Differential Cryptanalysis of Madryga

Ken Shirriff

Address: Sun Microsystems Labs, 2550 Garcia Ave., MS UMTV29-112, Mountain View, CA 94043. Ken.Shirriff@eng.sun.com

Abstract: The Madryga encryption algorithm is susceptible to differential cryptanalysis. The key can be determined with about 5000 chosen plaintexts.

Keywords: block code, differential cryptanalysis, Madryga

Introduction

This paper describes an attack on the Madryga encryption algorithm [2] that uses differential cryptanalysis [1]. This attack can determine the key with about 5000 chosen plaintexts.

The Madryga encryption algorithm

The Madryga encryption algorithm [2] was designed as an alternative to DES that could be efficiently implemented in software. To meet this goal, it performs byte operations (shifts and exclusive-ors) rather than bit operations. It encrypts a block of TL bytes using a key of typically 8 bytes. The algorithm is intended to combine transposition operations (via the shifts) with translation operations (via the exclusive-or with key bytes).

The algorithm consists of two nested loops; the outer loop performs eight rounds of the inner loop, which iterates a base step TL times. Each base step operates on a working frame of three successive bytes (W_1, W_2, and W_3) of the block, which is viewed circularly. The inner loop starts with W_1, W_2, W_3 set to the last two bytes and the first byte respectively of the data block: D_{TL-1}, D_{TL}, D_1. Each successive iteration moves the working frame one byte to the right, so the last inner iteration ends with the last three bytes of the data block: D_{TL-2}, D_{TL-1}, D_{TL}. For most of the analysis in this paper, a block size of TL=8 is assumed.

The base step operates as shown in Figure 1. First, a shift value is obtained by anding W_3 with 7. The 16-bit value W_1W_2 is rotated to the left by the number of bits specified by the shift value.

Finally, W_3 is exclusive-ored with a key-generated byte to conclude the base step. The key-generated byte in each base step is formed by rotating the key to the right 3 bits, exclusive-oring it with a constant (0x0F1E2D3C4B5A6978), and taking the rightmost byte. The base step is shown in Figure 1.

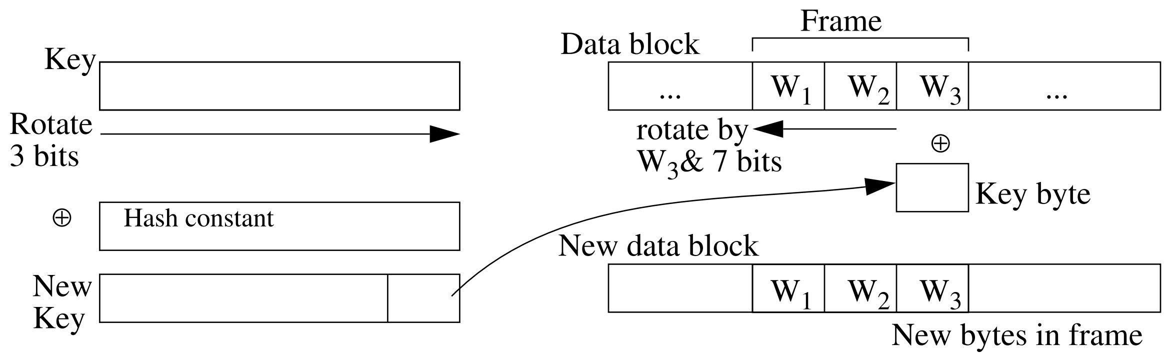

Figure 1. The base step of the Madryga algorithm. Each step modifies a three byte frame of the data block. First, W_1W_2 is rotated by W_3& 7 bits. Next, the key is rotated three bits and exclusive-ored with a constant to generate the new key. Finally, the low-order byte of the new key is exclusive-ored with W_3.

The attack

The attack uses differential cryptanalysis on chosen plaintext. The key idea is to encrypt two plaintexts that differ in one bit. Occasionally, the encryptions of the two plaintexts will continue to differ only in one bit, until the final step, where this one bit will cause the last shifts to be different for the two blocks. By comparing the two ciphertexts, one bit of the final key-generated byte can be determined. Once three bits of the key-generated bytes have been obtained, the encryption can be unwound one step and the attack can be repeated. Through iterations of determining key-generated bytes and unwinding the encryption successive steps, the entire sequence of key-generated bytes can be obtained, breaking the encryption.

Figure 2 illustrates the attack in more detail. Let two plaintexts differing in one bit are encrypted. Suppose, that the single bit change fails to propagate until the input of the final step, where it is in one of the three low-order bits of the final byte. The final step operates on the last three bytes; call the input for the first block e_5, e_6, and e_7 and the output (i.e. the final ciphertext) E_5, E_6, E_7. Call the second block's input and output for the last step e_5, e_6, $e_7 \oplus b$ and F_5, F_6, F_7 respectively, where b is the changed bit. As a result, e_5e_6 will be shifted a different amount from f_5f_6. Thus, E_5E_6 and F_5F_6 will differ by rotation while E_7 and F_7 will differ by the single bit b. To analyze this, call the (unknown) final key-generated byte K. Let S_0 be the shift for the first block (e_7&7) so $S_0 \oplus b$ is the shift for the second block (($e_7 \oplus b$&7). Thus E_5E_6 is e_5e_6 shifted by S_0 and F_5F_6 is e_5e_6 shifted by $S_0 \oplus b$. By comparing E_5E_6 to F_5F_6, we can determine how much more one was shifted than the other, i.e. $e_7 \oplus b - e_7$, which will be +/- b. We can compute F_7-E_7 from the ciphertext. If the two values are the same, then the corresponding bit (b) of K is 0; if the two values differ in sign, then the corresponding bit of K is 1. Thus, by finding suitable plaintext/ciphertext pairs, we can determine the three low-order bits of K, one at a time.

	First block	Second block	Example first block	Example second block
Last step input	$e_5\ e_6\ e_7$	$e_5\ e_6\ e_7 \oplus b$	11100010 11111011 11000110	11100010 11111011 11000010
Shift	shift $e_5\ e_6$ by e_7	shift $e_5\ e_6$ by $e_7 \oplus b$	shift by 6, xor with 34	shift by 2, xor with 34
Xor	xor e_7 with key K	xor with key K	10111110 11111000 11110010	10001011 11101111 11110110
Final result	$E_5\ E_6\ E_7$	$F_5\ F_6\ F_7$	10111110 11111000 11110010 shift -4 add 4 10001011 11101111 11110110	10001011 11101111 11110110

Figure 2. Cryptanalysis of a single step. In the example, b=4, K=34 (00010100). It can be observed that the second block was shifted 4 bits fewer to the left than the first, while the final byte is of the second block is 4 larger. Thus, it can be determined that bit 2 of K is set.

These bits can then be used to unwind the algorithm one step, by rotating E_0E_1 and F_0F_1 to the right by the obtained value. Finally, exclusive-oring E_2 and F_2 with the obtained bits will yield values for e_2 and f_2 that are correct in the three low-order bits, but are still exclusive-ored with the top five bits of K. This operation can continue, working to the left in the ciphertext, obtaining low-order bits of successive K's and unwinding the algorithm.

If we assume an eight-byte data block, the unwinding will work until the 7th step. At this point, the frame wraps around and E_0 and F_0 will be the final bytes of their respective blocks. Recall that we only obtained a value for this byte that is correct only in the low three bits since it is still exclusive-ored with the top five bits of the first K. However, by comparing E_0E_1 to F_0F_1 we still have enough good bits to determine the difference in shift. In addition, we can determine the top 5 bits of the first K. This comparison can be performed simply by trying the 6 different shifts (-1, -2, -4,

+1, +2, +4) and 32 values for the first K to see if a match can be obtained. Thus, the decrypting step will reveal one bit of the current K and the top 5 bits of the K from 6 steps earlier. The encryption can then be unwound another step; now that the first K is completely determined, all bits of the final byte can be obtained, rather than just the low order bits, and used for the next step of the cryptanalysis. Cryptanalysis can similarly proceed backwards through all steps of the encryption, as shown in Figure 3.

Step	Data bytes								Key bytes determined
Ciphertext	M	M	M	M	M	D	D	D	
After step 1	M	M	M	M	D	D	D	UL	L
After step 2	M	M	M	D	D	D	UL	UL	LL
After step 3	M	M	D	D	D	UL	UL	UL	LLL
After step 4	M	D	D	D	UL	UL	UL	UL	LLLL
After step 5	D	D	D	UL	UL	UL	UL	UL	LLLLL
After step 6	D	D	UL	UL	UL	UL	UL	UL	LLLLLL
After step 7	D	UL	UL	UL	UL	UL	UL	UD	LLLLLLA
After step 8	UL	UL	UL	UL	UL	UL	UD	UD	LLLLLLAA
After step 9	UL	UL	UL	UL	UL	UD	UD	UL	LLLLLLAAA
After step 10	UL	UL	UL	UL	UD	UD	UL	UL	LLLLLLAAAA
After step 11	UL	UL	UL	UD	UD	UL	UL	UL	LLLLLLAAAAA
...	...								...

Figure 3. Successive cryptanalytic steps. Bytes in the frame being cryptanalyzed are underlined. Bytes that match between the two encoded blocks are indicated with M. Bytes that are different are indicated with D. Once key-generated bytes are partially determined the encryption can be unwound successive steps to obtain bytes that are valid in the three low order bits; these are indicated by UL. Once key-generated bytes are fully determined, unwound bytes are correct in all bits; these are indicated with UD. Key-generated bytes are indicated with L if the low order bits are known and A if all the bits are known. For example, in the first step, the last three data bytes are examined, yielding for the key. The encryption can be unwound one step, so the last byte can be unwound revealing the low order bits.

By continuing this process of determining the shift and unwinding the encryption, we can obtain the entire key sequence. This allows us to decrypt any block encrypted with that key sequence.

Note that the specific algorithm used to generate the key sequence from the original key is irrelevant to this attack. If the standard Madryga key generation algorithm and key has are used, then the original key can be easily obtained. To determine the key, first generate the key sequence corresponding to a key of all zeros and exclusive-or this with the recovered key sequence. The result is the low byte of the original key, shifted three positions to the right each time. If the Madryga key hash is not known, it is not possible to recover all the bits of the the original key or the key hash. This can be shown by computing the 512x128 matrix showing the influence of each bit of the key and hash on each bit of the output sequence. This matrix is singular, with rank 72. Thus, only 72 of the 128 bits can be recovered. For instance, a key sequence of all zeros is generated if both the key and key hash are all zeros. But it is also generated by key 0x9080000000000000 with key hash 0x8290000000000000 or key 0x2101249240000000 with key hash 0x2521000008000000, or many other combinations.

Analysis of the attack

Empirical testing shows that on the average, about 10,000 chosen plaintext are required to cryptanalyze the algorithm. For efficiency, these plaintexts were generated as sets of 256 that differ only in one byte; each set of 256 yields 1024 pairs that differ in a single bit. Surprisingly, generating pairs from these sets was extremely inefficient for the first few rounds. It took a very large number of sets to get satisfactory pairs. The explanation is that because Madryga requires multiple rounds to sufficiently shuffle the input data, similar blocks are likely to produce similar output after two rounds. So if one pair in the set doesn't provide the needed single bit change, it is unlikely that the others will. Thus, it is considerably more efficient to generate unrelated pairs for the first few rounds.

We can analyze the propagation of a single bit change in more detail. Intuitively, a single bit change will result in avalanche if it is in the low 3 bits (changing the shift) and will otherwise propagate, yielding a probability of approximately 5/8 or 62.5% of surviving a round. For a more detailed analysis, consider the matrix P where P_{ji} is the probability that a single bit change in position i entering a round results in a single bit change in position j at the end of the round. The dominant eigenvector of the 64x64 matrix P shows the eventual distribution of a random bit change; good convergence to the eigenvector occurs after a single round. The dominant eigenvalue of the 64x64 matrix P is approximately 0.57 and shows that, after a couple rounds, a single bit change has about a 57% chance of survival. This is slightly lower than the 5/8 estimate because the distribution in the dominant eigenvector is not uniform; after one round, surviving single bit changes are in positions slightly more likely to avalanche in the next round. After 8 rounds, a single bit change has about a 1% chance of surviving.

The complexity of the attack can be estimated from these values. To cryptanalyze the final step, we need pairs with single bit differences that survived seven rounds and then ended up in the last three bits of the last byte. The probability of a bit difference surviving eight rounds is approximately $.57^8$, and only 1 out of 64 differences will end up in the right position. Thus, an average of about 64/.57:8 or about 5700 pairs will be required for each of the last three bits, and a decreasing number of pairs will be required for successive bits. The difficulty of the attack is determined by the maximum number of pairs required for any bit; in practice this averages about 10000.

Note that a set of 256 plaintexts that vary in one byte will generate 1024 pairs that differ in a single bit. This provides an efficient means of generating pairs with an average 1/4 plaintexts per pair, as opposed to the 2 plaintexts per pair from random pairs. Surprisingly, when the encryption has been unwound to leave 4 or fewer rounds, the set of 256 method is much less efficient than generating random pairs. The explanation is that after a small number of rounds, Madryga hasn't shuffled the data very much. Thus, the behavior of the 1024 pairs is highly correlated and bit differences tend to end up in the same places. Therefore, it is more efficient to generate individual pairs than sets to cryptanalyze the last few rounds. With this optimized pair generation, the algorithm requires about 5000 plaintexts on the average.

Increasing the number of round will increase the resistance of the algorithm to this differential cryptanalytic attack, but only relatively slowly. Measurements show that each additional round increases the number of plaintexts required by a factor of about 1.6. Doubling the number of rounds to 16 only increases the number of required plaintexts to about 200,000.

The Tcl language [5] was used to assist in analyzing Madryga. Tcl is an interactive interpreted programming language designed to run short scripts, while additional functions can be written in C and linked in. Tcl was used as a flexible testbed for performing operations such as encrypting a certain block of data for a certain number of steps, comparing this to the results of unwinding the data, searching for a pair of plaintexts satisfying a condition, and collecting various statistics. The encryption, search, and matrix operation routines were written in C for the speed advantage, while the interpreted nature of Tcl allowed different experiments to take place without the frequent recompilation that a C test program would require. Tcl may prove useful for other cryptographic analysis as it provides a convenient language for interactive experimentation while retaining the speed of a compiled language for computation-intensive operations such as encryption.

Weaknesses in Madryga

This attack reveals that Madryga is very weak in comparison to other block algorithms. For instance, the most effective known attack against DES (linear cryptanalysis) requires about 2^{43}known plaintexts [3]. Several factors are responsible for the weakness of Madryga: ssingle bit changes in Madryga avalanch poorly, the algorithm can easily be unwound step-by-step, and Madryga has unfortunate parity properties.

The fundamental problem is the failure of the bit avalanche property, which was detected by Gustafson et al [4]. In particular, a given output byte has about a 2.5% chance of being unchanged after a single bit change in the plaintext. This is a very high percentage; in comparison, DES will leave a byte unchanged with only the random chance of 1 in 256. The main problem with Madryga is that a bit change will avalanche only if the bit changes the rotation. Since only 3 of the 8 bits are used in a rotation, a bit change has about a 5/8 chance of not causing any change during a particular step of the inner loop and thus during an entire round. Once avalance does occur, it propagates rather slowly, two bytes to the left and an average of one byte to the right. (In contrast, a round of DES normally changes all the bytes.) Thus, even if avalanche occurs immediately, it typically takes 3 rounds before all bytes are affected. Because of this slow propagation, if avalanche is delayed a few rounds, it may not spread to all the bytes.

Thus, the avalanche characteristics of Madryga suffer from several problems. First, each round has a high probability of being unaffected by a single bit change. Second, when avalanche does occur, it propagates slowly. Finally, the number of rounds is too small to ensure that sufficient avalanche will occur.

A second fundamental problem is the ease of unwinding the algorithm step-by-step. Because only three bytes are affected in a simple way during a step, it is straightforward to determine what happend in the step and how to unwind the step. In addition, because the key-generated bytes are a fixed sequence, independent of the data, they are easy to obtain. If the key-generated bytes were a function of the data, the attack in this paper wouldn't work.

Finally, Madryga suffers a parity problem because of the simple combination of the key with the data. As Biham found [3], the parity of the plaintext and ciphertext combined is fixed, for a given key. This can easily be seen from the algorithm since the plaintext and the fixed key sequence are combined with exclusive-or, without any opportunities for “new” bits. In fact, for an 8 bit key and data, the parity is always even. DES avoids this problem through the S-boxes, which substitute variable bits.

References

[1] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In *Advances in cryptology, CRYPTO ’90 Proceedings*, Berlin: Springer-Verlag, pages 2–21, 1990.

[2] W. E. Madryga. A High Performance Encryption Algorithm. In *Computer Security: A Global Challenge.* J. H. Finch and E. G. Dougall(eds.), North Holland: Elsevier Science Publishers, pages 557-570, 1984.

[3] B. Schneier. *Applied Cryptography*. Wiley: New York, 1994.

[4] H. Gustafson, E. Dawson, B. Caelli. Comparison of block ciphers. *Advances in Cryptology — AUSCRYPT’90 Proceedings*, Berlin: Springer-Verlag, pages 208-220, 1990.

[5] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley:Reading, Mass., 1994.

Biographical Sketch

Ken Shirriff is a staff engineer at Sun Labs. He received his Ph.D. in computer science from U.C. Berkeley in 1995.