The MARS Encryption Algorithm

Carolynn Burwick [c] , Don Coppersmith [a] , Edward D'Avignon [c] , Rosario Gennaro [a] , Shai Halevi [a] , Charanjit Jutla [a] , Stephen M. Matyas [c] , Luke O'Connor [d] , Mohammad Peyravian [b] , David Safford [a] , Nevenko Zunic [c]

[a] IBM T. J. Watson Research, Yorktown Heights, NY 10598, USA

[b] IBM Corporation, Research Triangle Park, NC 27709, USA

[c] IBM Corporation, Poughkeepsie, NY 12601, USA

[d] IBM Zurich Research, Rueschlikon, Switzerland

August 27, 1999

ABSTRACT

This paper describes and analyzes the MARS symmetric-key encryption algorithm which is a new block cipher submitted to NIST for consideration as the Advanced Encryption Standard (AES). MARS supports 128-bit blocks and a variable key size. It 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. Specifically, in MARS we use a unique combination of S-box lookups, multiplications and data-dependent rotations. MARS has a heterogeneous structure, with *cryptographic core* rounds that are wrapped by simpler *mixing* rounds. The cryptographic core rounds provide strong resistance to all known cryptanalytical attacks, while the mixing rounds provide good avalanche and offer very wide security margins to thwart new (yet unknown) attacks. Our C implementation of MARS runs at rates of 85 Mbit/sec on a 200 MHz PowerPC, and 65 Mbit/sec on a 200 MHz Pentium-Pro. The cryptographic core runs at 160 Mbit/sec on the PowerPC, and 104 Mbit/sec on the Pentium-Pro. MARS can achieve a 10 times speedup factor in hardware. MARS is also suitable for limited-resource environments such as the smartcard since its code is remarkably compact.

Keywords: analysis, block cipher, encryption, MARS, symmetric key

1 Introduction

Symmetric-key block ciphers have long been used as a fundamental cryptographic element for providing information security. Although they are primarily designed for providing data confidentiality, their versatility allows them to serve as a main component in the construction of many cryptographic systems such as pseudorandom

number generators, message authentication protocols, stream ciphers, and hash functions. There are many symmetric-key block ciphers which offer different levels of security, flexibility, and efficiency. Among the many symmetric-key block ciphers currently available, some (such as DES, RC5, CAST, Blowfish, FEAL, SAFER, and IDEA) have received the greatest practical interest [6-11].

Most symmetric-key block ciphers (such as DES, RC5, CAST, and Blowfish) are based on a "Feistel" network construct and a "round function". A Feistel cipher involves dividing the plaintext into two halves and repeatedly applying a round function to the data for some number of rounds, where in each round using the round function and a key, the left half is transformed based on the right half and then the right half is transformed based on the modified left half. The round function provides a basic encryption mechanism by composing several simple linear and non-linear operations such as exclusive-or, substitution, permutation, and modular arithmetic [4,5].

Different round functions provide different levels of security, efficiency, and flexibility. The strength of a Feistel cipher depends heavily on the degree of diffusion and non-linearity properties provided by the round function. Many ciphers (such as DES and CAST) base their round functions on a construct called a "substitution box" (s-box) as a source of diffusion and non-linearity. Some ciphers (such as RC5) use bit-wise data-dependent rotations and a few other ciphers (such as IDEA) use multiplication in their round functions for diffusion.

In this paper, we present a novel symmetric-key block cipher, called MARS, with a block size of 128 bits and a variable key size, ranging from 128 to 448 bits. The MARS cipher uses a variety of operations to provide a combination of high security, high speed, and implementation flexibility. The main theme behind the design of MARS is to get the best security/performance tradeoff by utilizing the strongest techniques available today for designing block ciphers.

The rest of this paper is organized as follows. In Section 2, we present MARS which includes the description of encryption/decryption operation and key setup. The performance and implementation of MARS in software and hardware are discussed in Section 3. The rationale behind the design of MARS is presented in Section 4. A cryptanalysis of MARS and its resistance against linear and differential attacks is discussed in Section 5. Section 6 provides some concluding remarks.

2 Description of MARS

MARS takes as input four 32-bit plaintext data words A, B, C, D and produces four 32-bit ciphertext data words A', B', C', D'. The cipher is word-oriented, in that all the internal operations are performed on 32-bit words. MARS is a type-3 Feistel network, divided into three phases: a 16-round "cryptographic core" phase wrapped with two layers of 8-round "forward" and "backwards mixing" (Figure 1). The cryptographic core rounds provide strong resistance to all known cryptanalytical attacks, while the mixing rounds provide good avalanche and offer very wide security margins to thwart new (yet unknown) attacks.

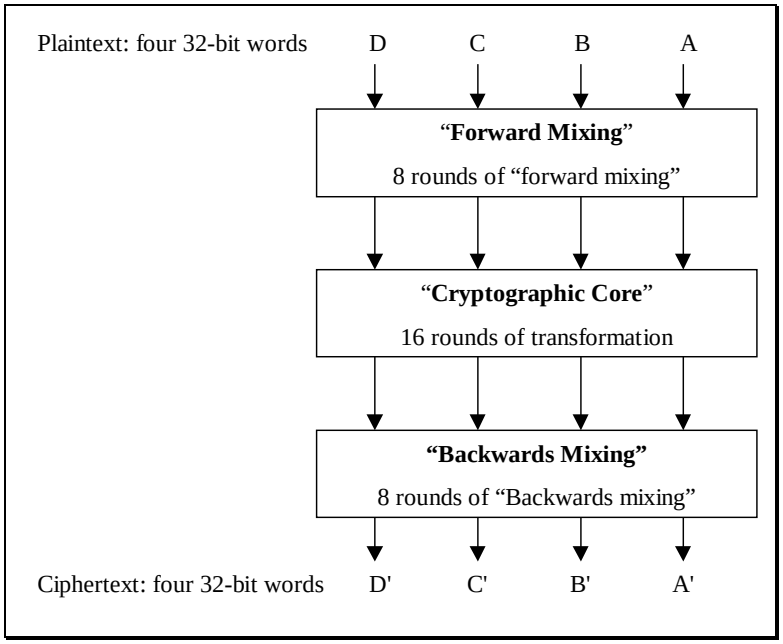

Figure 1: High-level structure of MARS encryption procedure

MARS accepts a variable size user-supplied key ranging from 4 to 14 words (i.e., 128 to 448 bits). MARS uses a key expansion procedure to "expand" the user-supplied key (consisting of *n* 32-bit words, where *n* is any number between 4 and 14) into a key array K[ ] of 40 words for the encryption/decryption operation.

The MARS cipher uses a variety of operations to provide a combination of high security, high speed, and implementation flexibility. Specifically, it combines exclusive-or (xor), addition, subtractions, multiplications, and both fixed and data-dependent rotations. MARS also uses a single (S-box) table of 512 32-bit words to provide good resistance against linear and differential attacks, as well as good avalanche of data and key bits. This S-box is also used by the key expansion procedure. Sometimes the S-box is viewed as two tables, each of 256 entries, denoted by S_0 and S_1. In the design of the S-box, we generated the entries in a "pseudo-random fashion" and tested that the resulting S-box has good differential and linear properties. The MARS S-box is shown in an appendix at the end of paper.

The pseudo-code in Figure 2 shows the encryption operation of MARS in detail. The operations used in the cipher are applied to 32-bit words, which are viewed as unsigned integers. In this pseudo-code we use the following notations. 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 c⊕d a bitwise exclusive-or of the two words c and d. We denote by c+d addition modulo 2^{32}, by c-d subtraction modulo 2^{32}, and by c×d multiplication modulo 2^{32}. Also, c<<<d and c>>>d, denote cyclic rotations of the 32-bit word c by d positions to the left and right, respectively.

The decryption operation of MARS is the inverse of the encryption operation and the code for decryption is similar

```
// Forward Mixing
(A,B,C,D) = (A,B,C,D) + (K[0],K[1],K[2],K[3])
For i = 0 to 7 do {
        B = (B ⊕ S0[A]) + S1[A>>>8]
        C = C + S0[A>>>16]
        D = D ⊕ S1[A>>>24]
        A = (A>>>24) + B(if i=1,5) + D(if i=0,4)
        (A,B,C,D) = (B,C,D,A)
}
// Cryptographic Core
For i = 0 to 15 do {
        R = ((A<<<13) × K[2i+5]) <<< 10
        M = (A + K[2i+4]) <<< (low 5 bits of (R>>>5))
        L = (S[M] ⊕ (R>>>5) ⊕ R) <<< (low 5 bits of R)
        B = B +L(if i<8) ⊕ R(if i≥8)
        C = C + M
        D = D ⊕ R(if i<8) + L(if i≥8)
        (A,B,C,D) = (B,C,D,A<<<13)
}
// Backwards Mixing
For i = 0 to 7 do {
        A = A - B(if i=3,7) - D(if i=2,6)
        B = B ⊕ S1[A]
        C = C - S0[A<<<8]
        D = (D - S1[A<<<16]) ⊕ S0[A<<<24]
        (A,B,C,D) = (B,C,D,A<<<24)
}
(A,B,C,D) = (A,B,C,D) - (K[36],K[37],K[38],K[39])

NOTE: S0[X] and S1[X] use low 8 bits of X.  S[X] uses low 9 bits of X.
S is the concatenation of S0 and S1.
```

Figure 2: MARS encryption pseudocode

to the code for encryption.

2.1 Description of MARS Key Expansion

The MARS key expansion procedure expands the user-supplied key ranging from 4 to 14 words into a 40-word key

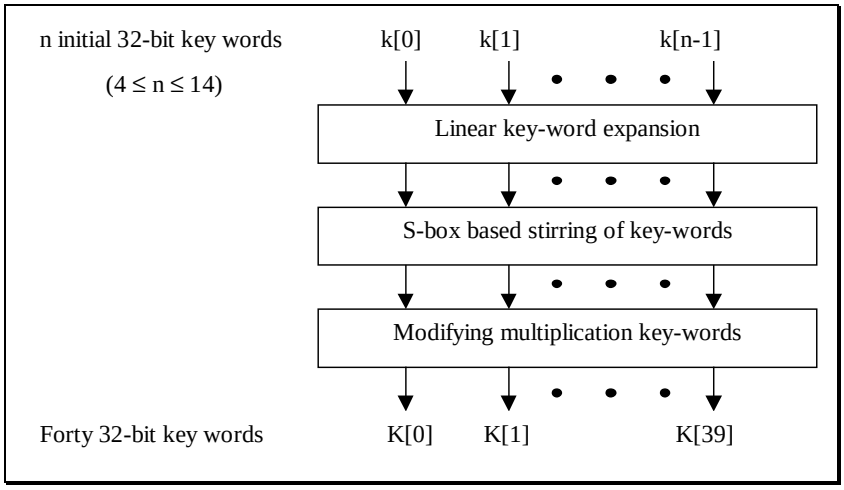

Figure 3: Key expansion procedure of MARS

```
// Initialize T[ ] With the Original Key Data k[ ]
T[0 … n-1] = k[0 … n-1],    T[n] = n,    T[n+1 … 14] = 0
For j = 0, 1, … ,3 do {
        // Linear Key-Word Expansion
        For i = 0, 1, … ,14 do { T[i] = T[i] ⊕ ((T[i-7 mod 15] ⊕ T[i-2 mod 15]) <<< 3) ⊕ (4i+j) }
        // S-box Based Stirring of Key-Words
        Repeat 4 times { For i = 0, 1, … ,14 do { T[i] = (T[i] + S[low 9 bits of T[i-1 mod 15]]) <<< 9 } }
        // Store Next 10 Key-Words into K[ ]
        For i = 0, 1, … ,9 do { K[10j+i] = T[4i mod 15] }
}
// Modifying Multiplication Key-Words
B[ ] = {0xa4a8d57b; 0x5b5d193b; 0xc8a8309b; 0x73f9a978}
For i = 5, 7, … ,35 do {
        j = least two bits of K[i]
        w = K[i] with both of the lowest two bits set to 1
        Compute a word mask M as follows:
                M = 0
                M_n = 1 iff w_n belongs to a sequence of 10 consecutive 0's or 1's in w,
                and also 2 ≤ n ≤ 30 and w_{n-1} = w_n = w_{n+1}
        r = least five bits of K[i-1]
        p = B[j] <<< r
        K[i] = w ⊕ (p ∧ M)
}

NOTE: x_i denotes the i-th bit in the 32-bit word x.
```

Figure 4: MARS key expansion pseudocode

for use in the encryption/decryption operation. The key expansion procedure consists of three steps (Figure 3). The first step is "linear expansion" which expands the original user-supplied key to forty 32-bit words using a simple linear transformation. The second step is "S-box based key stirring" which stirs the expanded key using seven rounds of a type-1 Feistel network to destroy linear relations in the key. Then a "multiplication key-word modifying" step examines the key words which are used in the MARS encryption/decryption operation for multiplication and modifies them if needed. The pseudo-code in Figure 4 shows the key expansion operation of MARS in detail. In the pseudo-code c∧d denotes bitwise-and of the two words c and d.

3 MARS Performance

Here we provide some performance measurements for MARS encryption/decryption operations as well as the key expansion procedure. The performance of MARS is independent of the key length used due to the structure of the key expansion procedure. Also, the performance measurements for the encryption and decryption operations are essentially the same since MARS is a symmetric-key cipher, its decryption process is the inverse of the encryption process, and the decryption code is similar to the encryption code.

MARS can achieve very high performance in software since it is designed to take full advantage of the powerful operations available in today's computers. We currently have a C implementation running at 85 Mbit/sec on PowerPC with a clock rate of 200MHz. On an IBM-compatible PC with a 200MHz Pentium-Pro processor, MARS

runs at 65 Mbit/sec. The speed of the MARS "cryptographic core" by itself on these machines is 160 Mbit/sec and 104 Mbit/sec, respectively. MARS is significantly faster than DES and Triple-DES. Current optimized C implementation of DES runs at 16.7 Mbit/sec on an IBM-compatible PC with a 200MHz Pentium-Pro processor. On the same machine, Triple-DES runs at 7.3 Mbit/sec.

The performance of the MARS key expansion operation is independent of the key length used due to the structure of the key expansion procedure. Our C implementation of the key expansion procedure sets up 121,000 keys/sec on a PowerPC with a clock rate of 200MHz. On an IBM-compatible PC with a 200MHz Pentium-Pro processor, it sets up 52,000 keys/sec.

MARS can also be implemented efficiently in hardware. We estimate that a hardware implementation of MARS with a single multiplier takes about 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-box, adders, and the multiplier. This cell count easily fits on all chips, including smartcards. As a basis for comparison, a typical DES implementation is approximately 28,000 cells. With this implementation, the forward and backwards mixing phases can be completed within 9 cycles, each. The cryptographic core phase takes 32 cycles. In total, we estimate that an encryption/decryption of one block takes 50 clock cycles. This implies a performance estimate of 640 Mbit/sec for encryption and decryption. 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 core 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 core 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 throughput of 8 Gbit/sec.

4 MARS Design Rationale

Cryptanalytical techniques typically treat the top and bottom rounds of the cipher differently than the middle rounds. 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 [12], 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 heterogeneous 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, this mixed structure can be thought of as an "insurance policy" to protect the cipher against future advances in cryptanalytical techniques.

4.1 Cryptographic Core

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 takes as input one data word and returns three data words as output. The structure of the Feistel network is depicted in Figure 5, and the E-function itself is shown in Figure 6. 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.

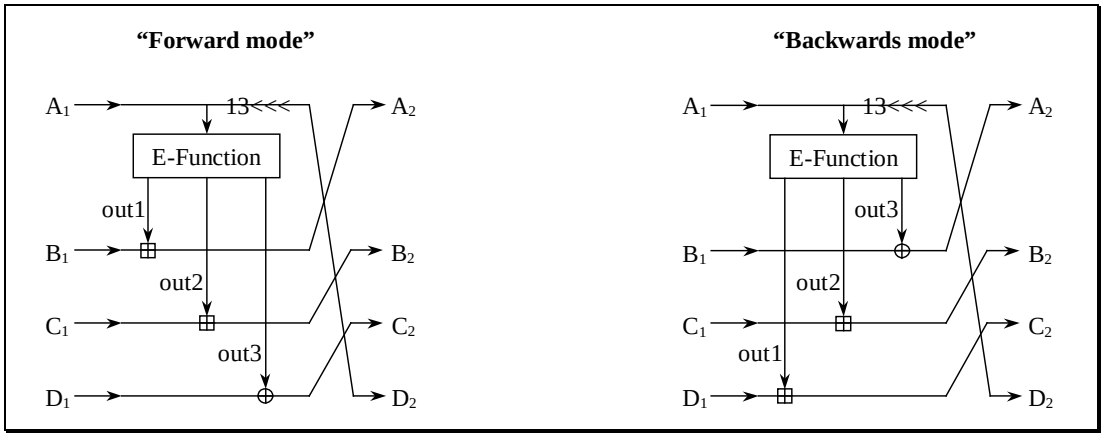

Figure 5: Type-3 Feistel network of the MARS cryptographic core

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, 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.

In the design of the E-function we combined 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:

1. The lower bits of the input to the multiplication have a larger effect on the product than the higher bits. Thus, in the E-function, 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.

2. The most significant bits are the "stronger bits" in the product since they are affected by almost all the input bits. In the combination of the multiplication and the data-dependent rotation, we use these "strong bits" to determine the amounts of the data-dependent rotations.

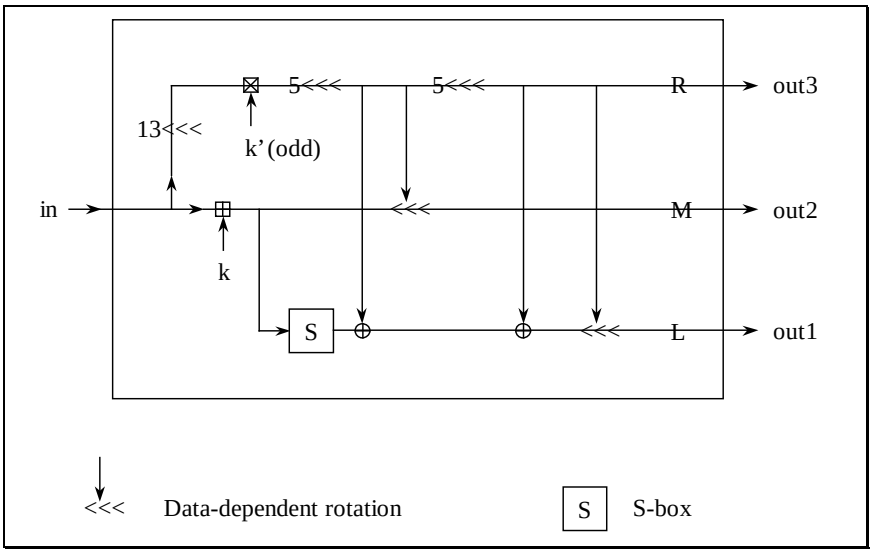

Figure 6: E-Function of the MARS cryptographic core

3. 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.

4. The E-function was designed to be efficient, and to admit a high level of parallelism. In particular, the two most expensive operations (multiplication and S-box lookup) can be carried out in parallel.

4.2 Wrapper Layers

The forward and backwards mixing phases (i.e., the wrapper layers) are essentially inverses of each other. The forward mixing phase begins with the addition of key-words into the data-words, followed by 8 rounds of S-box based, unkeyed mixing. The backwards-mixing phase has 8 rounds of the inverse mixing rounds, followed by key-subtraction.

In each forward and backwards mixing round, one data word (called the source word) is used to modify the other three data words (called the target words). The four bytes of the source word are viewed as indices into two S-boxes, S_0 and S_1, each consisting of 256 32-bit words, and the corresponding S-box entries are xored or added/subtracted into the other three data words. (For S_0 and S_1 we use the first and last 256 entries of the S-box S from the core rounds, respectively.)

Denote the four bytes of the source words by b_0, b_1, b_2, b_3, where b_0 is the lowest byte and b_3 is the highest byte. In the forward mixing rounds we use b_0, b_2 as indices into the S-box S_0 and b_1, b_3 as indices into the S-box S_1. We first xor $S_0[b_0]$ into the first target word, and then add $S_1[b_1]$ to the same word. We also add $S_0[b_2]$ to the second target word and xor $S_1[b_3]$ to the third target word. Finally, we rotate the source word by 24 positions to the right.

In the backwards mixing rounds we use b_0, b_2 as indices into the S-box S_1 and b_1, b_3 as indices into the S-box S_0. We xor $S_1[b_0]$ into the first target word, subtract $S_0[b_3]$ from the second data word, subtract $S_1[b_2]$ from the third target word and then xor $S_0[b_1]$ also into the third target word. Finally, we rotate the source word by 24 positions to the left.

In both forward and backwards mixing, the four words are rotated after each round, 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 becomes the next third target word.

In addition, we sometimes add/subtract one of the target words back into the source word. Specifically, in the

forward phase we add the third target word back into the source word after the first and fifth rounds, and add the first target word back into the source word after the second and sixth round. In the backwards-phase we subtract the first target word from the source word before the fourth and eighth rounds, and subtract the third target word from the source word before the third and seventh round. The reasons for these extra mixing operations are to eliminate some easy differential attacks against the mixing phase, to break the symmetry in the mixing phase and to get faster avalanche.

5 MARS Security

We expect the security level of MARS with an n-bit key to be 2^n for key lengths up to at least 256 bits. We do not expect the security level to grow as rapidly beyond 2^{256}. 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. In a full paper submitted to NIST for AES (Advanced Encryption Standard) [3] 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 [1], since it entails ignoring the 16 rounds of mixing in the cipher).

For linear attacks, we show 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 [2].

For differential attacks [1] 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.

6 Conclusion

MARS is a fast and secure symmetric-key block cipher with a heterogeneous structure. It offers much improved security/performance over existing ciphers by taking advantage of the powerful operations supported in today's computers. Both software and hardware implementations of MARS are remarkably compact, making it also suitable for limited-resource environments such as the smartcard. 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.

7 References

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

[2] 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.

[3] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic, *"MARS – a candidate cipher for AES"*, http://www.research.ibm.com/security/mars.html, July 1998.

[4] B. Schneier, *"Applied Cryptography,"* 2nd edition, John Wiley & Sons Inc, 1996.

[5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *"Handbook of Applied Cryptography,"* CRC Press, 1996.

[6] R. L. Rivest, *"The RC5 Encryption Algorithm,"* Dr. Dobb's Journal, Vol. 20, No. 1, Pages 146-148, January 1995.

[7] B. Schneier, *"The Blowfish Encryption Algorithm,"* Dr. Dobb's Journal, Vol. 19, No. 4, Pages 38-40, April 1994.

[8] National Bureau of Standards, "*Data Encryption Standard*," FIPS PUB 46, January 1977.

[9] J. L. Massey, *"SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm,"* Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, Pages 1-17, 1994.

[10] C. M. Adams, *"Constructing Symmetric Ciphers Using the CAST Design Procedure,"* Design, Codes, and Cryptography, Vol. 12, No 3, Pages 283-316, November 1997.

[11] B. Schneier, *"The IDEA Encryption Algorithm,"* Dr. Dobb's Journal, Vol. 18, No. 13, Pages 50-56, December 1990.

[12] 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, pages 189-199, 1997.

Appendix

The MARS S-box is shown below.

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

}