

Practical Preimages for Maraca

Sebastiaan Indestege*

Bart Preneel

Katholieke Universiteit Leuven

Dept. of Electrical Engineering ESAT/COSIC

Kasteelpark Arenberg 10/2446, B3001 Heverlee, Belgium

and Interdisciplinary Institute for BroadBand Technology (IBBT), Belgium

sebastiaan.insteeye@esat.kuleuven.be bart.preneel@esat.kuleuven.be

Abstract

The cryptographic hash function Maraca was submitted to the NIST SHA-3 competition [4] by Jenkins [3]. In this work, we show a practical preimage attack on Maraca. Our attack has been implemented and verified experimentally. This shows that Maraca does not achieve several important security properties which a secure cryptographic hash function is expected to offer.

1 Introduction

Cryptographic hash functions are easy to compute, deterministic functions that map an input message of arbitrary length to a short, fixed-length digest. They are important building blocks in many cryptographic applications. Secure cryptographic hash functions are required to have several security properties, such as collision resistance and preimage resistance. In this work, we focus on the latter, preimage resistance. Informally, this notion means that, given a hash function output y , it should be difficult to find an input message x hashing to this output.

As recent cryptanalytic advances have raised serious concerns regarding the security of several widely used hash functions, such as MD5 and SHA-1, the National Institute of Standards and Technology (NIST) has recently started a public competition, the SHA-3 competition [4]. This competition aims to develop a new cryptographic hash function standard. Maraca is a hash function proposal that was submitted as a candidate to this SHA-3 competition by Jenkins [3], but it was not selected for round 1 of the competition.

Canteaut and Naya-Plasencia [1] analysed the security of Maraca with respect to collision attacks, and constructed a theoretical collision attack, requiring 2^{237} calls to the compression function and a memory of $2^{230.5}$ bits.

In this work, we propose a practical preimage attack on Maraca. After a one-time precomputation, our attack can find many (first) preimages for any hash output in just a few seconds on an average PC. We have implemented our attack, and verified it experimentally. Of course, this attack can also be used to construct collisions or second preimages for Maraca.

This paper is organised as follows. First, a description of the Maraca hash function is given in Sect. 2. The basic ideas used in our attack are presented in Sect. 3. Section 4 focuses on a component of Maraca, the Maraca S-box, as an important weakness in this component will be exploited in our attack. Section 5 puts everything together, resulting in a practical preimage attack on the Maraca hash function. The practical aspects of this attack are discussed in Sect. 6. Finally, Sect. 7 concludes.

*F.W.O. Research Assistant, Fund for Scientific Research — Flanders (Belgium).

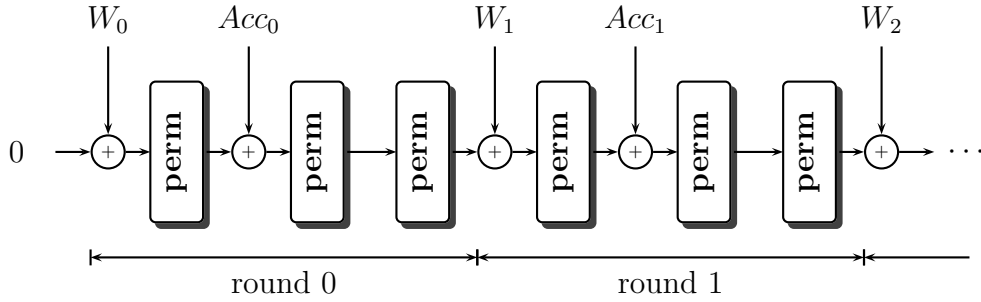


Figure 1: The Maraca Hash Function

2 Description of Maraca

Maraca is a cryptographic hash function proposed by Jenkins [3] as a SHA-3 candidate. It supports digest sizes of up to 1024 bits, and can optionally use a key. For simplicity, we only describe the unkeyed mode of Maraca here. Hashing a message with Maraca consists of three phases: message padding, message processing and digest generation.

First, the input message is padded to a multiple of 1024 bits as follows. If the message ends in a fractional byte, this byte is filled with zero bits. Then, a 16-bit tag containing the number of zero padding bits used, is appended to the message. Finally, zero bytes are used to further pad the message to an integer number of 1024-bit blocks. Note that, unlike many other hash functions, the message length is not included in the padding.

The operation of the Maraca hash function is shown in Fig. 1. Maraca has an internal state of 1024 bits, which is initialised to zero. For each 1024-bit message block W_i , a round is performed. A round consists of the following sequence of operations. First, the message block W_i is XORed into the internal state. Next, a 1024-bit permutation, which will be described in detail in Sect. 2.1, is applied once. Then, up to three message blocks are selected from a window consisting of the past 47 message blocks, where message blocks with a negative index are defined to be all zeroes. This selection of blocks is done in a way that ensures that each message block is used four times in total. Each of these three message blocks is subjected to a different fixed rotation and combined into the accumulator Acc_i using XOR. The 1024-bit quantity Acc_i is then XORed into the internal state. Finally, the same 1024-bit permutation, see Sect. 2.1, is applied twice.

After all message blocks have been processed, 47 blank rounds are performed. These are rounds where no new message blocks are used. Note that in these blank rounds, the 47 last message blocks are still reused via the accumulators Acc_i , even though there are no new message blocks. Finally, the Maraca permutation, see Sect. 2.1, is applied 28 more times. The output digest is then found by truncating the final internal state to the desired length. Thus, digest lengths of up to 1024 bits can be obtained.

2.1 The Maraca Permutation

The Maraca permutation is a fixed, nonlinear permutation operating on 1024 bits. It is shown schematically in Fig. 2. The input to this permutation is the 1024-bit internal state of Maraca, which is arranged as 16 words of 64 bits. First, the same nonlinear bijective 8×8 bit substitution box (S-box) is applied 128 times in parallel. Each S-box takes a single bit from eight distinct words as its input bits, as is shown in Fig. 2, and performs a table lookup as defined in Table 1. This arrangement is intended to allow for a simple bitsliced implementation of Maraca.

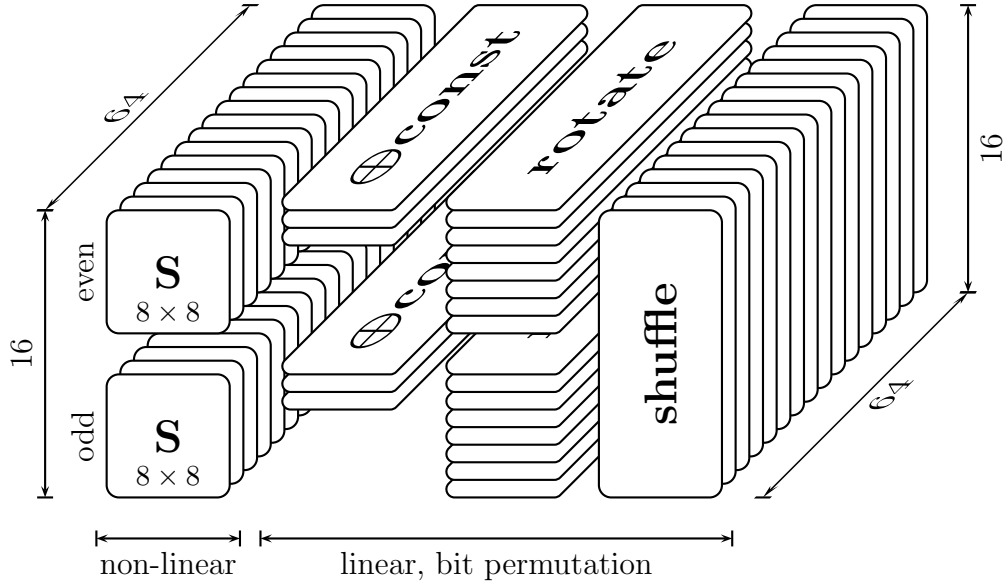


Figure 2: The Maraca Permutation

After the S-box layer, constants are XORed to six words. Then, the bits in each word are rotated by a fixed amount, which is different for each word. Finally, the words are shuffled, i.e., their order is changed. Note that only the S-box layer is nonlinear. Everything that comes after the S-box layer is just an XOR with constants, followed by a reordering of the 1024 bits, i.e., a bit permutation. This is clearly linear, or more precisely as there are constants, affine.

3 Basic Attack Idea

Consider a hypothetical hash function with an n -bit output which, for a certain fixed message length l , is a linear or an affine function over $\text{GF}(2)$. Clearly, any such function can be written as

$$[y]_{n \times 1} = [\mathbf{A}]_{n \times l} \cdot [x]_{l \times 1} \oplus [b]_{n \times 1} . \quad (1)$$

Here, x is a binary column vector containing the l bits of the input message and similarly, y holds the n -bit output digest. The binary matrix \mathbf{A} and the binary vector b allow to express any linear or affine function over $\text{GF}(2)$. In the remainder of this paper, we will drop the distinction between a linear and an affine function, and refer to both as linear.

Note that finding preimages for such a linear hash function is easy. Given any output y , it is easy to find a value for the message x such that $h(x) = y$. Indeed, all messages x for which $h(x) = y$ are simply the solutions of the system of linear equations over $\text{GF}(2)$ given by:

$$\mathbf{A} \cdot x = y \oplus b . \quad (2)$$

Such a system of equations can be solved easily, for instance using Gaussian elimination.

Even though Maraca is clearly not a linear function, it is in essence this method that will be used to construct preimages for Maraca. In the remainder of this work, we will show that by restricting the input messages in a carefully chosen way, it is possible to turn the Maraca hash function into a linear function over $\text{GF}(2)$.

	_0	_1	_2	_3	_4	_5	_6	_7	_8	_9	_a	_b	_c	_d	_e	_f
0_	00	41	26	4f	92	b3	94	bd	0e	47	28	49	9c	b5	9a	bb
1_	8d	a4	8b	a2	1f	56	39	50	83	aa	85	ac	11	58	37	5e
2_	17	76	31	78	a5	84	a3	8a	19	70	3f	7e	ab	82	ad	8c
3_	ba	93	bc	95	08	61	2e	67	b4	9d	b2	9b	06	6f	20	69
4_	98	d1	be	d7	4a	e3	4c	e5	96	df	b0	d9	44	ed	42	eb
5_	55	f4	53	fa	87	46	a1	48	5b	f2	5d	fc	89	40	af	4e
6_	8f	e6	a9	e0	6d	c4	6b	c2	81	e8	a7	ee	63	ca	65	cc
7_	72	d3	74	dd	90	71	b6	7f	7c	d5	7a	db	9e	77	b8	79
8_	91	10	b7	1e	43	e2	45	ec	9f	16	b9	18	4d	e4	4b	ea
9_	5c	f5	5a	f3	8e	07	a8	01	52	fb	54	fd	80	09	a6	0f
a_	86	27	a0	29	64	c5	62	cb	88	21	ae	2f	6a	c3	6c	cd
b_	7b	d2	7d	d4	99	30	bf	36	75	dc	73	da	97	3e	b1	38
c_	c9	c0	ef	c6	1b	32	1d	34	c7	ce	e1	c8	15	3c	13	3a
d_	04	25	02	2b	d6	57	f0	59	0a	23	0c	2d	d8	51	fe	5f
e_	de	f7	f8	f1	2c	05	2a	03	d0	f9	f6	ff	22	0b	24	0d
f_	33	12	35	1c	c1	60	e7	6e	3d	14	3b	1a	cf	66	e9	68

Table 1: The Maraca S-box (hexadecimal).

4 Linearising the Maraca S-box

The only component in Maraca which is not linear is the S-box, which is given in Table 1. However, as was noted by Canteaut and Naya-Plasencia [1], three of the eight output bits are linear functions of the input bits.

Our aim is to linearise the Maraca S-box completely, i.e., to turn it into a linear function. The idea is to choose a linear approximation for the nonlinear S-box, and restrict the inputs to those for which the approximation holds. A trivial example of this approach is to choose any two input values to the S-box. A linear function which maps these two input values to the correct outputs can be found easily.

More concretely, the inputs of the S-box are restricted to some affine space. The reason is that such restrictions can be incorporated easily in the system of linear equations in (2). If every component of Maraca is replaced by a linear approximation, then every intermediate bit is found as a linear combination of input bits and possibly a constant. Restricting a set of intermediate bits to some affine space thus corresponds to adding a number of linear equations to the system (2).

An exhaustive search through all possible affine input spaces was performed. For each such space, it was tested whether the Maraca S-box becomes a linear function when its inputs are restricted to this space. To keep the search complexity low, the problem was reformulated as a tree search, where new restrictions are added at every level of the tree. An early abort strategy was used to accelerate the search. Also, care was taken to avoid duplicate work arising from equivalent representations of the same affine spaces. Such duplicate work can be avoided by only investigating sets of conditions in reduced echelon form. The search takes just a few seconds on an average desktop PC.

The results are remarkable in the sense that imposing just three linear conditions on the input bits of the Maraca S-box can already linearise it. For comparison, at least six conditions are required for the AES [2] S-box, which is also an 8×8 bit S-box. An example of a set of conditions which linearises the Maraca S-box, is the following, where x_7, \dots, x_0 denote the eight input bits:

$$\begin{cases} x_0 = 0 \\ x_2 \oplus x_4 = 0 \\ x_7 = 0 \end{cases} \quad . \quad (3)$$

Many such sets of three or more linear conditions which linearise the Maraca S-box were found. Actually, due to the completeness of the search algorithm used, it is guaranteed that all of them are found.

5 A Preimage Attack on Maraca

Recall from Sect. 4 that, in Maraca, imposing just three linear conditions on the input bits of an S-box can already be sufficient to turn it into a linear function. As a first attempt, consider linearising every S-box in this way, thereby linearising the entire hash function. Clearly, this will not work, as there are $3 \cdot 128$ S-boxes in each round, thus requiring a total of 1152 conditions per round. But there are only 1024 degrees of freedom available in each round, which arise from the 1024-bit message block. In other words, for each round, many more equations than unknowns would be added to the system of equations in (2). As it is very unlikely that, by accident, enough of these equations would be linearly dependent, the system of equations is not expected to have any solutions.

5.1 Making Conditions Dependent

A way to overcome this problem is to make use of the fact that there are many ways to linearise the Maraca S-box. By carefully choosing how to linearise each S-box, an attempt can be made to make as many conditions as possible dependent on each other.

Recall the structure of a Maraca round, see Fig. 1. It consists of three calls to the Maraca permutation, which was introduced in Sect 2.1. Before the first permutation, a message block W_i is XORed into the internal state. Also, after the first permutation, a combination of message blocks denoted by Acc_i is XORed into the state. But in between the second and the third permutation, no additional inputs are added to the state. Investigating the linear part of the Maraca permutation leads to the observation that all eight output bits of an S-box in the second permutation of a round are input to different S-boxes in the third permutation of that round. Since conditions on the bits of a single S-box are required in order to linearise it, only conditions involving a single bit are useful as they apply to a single S-box in both the second and the third permutation of a round.

We propose the following approach. The S-boxes in the first permutation of a round are linearised using three conditions per S-box, for instance using (3). The S-boxes in the second permutation are linearised using a set of four conditions per S-box. Depending on the constants and whether the S-box is even or odd-numbered, one of the following two sets of linearising conditions is used:

$$\left\{ \begin{array}{l} x_1 \oplus x_3 = 0 \\ x_2 \oplus x_5 = 0 \\ x_4 \oplus x_5 = 1 \\ x_6 \oplus x_7 = 0 \end{array} \right. \quad \text{or} \quad \left\{ \begin{array}{l} x_1 \oplus x_3 = 0 \\ x_2 \oplus x_5 = 1 \\ x_4 \oplus x_5 = 0 \\ x_6 \oplus x_7 = 0 \end{array} \right. . \quad (4)$$

Again, x_7 to x_0 denote the S-box input bits. While at first it may seem counter-productive to use more than three conditions per S-box, the advantage is that as much as four S-box output bits are also fixed to a particular value by these conditions.

For the S-boxes in the third permutation, either a set of three or a set of five linearising conditions is used, again depending on the position of the S-box. However, because of the way the S-boxes of the second permutation were linearised, most of these conditions are satisfied automatically. Only one additional condition needs to be added for the even-numbered S-boxes, i.e., just half of the S-boxes in the third permutation. Thus, the total number of conditions per round is now $128 \cdot 3 + 128 \cdot 4 + 64 \cdot 1 = 960$. As there are 1024 degrees of freedom available in each round, 64 degrees of freedom are expected to remain per round.

5.2 Maraca’s Finalisation Phase

The last message block contains an amount of padding, which cannot be chosen by an adversary. However, by choosing an appropriate message length, this padding overhead can be reduced to just 16 bits whose value will be fixed and known a priori. This has the effect of reducing the available degrees of freedom in the last message block slightly, which does not cause any problems.

After all message blocks have been processed, Maraca performs 47 blank rounds. The S-boxes in these rounds also have to be linearised, but no new degrees of freedom are available. This problem can be overcome by building up enough degrees of freedom before the start of the finalisation phase. A simple estimate learns that about $47 \cdot 960 = 45120$ degrees of freedom are required for the finalisation phase. Each normal round yields on average 64 extra degrees of freedom. Thus, after 705 rounds, the required number of degrees of freedom could be achieved. In our experiments, 750 rounds were used, to provide for a reasonable margin of error.

Finally, there are 28 consecutive calls to the Maraca permutation. Note that these do not contribute to the security of Maraca in any way, as they are invertible. Indeed, each of the operations shown in Fig. 2 can be inverted easily. Note that the Maraca S-box is bijective. Starting from a Maraca digest, one first reverts the final truncation by adding arbitrary bits. Then, the final 28 permutations can be inverted in a straightforward way.

Another improvement is to not linearise the S-boxes in the 47 blank rounds, but instead fix the last 47 message blocks to some known value. Then, the 47 blank rounds can be inverted, as all the message blocks used in Acc_i in those rounds are now known. This can also be seen as moving the blank rounds 47 rounds towards the beginning, which may seem pointless at first. However, now the (fixed) message bits used in these “blank rounds” can be chosen, instead of being fixed to zeroes. Note that this principle can also be used to extend the attack to the keyed mode of Maraca. The key appears both at the beginning and the end of the padded message. For a fixed key, it is possible to remove the first and the last message block, which contain the key, and proceed as before.

5.3 Dealing with Contradictions

Up to now, it was silently assumed that none of the additional conditions that are imposed cause any contradictions. It turns out that this is mostly the case, but very sporadically, contradictions do occur. Even though they are rare, they constitute an important issue, as even a single contradiction suffices to make the entire approach fail.

However, it is possible to work around these unfortunate events at the cost of some degrees of freedom. When a contradiction is detected, one can just choose a different linearisation for the problematic S-box. For instance, a trivial linearisation using seven conditions, which is always possible, can be used. As contradictions only occur rarely, the overall impact of this procedure on the available degrees of freedom is close to being negligible.

6 Practical Aspects

Conceptually, our preimage attack on Maraca corresponds to building a system of linear equations over $\text{GF}(2)$, and solving this system of equations. There are some small complications, such as efficiently detecting contradictions, and modifying the system of equations to circumvent them, as was discussed in Sect. 5.3. However, the most important practical obstacle is the large dimension of the system. For 750 rounds, the system of equations has more than 760 000 equations in about as many unknowns. This amounts to a memory requirement of over 67 GB just to store the system. Also, for

such a large system, the cubic time complexity of straightforward Gaussian elimination is prohibitively large.

Note however that the system of equations has a block triangular structure. This is explained by the simple observation that a message block can not affect the rounds before the first use of this message block. Because of the ample diffusion in Maraca, the equations are dense otherwise. Furthermore, in order to be able to efficiently detect contradictions and work around them, it is advantageous to combine the building and the solving the system. This has the additional advantage of limiting the memory usage.

Our implementation of the attack consists of two distinct phases. First, there is a precomputation phase, which has to be done only once, and an online phase which generates a preimage. All complexity is concentrated in the precomputation phase.

6.1 The Precomputation Phase

The following approach, which is based on straightforward Gaussian elimination, was used to build and solve the system of equations simultaneously. Rather than storing equations, the solution space itself is continuously tracked throughout the rounds of Maraca. As this is an affine space, it can be represented by a single displacement vector and a set of basis vectors. These vectors contain a message of a fixed length of 750 message blocks, as well as the 1024-bit internal state at the current position. The solution space is continuously updated as follows.

1. A new message block is added. This corresponds to adding 1024 vectors to the solution space. Each contains a message with a single “1” bit in the current message block and zeroes otherwise. Then, the internal state is updated in every vector to include the XOR with the new message block.
2. Conditions on the internal state bits are imposed in order to linearise the S-boxes of the first permutation of the round. This step corresponds to performing a Gaussian elimination step on several of the internal state bits, and removing certain vectors from the solution space. Note that at this point it is easy to detect any contradictions, and to choose a different linearisation for the S-box in question.
3. Now, the internal state in each of the vectors can be updated to include the (linearised) first permutation of the round. Also, the XOR with Acc_i , which is a combination of previous message blocks, can be performed.
4. In a similar way, the conditions for linearising the S-boxes in the second and third permutation of the round are imposed, further reducing the solution space.

This procedure is repeated until all rounds have been processed. After a single round, the dimension of the solution space has been increased by 64, on average, as 1024 new degrees of freedom were introduced, but only 960 conditions were added. As explained in Sect. 5.2, the 47 last message blocks are set to a fixed value so that the blank rounds can be inverted. This corresponds to another Gaussian elimination step in these rounds, which further reduces the solution space.

In the end, an affine message space is found for which the Maraca hash function, omitting the blank rounds and the final permutations, is linear. Then, it is checked if this solution space allows to reach any value for the internal state before the blank rounds. If not, the number of rounds needs to be increased. Our experiments show that, with 750 message blocks, any state can be reached.

The precomputation phase was implemented to run distributed on a cluster of computers. Each node keeps a part of the basis vectors in memory, and most of the computations can be carried out in parallel. Running the precomputation phase

took about 10.7 CPU-days and 20 GB of distributed memory on a cluster of 32 AMD Opteron nodes. The result of the precomputation is a data file of 94 MB. This is the only data that is required by the online phase.

6.2 The Online Phase

The online phase constructs arbitrary preimages using the data from the precomputation phase. The target digest is first extended to 1024 bits by adding arbitrary bits. Then, the 28 final permutations and the 47 blank rounds are inverted, to retrieve the correct internal state before the start of the blank rounds. Finally, the appropriate basis vectors from the data file are combined using XOR to reach the desired internal state. This entire process takes just a few seconds on an average desktop PC.

As any internal state before the blank rounds can be obtained, our attack is guaranteed to succeed for any given digest value. By extending the target digest to 1024 bits in different ways, it is possible to find multiple messages hashing to a given digest, i.e., multi-preimages. Of course, our attack can also be used to construct collisions or second preimages for Maraca.

7 Conclusion

In this work, we have shown a practical preimage attack on the hash function proposal Maraca. The main weakness that we exploit lies within the Maraca S-box. We have shown that the Maraca S-box can be linearised successfully by imposing additional linear constraints on the S-box inputs. In this way, the Maraca hash function can be turned into a linear function, for which it is easy to construct preimages. Our attack has been implemented and verified experimentally. This clearly shows that Maraca is not preimage resistant nor collision resistant, and hence should not be considered to be a secure cryptographic hash function.

Acknowledgements

This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. This research was conducted utilizing high performance computational resources provided by the University of Leuven, <http://ludit.kuleuven.be/hpc>.

References

- [1] Anne Canteaut and María Naya-Plasencia, “Internal collision attack on Maraca,” 2008 (to appear). Preprint available online at <http://ehash.iaik.tugraz.at/uploads/5/52/Maraca.pdf>
- [2] Joan Daemen and Vincent Rijmen, “The design of Rijndael: AES — the Advanced Encryption Standard,” Springer-Verlag, 2002.
- [3] Robert J. Jenkins Jr., “Maraca: Algorithm Specification,” Submitted to the NIST SHA-3 competition, 2008. Available online at http://burtleburtle.net/bob/crypto/maraca/nist/Supporting_Documentation/specification.pdf
- [4] National Institute of Standards and Technology, “Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family,” Federal Register, vol. 72, nr. 212, pp. 62212–62220, November 2007.