some small probability (denoted *p*) the differences *y* and *b* cancel each other. We note that *y* contains at least 5 consecutive zeros and *b* contains at least ten consecutive zeros, so we can hope to get $p > 2^{-32}$. We return to this point later.

   If this happens, then the second round becomes passive.

3. With some other probability (denoted *q*) the low nine bits in *c* and *y* cancel each other. Here we note that although both *c* and *y* are known to have at least nine consecutive '0's, the '0's in *c are not* in the lowest bit positions, so this does not help the cancellation.

   If we get a cancellation in the low 9 bits, the input difference to the third round becomes $c' = v''w''0^9$ (with $|v''| = 13, |w''| = 10$). Notice that now the difference on Line *A* is $w0^9v$, the difference on Line *B* is 0 and the difference on Line *D* is $d = t'0^{10}$.

   Hence, we are in exactly the same situation as in the beginning of the characteristic and we can iterate it. We note that the same characteristic (with the same probabilities) works also for the backwards part of the keyed transformation, so in total we need eight iterations of this characteristic.

The above is therefore a plausible characteristic for the keyed transformation phase, with probability $(2^{-9} \cdot p \cdot q)^8$. However, we note that the same characteristic can be used *even if the E-function is*

replaced by an ideal $32 \to 96$ *expansion function*, with probability $(2^{-40})^8$. (For an ideal function we have probability 2^{-31} that the value on Line *B* cancels and probability 2^{-9} that the lowest nine bits on Line *C* cancel). Hence, this characteristic is useful to the cryptanalyst only if we can get $2^{-9} \cdot p \cdot q \gg 2^{-40}$.

To get a large value q (the probability of cancelling the low 9 bits in $c = w'0^9v'$) we can start with the differences a and c having *low Hamming weight*. In this case also $y \approx a \ll r_2$ will have low Hamming weight, and if the rotation amount r_2 is correct then we have a pretty good chance of cancellation. We therefore use the value $q = 2^{-5}$ in the calculations below.

As for the value of p, this is the probability that the differences z and b cancel each other. In general, in this characteristic the difference b is the output difference on the *R* line of E-function in some active round and z is the output difference on the *L* line of E-function in some other active round. Hence in general:

- The difference b is of the form $t0^{10}$, where t is determined by the output distribution of the data-key multiplication.
- The difference z is of the form $(x \oplus (x \ll 5)) \ll r$, with x being the *R* output from this 'other active round' and r is a random rotation amount.

So far we still do not have a rigorous analysis of the xor differential behavior of the data-key multiplication output, and so we cannot devise a rigorous bound for p. Instead, below we give a very informal argument that it is unlikely to get $p > 2^{-16}$. First, since the low ten bits of b are zeros, there is likely to be at most a single rotation amount r that causes b and y to cancel. Then, since b, x each contains about 22 non-zero bits, we estimate by at most 2^{-11} the probability that these bits are chosen in such a way so that b and $(x \oplus (x \ll 5)) \ll r$ actually cancel. Hence, we conjecture that $p \leq 2^{-16}$.

With these values for p, q, we have $2^{-9} \cdot p \cdot q = 2^{-30}$ which is only slightly better than the 2^{-40} we get for an ideal function. The probability of the characteristic on the full keyed transformation phase is therefore about $2^{-30 \cdot 8} = 2^{-240}$.

Can we do better? Below we give very informal arguments to the effect that the above characteristic is the best possible for MARS. We note the following

- To get a high-probability characteristic, one must use as few active rounds as possible. It is very unlikely that there exists a characteristic of the keyed transformation phase with less than two active rounds per super-round.
- Characteristics of the E-function with different rotation amounts on either line *L* or Line *M* have very low probability (except, perhaps, the Type-1 characteristic from Table 8). Hence it is unlikely that one can devise a high-probability characteristic of the keyed transformation using such characteristics of the E-function.

  Regarding the Type-1 characteristic in Table 8, it requires exactly one '1' in a particular position in the input difference. As was demonstrated by the second attempt from above, this cannot be maintained in the face of the fixed rotation amounts on the data lines in MARS.

- It is also unlikely that one can devise a high-probability characteristic including a difference in the S-box input and output (such as the Type-2 characteristic in Table 8), since the S-box output differences in general do not match any of the input differences in Table 8.
- Hence it seems that one must use the Type-3 characteristic of the E-function as the main building block for a characteristic of the keyed transformation.
- To maintain only two active rounds per super-round, one must arrange the outputs of the E-function in different rounds in pairs, so that in half of these pairs the two outputs completely cancel each other (with high probability) and in the other half the low 9 bits are cancelled.

  As was demonstrated in the first attempt above, it is not possible to have two adjacent rounds as the only active rounds in a super-round. This is because the *R* output line of the E-function cannot be used to cancel the low nine bits of another line (as its lowest ten bits are '0').

  Hence, one must have the *R* and *L* lines cancel each other, and the *M* line cancel the low nine bits in the input line (after the rotation by 13). As was demonstrated in the last attempt from above, this leads to a characteristic with probability $\leq 2^{-240}$.

Although the arguments above are quite speculative, we expect that the conclusion is still correct. Hence we estimate the security level of the keyed transformation phase against differential analysis to be at least 2^{240}. We comment that the data complexity which is associated with the above "best characteristic" is at least 2^{120}, and its key probability is at most 2^{-120}.

Devising a bound. Below we also provide a crude and much more conservative bound for the keyed transformation phase. For this bound we make only very weak assumptions on the way that characteristics of the E-function can be combined to construct a characteristic of the entire phase. Specifically, we assume that

1. Every characteristic of the keyed transformation uses at least two active rounds per super-round.
2. Every active E-function contributes a factor $\leq 2^{-12}$ to the differential probability (taken over both data and keys). This is because the highest-probability characteristic of the E-function has probability 2^{-9}, and each round contains three addition operations, each contributing (at least) one more factor of $1/2$.
3. Among the four random rotation amounts in each super-round, two must be fixed to specific amounts and the other two must be aligned. This contributes another factor of 2^{-15} for each super-round.

With these assumptions, we get a bound of $2^{-12 \cdot 8} \cdot 2^{-15 \cdot 4} = 2^{-156}$ on the probability of every characteristic of the keyed transformation. This bound implies data complexity of at least 2^{80} and key probability of at most 2^{-76}.

4.2.4 Analysis of the mixing phases

The purpose of the mixing phases in MARS is twofold:

- They provide better avalanche of the key bits than the keyed transformation, in the sense that stripping off mixing rounds requires guessing more effective key bits than stripping off rounds from the core.

- They are likely to break "input structures" that may be used in conjunction with the differential characteristics of the keyed transformation. For example, the differential analysis above suggests that input differences of small Hamming weight are useful in constructing characteristics of the keyed transformation. Therefore a potential attack may proceed by encrypting many plaintext blocks which lie in a Hamming sphere of small radius. Such a sphere of n words produces $\binom{n}{2}$ input pairs of small Hamming weight. This fact may be used to considerably reduce the data complexity of a differential attack.

  However, the mixing phases, being built out of S-boxes, make it harder to propagate such structures to the keyed transformation.

To gain some intuition into the structure of the mixing phase (and to explain some of the choices made in the design), we illustrate below two "sample attacks", on weakened versions of the mixing phase. To make the description of these attacks simpler, we consider a version of the mixing phase in which all the additions are replaced by xors (although similar attacks with slightly lower probabilities can also be devised against versions which include additions).

The role of the feedback additions. Recall that in the mixing phases we add one of the target words back into the source word after some of the mixing round. To demonstrate the importance of these "feedback additions" we describe below a simple attack against a version of the mixing phase which does not have these additions.

Let $\Delta_0 = S0[i] \oplus S0[j]$ be a difference of the S-box $S0$ which matches the Type-3 characteristic of the E-function (third column in Table 8) and has minimum Hamming weight, and denote $\delta = i \oplus j$ and $\Delta_1 = S1[i] \oplus S1[j]$. The attack, described in Figure 13, proceeds as follows:

1. We feed differences of 0 in Lines A and B, difference Δ_1 in Line D, and on Line C we feed differences of 0 in Bytes 0,2,3, and difference δ in Byte 1.

2. The difference δ is fed to the S-box $S1$ in the third round. With probability 2^{-8} the output difference is Δ_1, and this cancels with the difference on Line D, leaving a difference of 0. Also, since the source word is rotated by 24 positions to the right, the difference on Line C is now in Byte 2.

3. The difference δ is now fed to the S-box $S0$ in the seventh round. With another probability 2^{-8} the output difference is Δ_0. Line C is rotated again, so the difference is now in Byte 3.

4. Therefore, with probability of 2^{-16}, the output difference on Line A is Δ_0, the difference on Lines B and D is 0 and the difference on Line C is 0 in Bytes 0,1,2, and δ in Byte 3.

The property which enables the above attack is the following: Consider the 32 S-box lookups during this phase, and call an S-box lookup "free" if the value which is affected by this lookup was not used anywhere else thus far. Then, the structure above has free S-box lookups almost until the

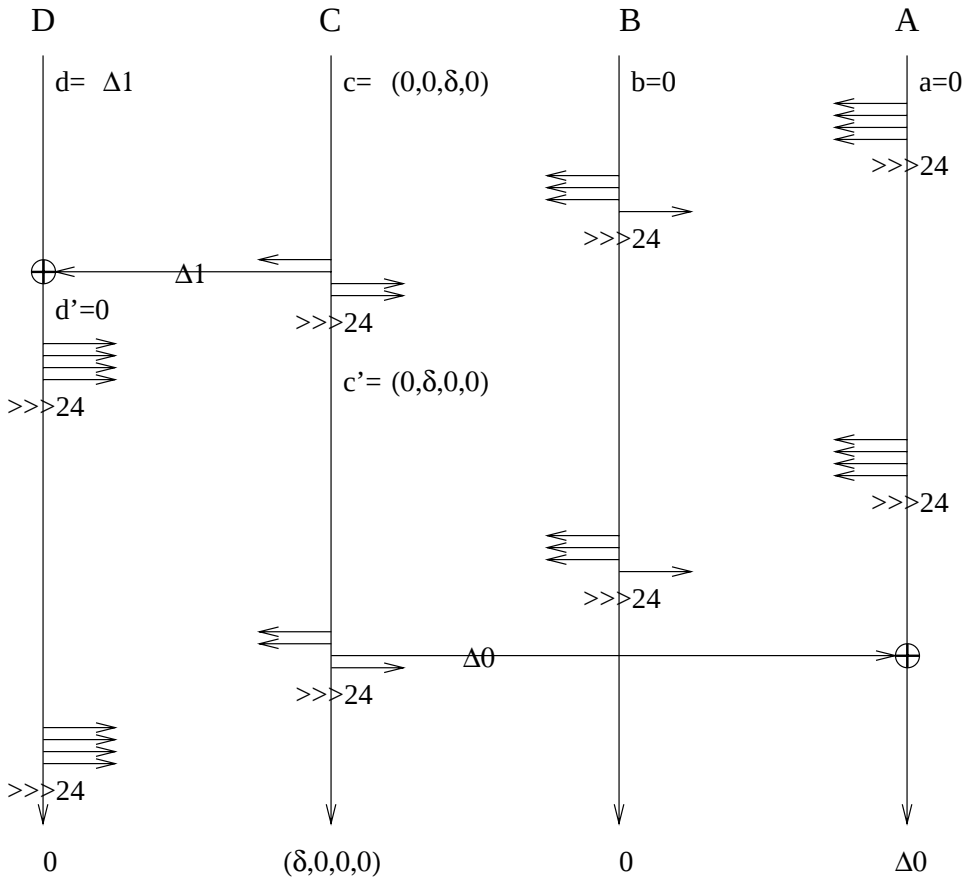

Figure 13: An attack on the mixing phase without the feedback additions.

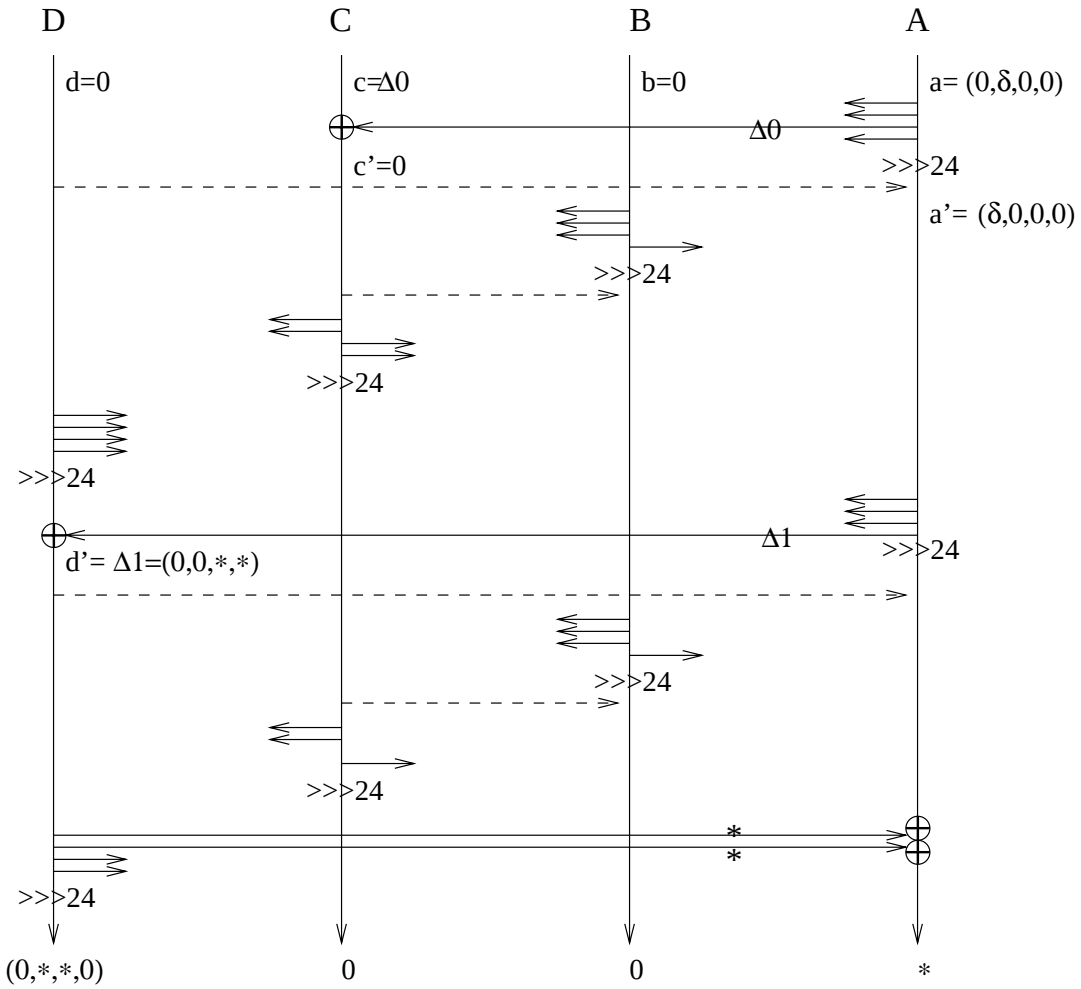

Figure 14: An attack on the mixing phase with a weak S-box.

end of the first super-round (specifically, the 10'th lookup is still free). Hence, one reason for the feedback additions is to eliminate free lookups as early as possible. Indeed, in the actual structure of the mixing phase, the 6'th S-box lookup is the last free lookup.

Avoiding weaknesses in the S-box. Even with the feedback additions, one must be careful to ensure that the mixing phase is not suseptible to attacks due to weaknesses in the S-box. Below we describe one such potential attack. The following attack also explains why we insist that S-boxes $S0$ and $S1$ do not include any differences with more than a single zero byte.

The attack is described in Figure 14. Assume that the S-box $S1$ contains two entries i, j such that $\Delta_1 = S1[i] \oplus S1[j]$ is zero in the two highest bytes, and denote $\delta = i \oplus j$ and $\Delta_0 = S0[i] \oplus S0[j]$. The attack proceeds as follows:

1. Set the input differences on Lines B, D to 0, the difference on Line C to Δ_0 and the difference on Line A to 0 in Bytes 0,1,3 and δ in Byte 2.

2. With probability 2^{-8}, the output difference from the S-box $S0$ in the first round will be Δ_0,

and this will cancel the difference on Line *C*, leaving a difference of 0.

Also, the rotation of Line *A* leaves the difference in Byte 3.

3. With probability 2^{-8}, the output difference from the S-box $S1$ in the fifth round will be Δ_1. This will cause the difference in Line *D* to be Δ_1, which after the feedback addition will also modify Line *A*.

4. In the last round, the difference on Line *D* will affect the first two S-box lookups, so the difference on Line *A* will change again. Also, the rotation of Line *D* will leave the first and last bytes with difference 0.

5. Therefore, with probability 2^{-16} the output differences on Lines *B* and *C* are 0, the output difference on Line *A* is "random" and the output difference on Line *D* is of the form $(0, \star, \star, 0)$. (Notice that in particular, with additional probability 2^{-10}, the differences on both Lines *A* and *D* have the lowest nine bits set to zero.)

The reason that this attack works is that when we have a difference of just two bytes in the S-box, it is possible that these two bytes are used for S-box lookups that affect the same data line. Hence, although the output difference from the S-box is fed back as input difference to the S-boxes, it still only affects a single data line.

Therefore, in the S-box generation process we made sure that any two entries in $S0, S1$ differ in at least three of the four bytes. This way, if the output difference from an S-box is used again as input to the S-boxes, we are guaranteed that at least two data lines are affected.

Expected strength. With the current structure of the mixing phase and the values in the S-box, we are not aware of any characteristic of the mixing phase which holds with probability of more than 2^{-32}. To devise a bound, we note that due to the feedback additions there could be no characteristic which uses less than two S-box lookups. Adding to that the additional effects of the carry bits, we claim a bound of 2^{-20} on the probability of any characteristic of the mixing round. Together with our estimate/bound for the keyed transformation, this gives us an estimate of $2^{240} \cdot 2^{20} \cdot 2^{20} = 2^{280}$ and a bound of $2^{156} \cdot 2^{20} \cdot 2^{20} = 2^{196}$ for the security-level of MARS with respect to differential attacks.

4.3 Other issues

Weak/equivalent keys. As far as we know, MARS does not have any weak keys: the key expansion procedure guarantees that the key words which are used for multiplication do not have any obvious weaknesses (e.g., they are not even), and we are aware of no other source of weak keys. Because of this, we put no restrictions on the key selection.

Also, in all likelihood MARS does not have any equivalent keys: it is highly unlikely that any two different 40-word keys have the same behavior, and the key expansion process is "random enough" so that it is highly unlikely that any two different keys yield the same expanded key array. To see the last point, notice that in the key expansion routine is completely reversible up to (and

including) the reordering of the key words. The only operation which may result in collisions is the "key fixing", where we ignore the lowest two bits in some of the key words. Recall now that the expanded key has about 2^{1248} effective bits, and so the expected number of pairs of original n-bit keys that are mapped to the same expanded array is about

$$\binom{2^n}{2} / 2^{1248} \approx 2^{2n-1249}$$

Therefore, as long as the original key is less than about 600 bits, it is highly unlikely that any pair of keys result in the same expanded array. Similar arguments show that it is just as unlikely that any MARS-key is its own inverse, that two MARS keys are inverses of each other, or that two keys have complementation properties.

No trapdoors. MARS was designed to be free of trapdoors. To help ensure this, we made the design of the S-box according to open principles, and once these principles are set, the choice of S-box was completely deterministic. As far as we know MARS does not have any trapdoors.

Resistance to Visual Cryptanalysis. Recently, Adi Shamir demonstrated that simple photography equipment can be used to considerably speed-up an exhaustive key search of some ciphers [13]. However, he noted that this technique is not efficient against ciphers which rely on data-dependent rotations (or other operations with boolean complexity). Since MARS relies heavily on such operations, it is expected that Visual Cryptanalysis is not very useful against MARS.

Timing attacks and Differential fault analysis. With a proper implementation, MARS should be resilient to timing attacks and differential fault analysis. We note that although in older machines the multiplication time varies widely between different inputs, our key-expansion routine eliminates exactly those keys for which multiplication works much faster (i.e., those with many consecutive 0's or 1's).

Algebraic properties. It is very unlikely that MARS has any algebraic properties. In particular, it is almost surely not a group.

5 Extensions (2.B.6)

5.1 Increasing the block length

Though MARS was designed for a block length of 128 bits, a similar design can be used also for larger blocks. Below we discuss some options for extending the current design to support block length of 256 bits.

Applying generic constructions. There are a few generic constructions which can be used to increase the block length of any cipher. For example, one may use the construction of Naor-Reingold [11], in which a 256-bit block is processed by two invocations of the 128-bit cipher in ECB mode, wrapped between two layers of invertible universal hashing. We estimate that applying this technique to MARS, using fast methods for universal hashing (such as the ones in [6]) yields a cipher with block length of 256 bits that runs in about half the speed of MARS.

Increasing the number of rounds. One way to alter the current design in order to support a block length of 256 bits, is by keeping the same round functions and increasing the number of rounds. Namely, instead of working with four 32-bit words we may work with eight 32-bit words, but still use the same round functions (with one source word and three target words), where in round i we use $D[i]$ as the source word and $D[i+1], D[i+2], D[i+3]$ as the target words (where the index arithmetic is done modulo eight). This, of course, forces us to at least double the number of rounds. More analysis needs to be done to determine if doubling the number of rounds is enough to get a sufficient level of security.

Working with 64-bit words. An alternative approach to increasing the block length is to increase the word length. Namely, instead of working with four 32-bit words, we may work with four 64-bit words. This change would have almost no effect on the design of the E-function and the keyed transformation phase, except that we would have to adjust the fixed rotation amounts. However, it would require a re-design of the mixing phases, since each word now has 8 bytes rather than 4, so we need many more S-box lookups.

5.2 Modes of operation

Block ciphers are routinely used as "building blocks" in the design of other cryptographic algorithms, including collision-resistant hash functions, pseudo-random number generators, stream ciphers, and message authentication codes (MACs). There are standard ways of adapting a block cipher for these applications, and MARS can be used in any of these ways. The underlying security of these constructed modes rests on the assumption that the block cipher simulates the behavior of a random permutation. The good cryptographic properties of MARS ensure that such constructions are strong when the underlying block cipher is instantiated with MARS. Below we briefly review some of these constructions.

Collision resistant hash functions. In the following we assume a 128-bit block and 128-bit key cipher E like MARS. We denote the ciphertext block resulting from encrypting x with key k by $E_k(x)$. Before processing, an input string x is always padded as necessary to make its length a multiple of 128 (for example a 1 and then extra 0's can be added to x). In the following we assume that the length of x is a multiple of 128, i.e. $x = x_1 \ldots x_\ell$ where each x_i is a 128-bit block and that we use some fixed 128-bit block as an initial value IV. Some known constructions of hash functions include:

MATYAS-MEYER-OSEAS HASH. Define recursively $H_0 = IV$ and $H_i = E_{H_{i-1}}(x_i) \oplus x_i$. The hash of x is defined as $H(x) = H_\ell$.

DAVIES-MEYER HASH. Define recursively $H_0 = IV$ and $H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$. The hash of x is defined as $H(x) = H_\ell$.

MIYAGUCHI-PRENEEL HASH. Define recursively $H_0 = IV$ and $H_i = E_{H_{i-1}}(x_i) \oplus x_i \oplus H_{i-1}$. The hash of x is defined as $H(x) = H_\ell$.

When instantiated with a secure block-cipher such as MARS, the above constructions result in secure 128-bit collision resistant hash functions (meaning that the best strategy to find a collision, i.e. two string $x \neq y$ such that $H(x) = H(y)$ would involve $\approx 2^{64}$ operations). If a longer hash code is desired then one could use the ISO/IEC Standard 10118-2 which results in a hash code whose length is two ciphertext blocks (e.g. 256 bits in the case of MARS).

In general, block-cipher based hash functions are not as efficient as customized ones (e.g. the MD*X*-family). There are several reasons for this, one of them being the need to re-key the algorithm E at each stage. This is true also when E is instantiated with MARS. However for MARS the penalty is limited as the cost of re-keying is bounded by 10 times the cost of encrypting a block.

Pseudo-random number generation. Pseudo-random number generators (PRNGs) are created out of a block cipher E by running it in various specific modes of operation. In each such mode, r is a parameter $1 \leq r \leq 128$ that specifies how many bits should be taken as pseudo-random from any specific iteration of the block cipher E. Given a 128-bit word w we denote by $w|_r$ the leftmost r-bits of w.

COUNTER MODE. In this mode the seed is the key k and the sequence $a_1, a_2, \ldots$ of r-bit numbers is generated as $a_i = E_k(i)|_r$.

OUTPUT FEEDBACK MODE. In this mode the seed is the key k and the sequence $a_1, a_2, \ldots$ of r-bit numbers is generated as $a_i = E_k^i(IV)|_r$ where IV is a fixed constant (or can be part of the seed), and $E_k^i(IV)$ denotes the 128-bit word obtained by successively encrypting IV i times.

Other methods have been proposed in the literature to generate strong PRNGs from a block cipher (for example see the method cited in the ANSI Standard X9.31-1998).

Stream Ciphers. A cryptographically strong PRNG automatically yields a good stream cipher: if a_i is the i^{th} bit output by the PRNG then one can use it to mask the i^{th} bit m_i of the message stream by transmitting $c_i = m_i \oplus a_i$. Thus the above two modes also constitute good implementations of stream ciphers.

CIPHER FEEDBACK MODE. This mode of operation for a block cipher can be used to implement a stream cipher (but it's not a PRNG since it uses previous bits derived from the input stream to modify the subsequent mask bits.) The CFB mode works as follows: define initially $x_0 = IV$, $w_0 = E_k(x_0)|_r$ and $c_0 = m_0 \oplus w_0$ where m_0 are the first r bits in the input stream. Then define recursively $x_i = (x_{i-1}|c_{i-1})_{127..0}$ (that is, the lowest 128 bits in the concatenation of x_{i-1} and c_{i-1}) and $w_i = E_k(x_i)|_r$ and $c_i = m_i \oplus w_i$, where m_i is the i^{th} r-bit block in the input stream. The encrypted stream is $c_0, c_1, \ldots$

Message authentication codes (MACs). Block ciphers are used in two basic ways to generate MACs.

CBC-MAC. In this method, the message m to be tagged is encrypted in CBC mode using the block cipher E, but the tag consists only of half of the bits of the last encrypted block. That is, if $m = m_1 \ldots m_\ell$ where m_i's are 128-bit blocks, and $c = c_1 \ldots c_\ell$ is the CBC encryption of m using E with key k, then $MAC_k(m) = c_\ell|_{64}$.

CARTER-WEGMAN MACS. In this method the key k is split in two parts k_1, k_2. The first subkey k_i is used to pick a hash function H_{k_1} from a universal family and the second is used as a key for the encryption. (A family of hash functions is universal if the probability of getting a collision for any two specific pre-images when picking a random function from the family is small.) The message m is first hashed down by computing $h = H_{k_1}(m)$. The value h should be of the same length as the block used by the cipher. Then the tag is computed as $c = E_{k_2}(h)$. Alternatively, the tag for the i'th message can be computed as $c = E_{k_2}(i) \oplus h$.

References

[1] C.M. Adams. "Constructing symmetric ciphers using the CAST design procedure". Designs, Codes and Cryptography, 12(3):283-316, November 1997.

[2] W. Becker, "Method And System For Machine Enciphering And Deciphering" U.S. Patent No. 4157454, 1979.

[3] E. Biham and A. Shamir, "Differential cryptanalysis of the data encryption standard", Springer-Verlag, 1993.

[4] FIPS 46, "Data encryption standard", Federal Information Processing Standards Publication 46, U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service, Springfield, Virginia, 1977 (revised as FIPS 46-1:1988; FIPS 46-2:1993).

[5] A. Biryukov and E. Kushilevitz, "Improved cryptanalysis of RC5", *Advances in Cryptology, EUROCRYPT 98, Lecture Notes in Computer Science, vol. 1403,* K. Nyberg ed., Springer-Verlag, pages 85–99, 1998.

[6] S. Halevi and H. Krawczyk, "MMH: software message authentication in the Gbit/sec rates", Proceedings of 4th FSE workshop, Lecture Notes in Computer Science, vol. 1267, Springer-Verlag, 1997, pages 172-189.

[7] X. Lai, J.Massey and S. Murphy, "Markov ciphers and differential cryptanalysis", Proceedings of Eurocrypt '91, pages 17-38.

[8] W.E. Madryga, "A high performance encryption algorithm", Computer security: a global challenge, Elsevier Science Publishers, 1984, pages 557-570.

[9] M. Matsui. "Linear cryptanalysis method for DES cipher". *Advances in Cryptology, EUROCRYPT 93, Lecture Notes in Computer Science, vol. 765*, T. Helleseth ed., Springer-Verlag, pages 386–397, 1994.

[10] S. Moriai, K. Aoki, and K. Ohta. Key-dependency of linear probability of RC5. *IEICE Trans. Fundamentals*, E80-A(1):9–18, 1997.

[11] M. Naor and O. Reingold, "On the construction of pseudo-random permutations: Luby-Rackoff Revisited", Proceedings of the 29'th ACM Symposium on Theory of Computing, 1997, pages 189-199.

[12] R. Rivest, "The RC5 Encryption Algorithm" K.U. Leuven workshop on cryptographic algorithms, Springer-Verlag, 1995.

[13] A. Shamir, "Visual Cryptanalysis", *Advances in Cryptology, EUROCRYPT 98, Lecture Notes in Computer Science, vol. 1403,* K. Nyberg ed., Springer-Verlag, pages 201–210, 1998.

A S-box

Below is the S-box we use in the cipher.

```
WORD Sbox[ ] = {
  0x09d0c479, 0x28c8ffe0, 0x84aa6c39, 0x9dad7287, 0x7dff9be3, 0xd4268361,
  0xc96da1d4, 0x7974cc93, 0x85d0582e, 0x2a4b5705, 0x1ca16a62, 0xc3bd279d,
  0x0f1f25e5, 0x5160372f, 0xc695c1fb, 0x4d7ff1e4, 0xae5f6bf4, 0x0d72ee46,
  0xff23de8a, 0xb1cf8e83, 0xf14902e2, 0x3e981e42, 0x8bf53eb6, 0x7f4bf8ac,
  0x83631f83, 0x25970205, 0x76afe784, 0x3a7931d4, 0x4f846450, 0x5c64c3f6,
  0x210a5f18, 0xc6986a26, 0x28f4e826, 0x3a60a81c, 0xd340a664, 0x7ea820c4,
  0x526687c5, 0x7eddd12b, 0x32a11d1d, 0x9c9ef086, 0x80f6e831, 0xab6f04ad,
  0x56fb9b53, 0x8b2e095c, 0xb68556ae, 0xd2250b0d, 0x294a7721, 0xe21fb253,
  0xae136749, 0xe82aae86, 0x93365104, 0x99404a66, 0x78a784dc, 0xb69ba84b,
  0x04046793, 0x23db5c1e, 0x46cae1d6, 0x2fe28134, 0x5a223942, 0x1863cd5b,
  0xc190c6e3, 0x07dfb846, 0x6eb88816, 0x2d0dcc4a, 0xa4ccae59, 0x3798670d,
  0xcbfa9493, 0x4f481d45, 0xeafc8ca8, 0xdb1129d6, 0xb0449e20, 0x0f5407fb,
  0x6167d9a8, 0xd1f45763, 0x4daa96c3, 0x3bec5958, 0xababa014, 0xb6ccd201,
  0x38d6279f, 0x02682215, 0x8f376cd5, 0x092c237e, 0xbfc56593, 0x32889d2c,
  0x854b3e95, 0x05bb9b43, 0x7dcd5dcd, 0xa02e926c, 0xfae527e5, 0x36a1c330,
  0x3412e1ae, 0xf257f462, 0x3c4f1d71, 0x30a2e809, 0x68e5f551, 0x9c61ba44,
  0x5ded0ab8, 0x75ce09c8, 0x9654f93e, 0x698c0cca, 0x243cb3e4, 0x2b062b97,
  0x0f3b8d9e, 0x00e050df, 0xfc5d6166, 0xe35f9288, 0xc079550d, 0x0591aee8,
  0x8e531e74, 0x75fe3578, 0x2f6d829a, 0xf60b21ae, 0x95e8eb8d, 0x6699486b,
  0x901d7d9b, 0xfd6d6e31, 0x1090acef, 0xe0670dd8, 0xdab2e692, 0xcd6d4365,
  0xe5393514, 0x3af345f0, 0x6241fc4d, 0x460da3a3, 0x7bcf3729, 0x8bf1d1e0,
  0x14aac070, 0x1587ed55, 0x3afd7d3e, 0xd2f29e01, 0x29a9d1f6, 0xefb10c53,
  0xcf3b870f, 0xb414935c, 0x664465ed, 0x024acac7, 0x59a744c1, 0x1d2936a7,
  0xdc580aa6, 0xcf574ca8, 0x040a7a10, 0x6cd81807, 0x8a98be4c, 0xaccea063,
```

```
0xc33e92b5, 0xd1e0e03d, 0xb322517e, 0x2092bd13, 0x386b2c4a, 0x52e8dd58,
0x58656dfb, 0x50820371, 0x41811896, 0xe337ef7e, 0xd39fb119, 0xc97f0df6,
0x68fea01b, 0xa150a6e5, 0x55258962, 0xeb6ff41b, 0xd7c9cd7a, 0xa619cd9e,
0xbcf09576, 0x2672c073, 0xf003fb3c, 0x4ab7a50b, 0x1484126a, 0x487ba9b1,
0xa64fc9c6, 0xf6957d49, 0x38b06a75, 0xdd805fcd, 0x63d094cf, 0xf51c999e,
0x1aa4d343, 0xb8495294, 0xce9f8e99, 0xbffcd770, 0xc7c275cc, 0x378453a7,
0x7b21be33, 0x397f41bd, 0x4e94d131, 0x92cc1f98, 0x5915ea51, 0x99f861b7,
0xc9980a88, 0x1d74fd5f, 0xb0a495f8, 0x614deed0, 0xb5778eea, 0x5941792d,
0xfa90c1f8, 0x33f824b4, 0xc4965372, 0x3ff6d550, 0x4ca5fec0, 0x8630e964,
0x5b3fbbd6, 0x7da26a48, 0xb203231a, 0x04297514, 0x2d639306, 0x2eb13149,
0x16a45272, 0x532459a0, 0x8e5f4872, 0xf966c7d9, 0x07128dc0, 0x0d44db62,
0xafc8d52d, 0x06316131, 0xd838e7ce, 0x1bc41d00, 0x3a2e8c0f, 0xea83837e,
0xb984737d, 0x13ba4891, 0xc4f8b949, 0xa6d6acb3, 0xa215cdce, 0x8359838b,
0x6bd1aa31, 0xf579dd52, 0x21b93f93, 0xf5176781, 0x187dfdde, 0xe94aeb76,
0x2b38fd54, 0x431de1da, 0xab394825, 0x9ad3048f, 0xdfea32aa, 0x659473e3,
0x623f7863, 0xf3346c59, 0xab3ab685, 0x3346a90b, 0x6b56443e, 0xc6de01f8,
0x8d421fc0, 0x9b0ed10c, 0x88f1a1e9, 0x54c1f029, 0x7dead57b, 0x8d7ba426,
0x4cf5178a, 0x551a7cca, 0x1a9a5f08, 0xfcd651b9, 0x25605182, 0xe11fc6c3,
0xb6fd9676, 0x337b3027, 0xb7c8eb14, 0x9e5fd030,
0x6b57e354, 0xad913cf7, 0x7e16688d, 0x58872a69, 0x2c2fc7df, 0xe389ccc6,
0x30738df1, 0x0824a734, 0xe1797a8b, 0xa4a8d57b, 0x5b5d193b, 0xc8a8309b,
0x73f9a978, 0x73398d32, 0x0f59573e, 0xe9df2b03, 0xe8a5b6c8, 0x848d0704,
0x98df93c2, 0x720a1dc3, 0x684f259a, 0x943ba848, 0xa6370152, 0x863b5ea3,
0xd17b978b, 0x6d9b58ef, 0x0a700dd4, 0xa73d36bf, 0x8e6a0829, 0x8695bc14,
0xe35b3447, 0x933ac568, 0x8894b022, 0x2f511c27, 0xddfbcc3c, 0x006662b6,
0x117c83fe, 0x4e12b414, 0xc2bca766, 0x3a2fec10, 0xf4562420, 0x55792e2a,
0x46f5d857, 0xceda25ce, 0xc3601d3b, 0x6c00ab46, 0xefac9c28, 0xb3c35047,
0x611dfee3, 0x257c3207, 0xfdd58482, 0x3b14d84f, 0x23becb64, 0xa075f3a3,
0x088f8ead, 0x07adf158, 0x7796943c, 0xfacabf3d, 0xc09730cd, 0xf7679969,
0xda44e9ed, 0x2c854c12, 0x35935fa3, 0x2f057d9f, 0x690624f8, 0x1cb0bafd,
0x7b0dbdc6, 0x810f23bb, 0xfa929a1a, 0x6d969a17, 0x6742979b, 0x74ac7d05,
0x010e65c4, 0x86a3d963, 0xf907b5a0, 0xd0042bd3, 0x158d7d03, 0x287a8255,
0xbba8366f, 0x096edc33, 0x21916a7b, 0x77b56b86, 0x951622f9, 0xa6c5e650,
0x8cea17d1, 0xcd8c62bc, 0xa3d63433, 0x358a68fd, 0x0f9b9d3c, 0xd6aa295b,
0xfe33384a, 0xc000738e, 0xcd67eb2f, 0xe2eb6dc2, 0x97338b02, 0x06c9f246,
0x419cf1ad, 0x2b83c045, 0x3723f18a, 0xcb5b3089, 0x160bead7, 0x5d494656,
0x35f8a74b, 0x1e4e6c9e, 0x000399bd, 0x67466880, 0xb4174831, 0xacf423b2,
0xca815ab3, 0x5a6395e7, 0x302a67c5, 0x8bdb446b, 0x108f8fa4, 0x10223eda,
0x92b8b48b, 0x7f38d0ee, 0xab2701d4, 0x0262d415, 0xaf224a30, 0xb3d88aba,
0xf8b2c3af, 0xdaf7ef70, 0xcc97d3b7, 0xe9614b6c, 0x2baebff4, 0x70f687cf,
0x386c9156, 0xce092ee5, 0x01e87da6, 0x6ce91e6a, 0xbb7bcc84, 0xc7922c20,
0x9d3b71fd, 0x060e41c6, 0xd7590f15, 0x4e03bb47, 0x183c198e, 0x63eeb240,
0x2ddbf49a, 0x6d5cba54, 0x923750af, 0xf9e14236, 0x7838162b, 0x59726c72,
0x81b66760, 0xbb2926c1, 0x48a0ce0d, 0xa6c0496d, 0xad43507b, 0x718d496a,
0x9df057af, 0x44b1bde6, 0x054356dc, 0xde7ced35, 0xd51a138b, 0x62088cc9,
0x35830311, 0xc96efca2, 0x686f86ec, 0x8e77cb68, 0x63e1d6b8, 0xc80f9778,
0x79c491fd, 0x1b4c67f2, 0x72698d7d, 0x5e368c31, 0xf7d95e2e, 0xa1d3493f,
```

```
  0xdcd9433e, 0x896f1552, 0x4bc4ca7a, 0xa6d1baf4, 0xa5a96dcc, 0x0bef8b46,
  0xa169fda7, 0x74df40b7, 0x4e208804, 0x9a756607, 0x038e87c8, 0x20211e44,
  0x8b7ad4bf, 0xc6403f35, 0x1848e36d, 0x80bdb038, 0x1e62891c, 0x643d2107,
  0xbf04d6f8, 0x21092c8c, 0xf644f389, 0x0778404e, 0x7b78adb8, 0xa2c52d53,
  0x42157abe, 0xa2253e2e, 0x7bf3f4ae, 0x80f594f9, 0x953194e7, 0x77eb92ed,
  0xb3816930, 0xda8d9336, 0xbf447469, 0xf26d9483, 0xee6faed5, 0x71371235,
  0xde425f73, 0xb4e59f43, 0x7dbe2d4e, 0x2d37b185, 0x49dc9a63, 0x98c39d98,
  0x1301c9a2, 0x389b1bbf, 0x0c18588d, 0xa421c1ba, 0x7aa3865c, 0x71e08558,
  0x3c5cfcaa, 0x7d239ca4, 0x0297d9dd, 0xd7dc2830, 0x4b37802b, 0x7428ab54,
  0xaeee0347, 0x4b3fbb85, 0x692f2f08, 0x134e578e, 0x36d9e0bf, 0xae8b5fcf,
  0xedb93ecf, 0x2b27248e, 0x170eb1ef, 0x7dc57fd6, 0x1e760f16, 0xb1136601,
  0x864e1b9b, 0xd7ea7319, 0x3ab871bd, 0xcfa4d76f, 0xe31bd782, 0x0dbeb469,
  0xabb96061, 0x5370f85d, 0xffb07e37, 0xda30d0fb, 0xebc977b6, 0x0b98b40f,
  0x3a4d0fe6, 0xdf4fc26b, 0x159cf22a, 0xc298d6e2, 0x2b78ef6a, 0x61a94ac0,
  0xab561187, 0x14eea0f0, 0xdf0d4164, 0x19af70ee
};
```

B Pseudo-code for decryption

MARS-decrypt(input: $D[\,], K[\,]$)

Phase (I): Forward mixing

1. // First add subkeys to data
2. for $i = 0$ to 3 do
3. $\quad D[i] = D[i] + K[36+i]$
4. // Then do eight rounds of forward mixing
5. for $i = 7$ down to 0 do
6. $\quad$ // rotate $D[\,]$ by one word to the left for this round
7. $\quad (D[3], D[2], D[1], D[0]) \leftarrow (D[2], D[1], D[0], D[3])$
8. $\quad$ // and rotate of the source word to the right
9. $\quad D[0] = D[0] \gg 24$
10. $\quad$ // four S-box look-ups
11. $\quad D[3] = D[3] \oplus S0[$ 2nd byte of $D[0]\,]$
12. $\quad D[3] = D[3] + S1[$ 3rd byte of $D[0]\,]$
13. $\quad D[2] = D[2] + S0[$ high byte of $D[0]\,]$
14. $\quad D[1] = D[1] \oplus S1[$ low byte of $D[0]\,]$
15. $\quad$ // followed by additional mixing operations
16. $\quad$ if $i = 2$ or 6 then
17. $\quad\quad D[0] = D[0] + D[3]$ // add $D[3]$ back to the source word
18. $\quad$ if $i = 3$ or 7 then
29. $\quad\quad D[0] = D[0] + D[1]$ // add $D[1]$ back to the source word
20. end-for

Phase (II): Keyed transformation

21. // Do 16 rounds of keyed transformation
22. for $i = 15$ down to 0 do
23. // rotate $D[\,]$ by one word to the left for this round
24. $(D[3], D[2], D[1], D[0]) \leftarrow (D[2], D[1], D[0], D[3])$
25. $D[0] = D[0] \gg 13$
26. $(out1, out2, out3) = \text{E-function}(D[0], K[2i+4], K[2i+5])$
27. $D[2] = D[2] - out2$
28. if $i < 8$ then // last 8 rounds in forward mode
29. $D[1] = D[1] - out1$
30. $D[3] = D[3] \oplus out3$
31. else // first 8 rounds in backwards mode
32. $D[3] = D[3] - out1$
33. $D[1] = D[1] \oplus out3$
34. end-if
35. end-for

Phase (III): Backwards mixing

36. // Do eight rounds of backwards mixing
37. for $i = 7$ down to 0 do
38. // rotate $D[\,]$ by one word to the left for this round
39. $(D[3], D[2], D[1], D[0]) \leftarrow (D[2], D[1], D[0], D[3])$
40. // additional mixing operations
41. if $i = 0$ or 4 then
42. $D[0] = D[0] - D[3]$ // subtract $D[3]$ from source word
43. if $i = 1$ or 5 then
44. $D[0] = D[0] - D[1]$ // subtract $D[1]$ from source word
45. // and rotation of the source word to the left
46. $D[0] = D[0] \ll 24$
47. // four S-box look-ups
48. $D[3] = D[3] \oplus S1[$ high byte of $D[0]\,]$
49. $D[2] = D[2] - S0[$ 3rd byte of $D[0]\,]$
50. $D[1] = D[1] - S1[$ 2nd byte of $D[0]\,]$
51. $D[1] = D[1] \oplus S0[$ low byte of $D[0]\,]$
52. end-for
53. // Then subtract subkeys from data
54. for $i = 0$ to 3 do
55. $D[i] = D[i] - K[i]$