

Suggested "tweaks" for the MARS cipher

=====

We suggest the following two "tweaks" to the key expansion procedure of the MARS cipher.

=====

1. In the procedure for modification of the multiplication keys, (Lines 20-32 in the pseudocode on Page 22) we would like to handle the most significant bit of the mask M differently than what is currently done.

Specifically, in the modified code we always reset the most significant bit of the mask M to zero. Namely, we replace Lines 25-26 of the pseudocode with

```
25. M_l=1 iff w_l belongs to a sequence of 10 consecutive 0's or 1's
26.      in w, and also 2 <= l <= 30 and w_{l-1} = w_l = w_{l+1}
```

This change has no effect on either the efficiency or the security of the cipher, and it is only meant to make the specification easier to read and understand:

In the English description of this procedure, we explain that after initializing the mask M, we reset to 0 those bits in M that correspond to the "end of runs of 1's or 0's in the key word w". However, the way this step was implemented, the most significant bit of M was reset to 0 only if the corresponding bit of w was 1.

Since the highest bit of w always "correspond to the end of a run", the spec will be more intuitive if we always reset the highest bit of M to 0.

(We thank Brian Gladman for suggesting that we make this change.)

=====

2. The second change is meant mainly to simplify implementation of the key expansion procedure in extremely low-memory environments:

During the first round of the AES, much discussion revolved around the suitability of implementing the AES candidates on low-memory smart cards. We recommend that the low-memory smart card not be used as a criterion in the judging of the AES candidates. We have provided supporting reasons for this position in our official comments (available on NIST's AES page). Some of the reasons include the fact that low-end smart cards are not secure, 16 and 32 bit smart cards with over 1KB of RAM are already available, and new smart cards have built in hardware countermeasures against power attacks. Therefore, the selection of the AES algorithm should not be influenced by low-end smart cards. In spite of these facts, many still insist on analyzing the AES candidates in constrained

environments and consider this to be an issue.

To address this point, we are proposing the following simple change to the MARS key expansion routine, which will enable implementation in severely limited environments. Instead of calculating the 40 words "in one shot", we use essentially the same procedure to calculate 10 words at a time, and run it four times. More details follow:

In the current procedure, we manipulate a temporary array $T[]$ for some time, and then take the resulting array as the words of the expanded key. Namely, currently the process of computing the key consists of

- a. Apply linear transformation to fill a temporary array $T[]$
- b. Stir the array $T[]$ for several rounds (currently, 7 rounds)
- c. Reorder the words in $T[]$ into $K[]$

and then

- d. modify the key words used for multiplication

Instead, we propose to repeat the same process four times, each time computing a quarter of the expanded key array. Namely we propose the following:

1. Repeat four times
 - a. Do linear transformation (using the same formula as before)
 - b. Stir the array $T[]$ for several rounds (using the same formula as before, we propose 4 rounds)
 - c. Reorder the words in $T[]$ into the next ten words in $K[]$
2. modify the multiplication keys

The advantage is that $T[]$ can be a lot smaller than 40 words, and so this new procedure runs faster and takes up less memory than the current one. Moreover, we propose to keep $T[]$ larger than 10 words (specifically, we propose that T be 15-word long), and we believe that this also makes the new procedure cryptographically stronger than the current one.

To see why, note that the new process resembles the way most pseudorandom-generators are built. Namely, you keep a state, and each time you use part of the state to output words, and then re-shuffle it so that you can output more words. In a sense, we are building an "almost pseudorandom-generator", using the previous key expansion procedure as our "re-shuffling" operation.

Implementation of this idea requires a minor change to the linear transformation formula. (The reasoning for this is provided later in this note.) Specifically, the previous formula was

$$T[i] = ((T[i-7] \text{ xor } T[i-2]) \lll 3) \text{ xor } k[i \bmod n] \text{ xor } i$$

where $T[]$ is the temporary array, $k[]$ is the original key and n is the number of words in that key. In the new routine, we instead initialize $T[]$ with the words of $k[]$ at the very beginning (so there is no need for the " $\text{xor } k[i \bmod n]$ " part) and then do in

the j'th iteration (j=0..3)

```
T[i] = T[i] xor ((T[i-7] xor T[i-2]) <<< 3) xor (4i+j)
```

The stirring formula remains unchanged,

```
T[i] = (T[i] + S[ low 9 bits of T[i-1] ]) <<< 9
```

A pseudocode for this new procedure is provided, followed by a more comprehensive discussion.

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

```
1. n is the number of words in the original key buffer k[], (4 <= n <= 14)
2. K[] is the expanded key array, consisting of 40 words
3. T[] is a temporary array, consisting of 15 words, T[0] .. T[14]
4. B[] is a fixed table of four words

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

7. // Initialize T[] with the key data
8. T[0..n-1] = k[0..n-1], T[n] = n, T[n+1..14] = 0

9. for j=0 to 3 do // compute 10 words of K[] in each iteration
10. // Do linear transformation
11. for i=0 to 14 do
12. T[i] = T[i] xor ((T[i-7 mod 15] xor T[i-2 mod 15]) <<< 3) xor (4i+j)

13. // Do four rounds of stirring
14. repeat four times
15. for i = 0 to 14 do
16. T[i] = (T[i] + S[low 9 bits of T[i-1 mod 15]]) <<< 9
17. end-repeat

18. // Store next 10 key words into K[]
19. for i = 0 to 9
20. K[10*j+i] = T[4*i mod 15]
21. end-for

22. // Modify multiplication key-words
23. for i = 5, 7, ... 35 do
24. j = least two bits of K[i]
25. w = K[i] with both of the least two bits set to 1

26. // Generate a bit-mask M (if K[i] should not be modified then M=0)
27. M_1=1 iff w_1 belongs to a sequence of 10 consecutive 0's or 1's
28. in w, and also 2 <= l <= 30 and w_{l-1} = w_l = w_{l+1}

29. // Select a pattern from the fixed table and rotate it
30. r = least five bits of K[i-1] // rotation amount
31. p = B[j] <<< r

32. // Modify K[i] with p under the control of the mask M
33. K[i] = w xor (p and M)
34. end-for
```

We stress that the main consideration in the design of the key expansion was and remains cryptographic strength. Hence we use essentially the same design for the modified procedure. Indeed we believe that the change also makes the procedure cryptographically stronger than the current version. Incidentally, this change also improves the performance of the key setup procedure in high-memory environments.

The advantages of this new procedure over the current one (in order of importance) are:

- a. It is more "one way" than the current one. With the current MARS key setup, knowing only 12 "properly selected" words of the expanded key enables you to reconstruct the original key and subsequently derive all the other words of the expanded key. With the new procedure, even knowing all the words of the expanded key, it is not clear how to efficiently reconstruct the original key. Moreover, it seems that knowing only part of the expanded key words still does not imply an efficient way to derive the other words.
- b. It is much easier to implement it in a limited-memory environment. Specifically, all you need is the 15-word "temporary" array `T[]` (which is 60 bytes of memory), and a few "scratch" words.
- c. It is faster than current procedure. We estimate that this is about 15% faster.

The drawback of the new procedure with respect to the current one is that the user-supplied key cannot be longer than 14 words (448 bits). This does not seem to be a major concern, though, since longer keys can always be hashed (say, using SHA) before they are used in the cipher. (Doing this will also eliminate the "equivalent keys" that can be shown when using long keys with the current procedure.)

The reasoning for the changes that we made to the linear transformation formula are as follows:

1. The reason that we initialize `T[]` with `k[]` instead of incorporating the words of `k[]` to the formula, is to avoid having to store the original key, in addition to the temporary array `T[]`.
2. The reason that we replaced the "xor i" with "xor (4i+j)" is to avoid the remote possibility of having a "fixed point" in this procedure: If we view the "linear transformation + stirring" as a "random permutation", then there is some probability that it has a fixed point. Moreover, there is a small probability that this fixed point can actually result from some key (most likely, a 14-word key. This has probability of about 2^{-32}). If this is true, then this fixed point key implies an extended key where $K[0..9] = K[10..19] = K[20..29] = K[30..39]$

This does not seem to be a big deal. In particular, we don't see why such key would be "weak" for encryption, it is only remotely likely to exist with very long keys (14 words) and the probability of such

"weak keys" is then about $2^{\{-450\}}$.

Still, whenever a repeated function is used, it is a good idea to make some tweaks in it from iteration to iteration, to overcome the possibility of fixed points. A simple such tweak would be to use different xor values in different iterations, thus the "xor (4i+j)" term.

3. Other change that we made are:

- Eliminated the initialization of T[] with seven constants from the S-box. This is just not needed.
- Changed the reordering formula, to interact better with the size of T[].
- In the modification of the multiplication key words, the new procedure uses K[i-1] when modifying K[i] (see Line 32 in the pseudocode above), while the old one uses K[i+3]. The reason for this is to make sure that these two words belong to the same "batch" of 10 words.