Improved Analysis of Zorro-Like Ciphers

Achiya Bar-On[3], Itai Dinur[2], Orr Dunkelman[1,*],
Virginie Lallemand[4,†], and Boaz Tsaban[3]

[1] Computer Science Department, University of Haifa, Israel
[2] Département d'Informatique, École Normale Supérieure, Paris, France
[3] Department of Mathematics, Bar-Ilan University, Israel
[4] Inria, France

Abstract. Zorro is a 128-bit lightweight block cipher supporting 128-bit keys, presented at CHES 2013 by Gérard et al. One of the main design goals of the cipher was to allow efficient masking, which is a common way to protect against side-channel attacks. This led to a very unconventional design, which resembles AES, but uses only partial non-linear layers. Despite the security claims of the designers, the cipher was recently broken by differential and linear attacks due to Wang et al., recovering its 128-bit key with complexity of about 2^{108}. These attacks are based on high-probability iterative characteristics that are made possible due to a special property of the linear layer of Zorro, which is shown to be devastating in combination with its partial non-linear layer.
In this paper, we analyze the security of Zorro-like ciphers with partial non-linear layers by devising differential and linear characteristic search algorithms and key recovery algorithms. These algorithms exploit in a generic way the small number of Sboxes in a Zorro-like round, and are independent of any specific property of its linear layer (such as the one exploited by Wang et al.), or its Sbox implementation. When applied to the Zorro block cipher itself, we were able to find *the highest* probability characteristics for the full cipher and devise significantly improved attacks. Our differential attack has a time complexity of about 2^{45}, requiring about $2^{41.5}$ chosen plaintexts, and our linear attack has a time complexity of about 2^{45}, requiring about 2^{45} known plaintexts.
Independently of our results, the recently published paper by Rasoolzadeh et al. found similar iterative characteristics for Zorro by exploiting in a different way the devastating property of its linear layer, described by Wang et al. However, our improved key recovery techniques result in differential and linear attacks which are at least 2^{11} times faster. More significantly, the surprisingly large number of Zorro-like rounds analyzed by some of our generic techniques raises questions over the general design strategy of Zorro, namely, the use of partial non-linear layers.
Keywords: Block cipher, lightweight, Zorro, cryptanalysis, differential attack, linear attack.

[*]The third author was supported in part by the German-Israeli Foundation for Scientific Research and Development through grant No. 2282-2222.6/2011.

[†]The fourth author was partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

1 Introduction

Zorro is a 128-bit lightweight block cipher that was proposed by Gérard et al. at CHES 2013 [1]. The cipher supports 128-bit keys, and was designed with the twofold objective of achieving small performance overhead and resisting side channel attacks. More specifically, a central design goal of Zorro was to allow efficient masking (e.g., according to the Rivain and Prouff Scheme [4]).

The resultant design of Zorro is rather unconventional, as it applies a sequence of AES rounds, with a partial Sbox layer in each round, containing only 4 out of the possible 16 Sboxes. Although the cipher was published with an analysis which claims it is secure against standard linear and differential cryptanalysis, this analysis was *heuristic* and not accompanied by a formal proof (as common for AES-like designs). Indeed, in the recent paper [5], the authors published a differential attack with complexity of about 2^{108} and a linear distinguisher with complexity 2^{105} on the full cipher.

The results of [5] are based on 4-round iterative differential and linear characteristics with only 4 active Sboxes. These iterative characteristics are made possible by a combination of symmetric properties of AES-like ciphers, along with a special property of the Zorro (or AES) MixColumns matrix MC, namely, $MC^4 = I$. While it is not clear how to exploit these properties in standard AES-like designs, in the particular case of Zorro, its partial Sbox layers, combined with these properties, gives iterative characteristics with a very small number of active Sboxes.

In this paper, we propose efficient algorithms for the analysis of Zorro-like ciphers with partial Sbox layers. Our algorithms exploit in a generic way the small number of Sboxes in the non-linear layers of the cipher. In fact, our techniques can even be applied to ciphers with an arbitrary linear layer, and/or Sbox implementation, which do not resemble AES-like designs. More specifically, we first devise a generic differential/linear characteristic search algorithm for Zorro-like ciphers, allowing us to search for the best differential/linear characteristics for many rounds with practical complexity. Then, we devise efficient key recovery techniques that exploit differential/linear characteristics, and are particularly efficient for ciphers with partial Sbox-layers, such as Zorro.

By using our characteristic search algorithm, we were able to find higher probability 4-round iterative differential and linear characteristics for Zorro with only 2 active Sboxes, and formally prove (with the aid of a computer) that we found *the best* characteristics (iterative, or not) for full Zorro. We exploit these characteristics using our key recovery techniques, in order to mount attacks with practical complexity: Our best differential attack requires about $2^{41.5}$ chosen plaintexts, takes 2^{45} time and uses less than 2^{10} memory. Our best linear attack requires about 2^{45} known plaintexts, takes about 2^{45} time and uses about 2^{17} memory.

Independently of our work (and a few days prior to the publication of the initial version of this paper), a related paper [3] was published on the eprint server. The results of [3] are based on 4-round iterative characteristics of the same type as in our attacks. However, in contrast to our generic techniques, the characteristics of [3, 5] were found by specifically looking for characteristics of a particular type, which is based on the property $MC^4 = I$ of the Zorro MixColumns operation. This specific technique was initially used in [5], however, the recent paper [3] used different AES-based symmetry properties (leading to fewer active Sboxes).

The techniques that we develop in this paper have several advantages over all previously and independently published analysis:

1. Our key-recovery technique for the differential attack on Zorro-like ciphers allows us to mount an efficient attack by exploiting a relatively short differential characteristic, which spans only 19 out of the full 24 rounds of the cipher.
   Thus, although we use an iterative characteristic which is similar to the one independently found in [3], the time and data complexities of our differential attack are better than that of [3] (which is based on a longer 23-round characteristic) by a factor of about 2^{11}.
2. The key-recovery algorithm of our linear attack is more efficient, giving an attack which is faster than [3] by a factor of about 2^{12}.
3. Our characteristic search algorithm allows us to *formally prove* that the extension of the 4-round differential/linear characteristics are the best differential/linear characteristics (iterative of not) for 8-round Zorro , and thus also for the full 24-round Zorro.
4. Our characteristic search algorithm can be used to find in practical time, the best differential/linear characteristic for 8 rounds (and perhaps more than 10) of almost any modified variant of Zorro, in which the linear layer, and/or Sbox implementation is changed.

All the previous, independent and our improved results on the full Zorro block cipher[1] are summarized in Table 1.

The algorithms described in this paper are mostly based on linearization techniques, which may be of independent interest. These techniques exploit the small number of Sboxes in the non-linear layers of the cipher in order to efficiently analyze many of its rounds, combining in a novel way methods from simple linear algebra and combinatorics.

The paper is organized as follows: We first give a brief description of Zorro in Section 2, and describe our notations in Section 3. Next, we describe our characteristic search algorithm in Section 4, and our key-recovery techniques for differential and linear attacks in Sections 5 and 6, respectively. Then, we describe our specific differential and linear attacks on Zorro in Sections 7 and 8, respectively. Finally, we conclude in Section 9.

2 Description of Zorro

Zorro is a 128-bit lightweight block cipher that was proposed by Gérard et al. at CHES 2013 [1], performing 24 AES-like rounds. The key schedule of Zorro simply adds the 128-bit master key every four rounds, (see Figure 1).

[1]The table does not include the results of [2], which only attacks a weak-key set.

Reference	Time	Data	Memory	Attack Technique
[5]	2^{108}	2^{112} CP	negligible	Differential
[3] †	$\approx 2^{55}$ ††	$2^{55.12}$ CP	2^{17}	Differential
[3] †	$2^{57.85}$	$2^{45.44}$ KP	2^{17}	Linear
Sec. 7	2^{45}	$2^{41.5}$ CP	2^{10}	Differential
Sec. 8	2^{45}	2^{45} KP	2^{17}	Linear

KP - Known plaintext, CP - Chosen plaintext

† The results were obtained independently of ours.

†† The time complexity is specified as $2^{52.74}$ in [3]. However, we take into account the 2^{55} time required to generate the data.

Table 1. Previous, Independent and New Key-Recovery Attacks on Full Zorro

Each Zorro round is made of four AES-like operations, namely SB^*, AC, SR and MC (see Figure 2). SR and MC are exactly the same as the ones used in AES, where AC for round i adds the four constants $(i, i, i, i \ll 3)$ to the 4 bytes of the first row. The main difference of Zorro from AES-like designs is its non-linear operation SB^*, which contains only 4 Sboxes (instead of 16), located in the first row of the state matrix. Moreover, the actual 8×8 Sbox is different, and we refer the reader to the design document [1] for its description.

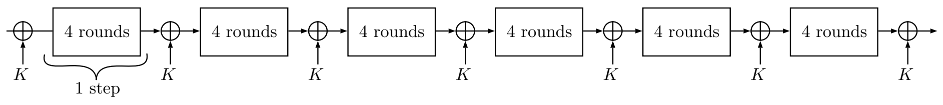


Fig. 1. The Key Schedule of Zorro

3 Notations and Conventions

The rounds of Zorro are numbered $1, 2, ..., 24$. The 128-bit intermediate states and keys are viewed as 4×4 byte matrices, in which the rows and columns are numbered from 1 to 4, as shown in Figure 2.

We denote by K the key associated to the key addition ARK performed every 4 rounds. As in many previous works on AES-based designs, we can exchange the order of the final key addition and linear operations SR and MC. We denote by $\tilde{K} = SR^{-1}(MC^{-1}(K))$ the equivalent key that is associated with this exchange.

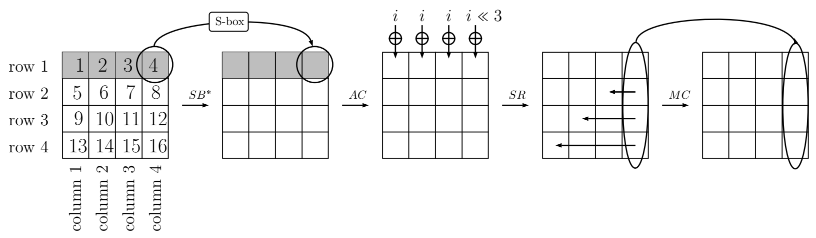


Fig. 2. One Round of Zorro

In order to compute the complexities of our attacks, we use the standard conventions, and measure their time complexity in terms of the number of evaluations of the full cipher. The memory complexity is computed in terms of 128-bit words (i.e., the block size of Zorro).

4 Efficient High-Probability Characteristics Search for Zorro-Like Ciphers

We present a novel efficient high-probability characteristics search algorithm for Zorro-like ciphers. As there is a strong correlation between the probability of a characteristic and its number of active Sboxes, the algorithm looks for characteristics with a limited number of active Sboxes (and thus with high probability). In the rest of this section, we describe the main ideas of the search algorithm for differential characteristics, but note that the search algorithm for linear characteristics is very similar.

We define a *pattern* of a differential characteristic to be a description of the activity for each of its spanned Sboxes (namely, it is a function that specifies for each spanned Sbox whether it is active or not). As r-round Zorro only has $4r$ Sboxes, our first observation is that it has $\binom{4r}{\leq a}$ distinct patterns with at most a active Sboxes. For reasonably small values of r and a, the number of possible distinct patterns is quite small, and we can iterate all of them. In particular, for $r = 8$ and $a = 4$, there are only $\binom{32}{\leq 4} \approx 2^{16}$ distinct patterns.

Our second observation holds, in fact, for any cipher in which the Sboxes are the only non-linear operation. We notice that once we fix a pattern for such a cipher (i.e., fix its active and non-active Sboxes), we can typically calculate the actual characteristics which follow this pattern in an efficient way. This is a result of the fact that once the activity/non-activity of each Sbox is determined, all the possible characteristics reside in a restricted linear subspace which can be easily calculated using linear algebra. After calculating the linear subspace of all the possible characteristics, we apply a post-filtering phase, which enumerates the elements of the subspace, and filters out characteristics in which the active Sbox transitions are impossible (according to the difference distribution table of the cipher). Given that the dimension of the subspace is small enough, we can efficiently post-filter its elements, and thus output all the possible characteristics for the given pattern.

Combining the two observations, when r and a are not too large, we conclude that we can efficiently enumerate all the possible differential characteristics for an r-round Zorro-like cipher with at most a active Sboxes. The full details of the algorithm are given in Appendix A. This appendix shows that the complexity of the algorithm is indeed proportional to $\binom{4r}{\leq a}$, given that the output size (i.e., the number of possible characteristics) is not too large.[2] This implies that the algorithm is practical for a surprisingly wide choice of parameters (e.g., for $r = 10$ rounds with at most $a = 10$ active Sboxes, its complexity is still below 2^{32}).

5 Improved Key-Recovery for Differential Attacks on Zorro-Like Ciphers

In this section, we assume that we have found a differential characteristic with probability p for r rounds of a Zorro-like cipher (e.g., by using the search algorithm of Section 4). We show how to recover the secret key after additional 4 rounds (i.e., after a total of $r+4$ rounds), with data and time complexity of only about $2 \cdot p^{-1}$, using negligible memory.

We denote the difference of the characteristic after i rounds by Δ_i, and thus the characteristic determines Δ_i for $i \in \{0, 1, \ldots, r\}$. The algorithm works by asking for the encryptions of p^{-1} plaintext pairs with input difference Δ_0, and thus we expect that at least one of them follows the characteristic for r rounds. However, since we only have the output after $r + 4$ rounds, it is not clear how to determine which is the right pair that follows the characteristic.

In order to work around this problem, we first note that given the actual values at the output of round $r + 4$, there is, on average, only one 128-bit key, which leads to the fixed difference[3] of Δ_r. If we can efficiently find this key (or keys in general), for each of the p^{-1} ciphertext pairs, then we can perform a trial encryption in order to test if it is correct. Thus, we do not actually need to immediately determine which is the right pair. Indeed, our algorithm uses the special structure of Zorro-like ciphers to quickly find the (small number of) possible keys for each ciphertext pair, and tests each one of them.

The main observation that we use is that given an arbitrary output difference after $r + 4$ rounds, Δ_{r+4}, the full (undetermined) differential sequence starting from the known Δ_r (i.e., Δ_{r+1}, Δ_{r+2} and Δ_{r+3}) attains only one value on average, which we can easily compute using linear algebra. This is due to the fact that the only non-linearity of the system stems from $4 \cdot 4 = 16$ unknown Sbox

[2] As we are mainly interested in characteristics with the smallest number of active Sboxes, their number is typically not very large, and thus it is reasonable to assume that the output size is small.

[3] When partially decrypting the two ciphertexts until round r with a random key, their difference is equal to Δ_r with probability of 2^{-128}.

output differences in the last 4 rounds, which can be described using $16 \cdot 8 = 128$ binary variables. Once these output differences are linearized, we can compute them given the 128-bit constraint imposed by the fixed Δ_r and Δ_{r+4} (which is the difference of the ciphertexts).

After the differential sequence is determined, then (given the actual ciphertext values at the output of round $r+4$), we can efficiently obtain the corresponding key suggestions to test. Indeed, the determined differential transitions for the 16 Sboxes give us the actual possible transition *values*. Similarly to the difference resolution, after the values of the Sbox transitions are resolved, the non-linearity is eliminated from the system, and the key suggestions can be computed using simple linear algebra.

The full details of the algorithm are given in Appendix B. Its data complexity is $2 \cdot p^{-1}$ chosen plaintexts, its time complexity is a bit more than $2 \cdot p^{-1}$, and its memory complexity is very small (less than 2^{10}).

6 Improved Key-Recovery for Linear Attacks on Zorro-Like Ciphers

In this section, we assume that we have found a linear characteristic with bias p for r rounds of a Zorro-like cipher (e.g., by using the search algorithm of Section 4), and show how to recover (a part of) the secret key after an additional round, with data and time complexity of only about p^{-2}. The details of the attack depend on the number of active Sboxes[4] in round $r+1$, which we denote by $ac \in \{1, 2, 3, 4\}$. The memory complexity of the attack is less than $2^{(8 \cdot ac)+1}$ 32-bit words.

We denote the mask of the characteristic after i rounds by Ω_i, determining Ω_i for $i \in \{0, 1, \ldots, r\}$. The algorithm works by asking for the encryptions of p^{-2} arbitrary plaintexts, and thus we expect to obtain a strong linear distinguisher after r rounds. In order to efficiently exploit this distinguisher, we first exchange the order of the final linear operations ARK with MC and SR (and thus we will actually recover bytes of the key $\tilde{K} = SR^{-1}(MC^{-1}(K))$, as denoted in Section 3). This allows us to "peel-off" the final linear operations. As the last Sbox layer contains only ac active Sboxes with a non-zero mask, given a plaintext-ciphertext pair, we can actually compute the parity of the linear equation after r rounds (determined by Ω_r) for all, except $8 \cdot ac$ bits.[5] Thus, the parity of the linear equation after r rounds, depends only on the $(8 \cdot ac)$ bits of $\tilde{K}$, corresponding to the active Sboxes, and a fixed 1-bit linear combination of the key.

We now give a rough description of how to efficiently recover the $8 \cdot ac$ bits of $\tilde{K}$. The algorithm works in two consequential stages, where in the first stage we group the ciphertexts according to $(8 \cdot ac) + 1$ bits:

[4]Although the characteristic spans only r rounds, it can still limit the number of active Sboxes in the next round (due to slow diffusion properties of the cipher).

[5]Up to a fixed parity, which depends on a linear combination of the bits of $\tilde{K}$.

1. The single parity bit, computed by application of the mask Ω_r to $SR^{-1}(MC^{-1}(C))$ on $128-(8 \cdot ac)$ bits (XORed with the corresponding bit, computed from the side of the plaintext).
2. The remaining $8 \cdot ac$ bits of $SR^{-1}(MC^{-1}(C))$ (XORed with the corresponding bits computed from the side of the plaintext).

The grouping procedure is implemented by allocating $2^{(8 \cdot ac)+1}$ counters for the possible values of these $(8 \cdot ac)+1$ bits, and incrementing the corresponding counter for each plaintext-ciphertext pair. This initial grouping stage requires about p^{-2} time (as we perform very little work for each plaintext-ciphertext pair).

We now have $2^{(8 \cdot ac)+1}$ counters, and we are ready to execute the second phase, in which we guess the $8 \cdot ac$ bits of $\tilde{K}$ (corresponding to the active Sboxes). For each guess, we iterate over the $2^{(8 \cdot ac)+1}$ counters and sum all the parities after r rounds. Thus, we can filter out almost all wrong guesses in which there is no significant bias (of at least p). The complexity of the guessing phase is about $2^{8 \cdot ac} \cdot 2^{(8 \cdot ac)+1} = 2^{(16 \cdot ac)+1}$. Assuming that $p^{-2} \gg 2^{(16 \cdot ac)+1}$ (which is indeed the case for our attacks on Zorro), then the time complexity of the attack remains about p^{-2}.

We note that the guessing phase of the attack is not optimal and can be further improved (although this improvement does not effect the complexity of our linear attacks on Zorro). The full details of the key-recovery algorithm, along with the various optimizations will appear in a future version of this paper.

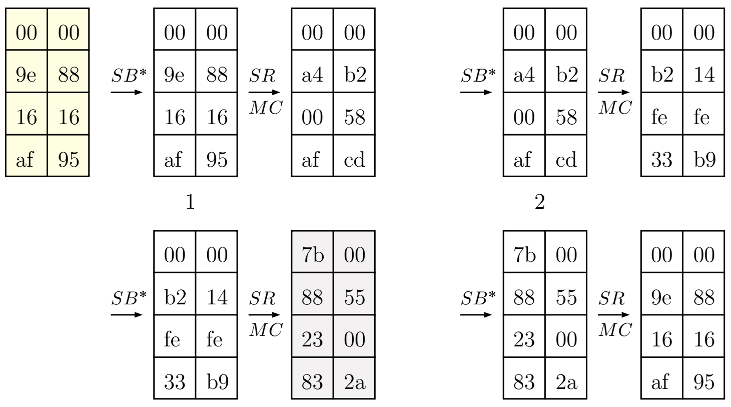
7 Improved Differential Attack on Full Zorro

In order to mount a differential attack on Zorro, we first apply the differential characteristic search algorithm of Section 4, and then use the key recovery technique of Section 5.

Differential Characteristic Search we applied the differential search algorithm of Section 4 on $r = 8$ rounds of Zorro with a maximum of $a = 4$ active Sboxes. The highest probability characteristic that we found is obtained by concatenating 2 instances of the 4-round iterative characteristic given[6] in Figure 3. In fact, there are 2 additional linearly-dependant variants (over GF^{2^8}) of the presented characteristic with the same probability, as shown in Table 2. Additional similar characteristics are obtained by exchanging the order of the two columns.

As the transition probability for each active Sbox is 6/256, the probability of the 8-round characteristic is $(6/256)^4 > 2^{-22}$. Moreover, according to the difference distribution table of Zorro, any differential characteristic with at least 5 active Sboxes has probability less than $2^{-4.5 \cdot 5} < 2^{-22}$. Thus, our 8-round characteristic is highest probability differential characteristic for 8-round Zorro (and it also gives the best differential characteristic for the full 24-round Zorro).

[6]Similar iterative characteristics were independently found in [3].



The full characteristic has probability $(6/256)^2 \approx 2^{-11}$. It is obtained by duplicating the two columns in order to obtain each full state.

Fig. 3. A 4-round Iterative Characteristic for Zorro, Described Using 2 Symmetric Columns for Each State

A	00
B	E
C	00
D	F

	A	B	C	D	E	F
1	0x7b	0x88	0x23	0x83	0x55	0x2a
2	0xea	0x5c	0x5d	0xe4	0xa8	0x71
3	0xf7	0x16	0x8c	0x3a	0x4f	0xa8

The full characteristic is constructed similarly to Figure 3. Additional similar characteristics are obtained by exchanging the order of the two columns.

Table 2. Generic Representation of the Three 4-Round Iterative Characteristics of Probability $(6/256)^2$, Along with Their Values

Key Recovery for the Differential Attack In order to exploit the characteristic in an attack, we extend it up to round 19 (see Figure 5). It has 8 active Sboxes, and thus has probability $(6/256)^8 \approx 2^{-43}$. Our aim is to apply the algorithm of Section 5 in order to recover the secret key. However, the straightforward implementation of this algorithm recovers the key of the 24-round cipher, given the output difference after round $24 - 4 = 20$, rather than round 19.

In this attack, however, we can apply the key recovery algorithm on another round, with no added complexity. The main observation that we use is that we can exploit the specific super-Sbox structure of Zorro (and AES-like designs in general), and extend the characteristic with 2 more inactive Sboxes in round 20 (see Figure 4). Thus, we have a total of 16 active Sboxes (similarly to 4 fully active Zorro rounds), whose unknown outputs can be linearized using a simple variant of the algorithm of Section 5.

According to Section 5, as the 19-round characteristic has a probability of about $p = 2^{-43}$, the data complexity of the attack is about $2 \cdot p^{-1} = 2^{44}$ chosen plaintexts, its time complexity is about 2^{45}, and its memory complexity is less than 2^{10}.

Reducing the Data Complexity Using Structures We can reduce the data complexity of the attack by a factor of 6 by using structures that exploit all the 3 characteristics of Table 2, and the 3 additional ones obtained by rotating their columns by one byte to the right. This is a common technique in differential cryptanalysis, and was used (for example) in the related paper of [5].

Each structure we use is an affine subspace of dimension 6, which is constructed from an arbitrary plaintext, by XORing to it the all the 2^6 linear combinations (over $GF(2)$) of the 6 initial differences of the characteristics of probability $p = 2^{-43}$. Such a structure contains $6 \cdot 2^5$ plaintext pairs which we can exploit, thus reducing the data complexity of the attack by a factor of 6 to about $2^{41.5}$. The time complexity remains the same, as we still need to process each of the $6 \cdot 2^5$ plaintext pairs in each structure separably. The memory complexity remains less than 2^{10}, as the structures we use contain only 2^6 elements.

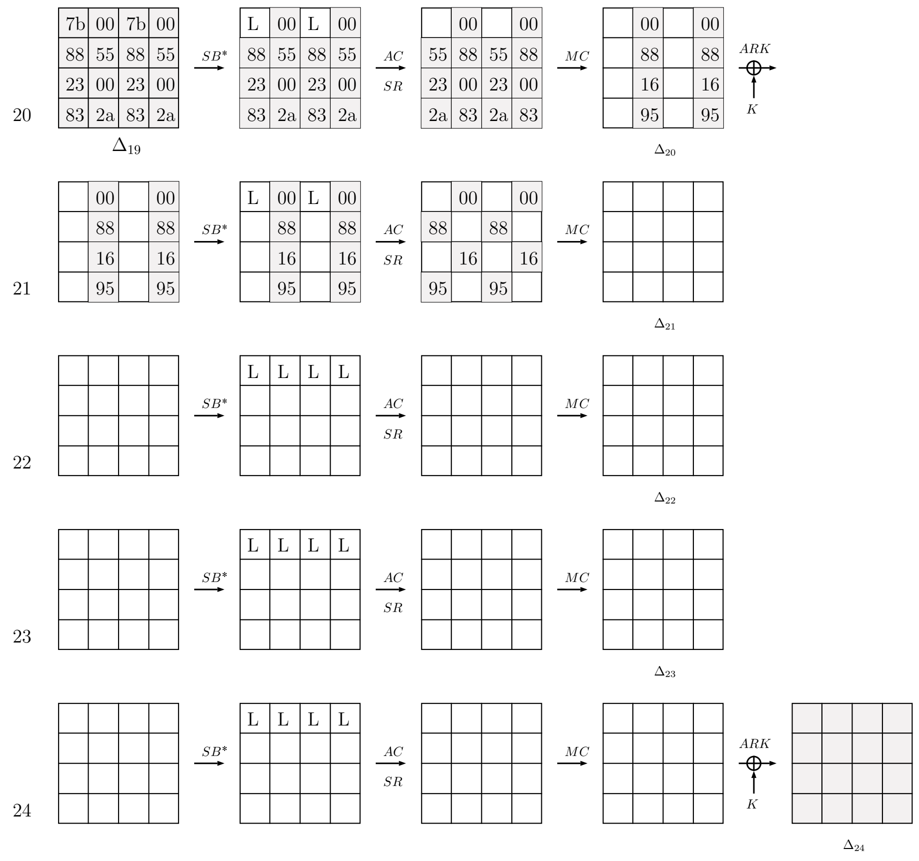
8 Improved Linear Attack on Full Zorro

In order to mount a linear attack on Zorro, we first apply the linear characteristic search algorithm of Section 4, and then use the key recovery technique of Section 6.

Linear Characteristic Search We applied the linear characteristic search algorithm of Section 4 on $r = 8$ rounds of Zorro with a maximum of $a = 4$ active Sboxes. Similarly to the differential case, the best characteristics are a concatenation of two 4-round iterative linear characteristics, which can be viewed as counterparts of the differential ones, and follow a similar representation as in Table 2. In particular, we can obtain a 4-round iterative characteristic with a bias of $(56/256)^2$ by setting the following byte values[7] in Table 2: $A = 0x88, B = 0x5f, C = 0xaa, D = 0xa3, E = 0x25, F = 0xea$. Furthermore, as our linear characteristic search algorithm indicates (similarly to the differential case), the extended 8-round characteristic is the highest bias linear characteristic for 8-round Zorro (and it also gives the best linear characteristic for the full 24-round Zorro).

Key Recovery for the Linear Attack In order to exploit the linear characteristic in an attack, we extend it up to round 23. It has 10 active Sboxes, and thus has a bias of $p = (56/256)^{10} \approx 2^{-22}$. Our goal is to apply the algorithm of

[7] Similar iterative characteristics were independently found in [3].



The output differences of the 16 Sboxes marked with L are initially unknown. They are linearized and recovered according to Section 5, leading to an efficient key recovery.

Fig. 4. Key-Recovery Attack on Full Zorro Using a 19-Round Characteristic

Section 6 in order to recover the secret key. Recall that the details of this algorithm depend on the number of active Sboxes in round 24, which is $ac = 2$ in our case. Thus, a straightforward application of this algorithm recovers 2 bytes of $\tilde{K}$ using $p^{-2} = 2^{44}$ known plaintexts.

In order to recover additional 2 bytes of $\tilde{K}$, we can simultaneously and independently (using the same data) exploit the variant of the same linear characteristic, in which the 2 columns are swapped. Furthermore, we can simultaneously exploit another variant of the iterative characteristic which spans rounds 2–24 (with the active Sboxes in round 2), and apply the key recovery on the encryption side. This allows us to recover 2 bytes of K, and additional 2 bytes can be simultaneously recovered by swapping the columns in the last characteristic. As the time complexity bottleneck in all of these 4 simultaneous attacks is the actual collection of data, the total time complexity of recovering the 8 bytes of key material remains about 2^{44}, and the memory complexity is less than $4 \cdot 2^{(2 \cdot 8)+1} = 2^{19}$ words of 32 bits, or 2^{17} words of 128 bits.

After determining the 8 bytes of key material used in rounds 1 and 24 which contribute to all non-linear operations, we can "peel off" this non-linearity and apply the same ideas to the inner rounds 2 and 23 in order to recover the 8 additional (linear combinations of) key bytes, which contribute to the non-linearity in these rounds. This is done by exploiting the iterative characteristics in which the active Sboxes are in round 2, and in round 23. However, due to the dependency of the inner-round attacks on the previously recovered 8 bytes, it is not obvious how to perform these attacks simultaneously, and thus (in order to avoid the large memory overhead of storing the original data) we can request additional 2^{44} known plaintexts in order to recover the rest of the key. This leads to an attack that uses 2^{45} known plaintexts, runs in 2^{45} time, and requires memory of about 2^{17} words of 128 bits. The full description and additional optimizations of the attack will be given in a future version of this paper.

9 Conclusions

In this paper, we described several algorithms for the analysis of Zorro-like ciphers with partial Sbox layers. These algorithms enable us to significantly improve the previous and independently published attacks on full Zorro. Furthermore, due to the generic nature of our algorithms, they can be efficiently applied to analyze modified variants of Zorro (where the linear layer and/or the Sbox implementation are completely redesigned), on which the other attacks strategies appear to fail.

References

1. Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2013.
2. Jian Guo, Ivica Nikolic, Thomas Peyrin, and Lei Wang. Cryptanalysis of Zorro. Cryptology ePrint Archive, Report 2013/713, 2013. `http://eprint.iacr.org/`.
3. Shahram Rasoolzadeh, Zahra Ahmadian, Mahmood Salmasizadeh, and Mohammad Reza Aref. Total Break of Zorro using Linear and Differential Attacks. Cryptology ePrint Archive, Report 2014/220, 2014. `http://eprint.iacr.org/`.
4. Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
5. Zhiyuan Guo Yanfeng Wang, Wenling Wu and Xiaoli Yu. Differential Cryptanalysis and Linear Distinguisher of Full-Round Zorro. Cryptology ePrint Archive, Report 2013/775, 2013. `http://eprint.iacr.org/`.

A Details of the Efficient Search for High-Probability Characteristics for Zorro-Like Ciphers

In this section, we describe the details of the search algorithm for high-probability characteristics. We recall that we only explicitly deal with differential characteristics, and the search algorithm for linear characteristics is very similar.

A.1 Analysis of a Differential Pattern

We describe the 2-step pattern-analysis algorithm which we use in order to analyze a single differential pattern (using the definitions and notations of Section 4).

Calculating the Linear Subspace of a Pattern We maintain a symbolic representation of the 128-bit state difference at round i, ST_i, using 128 linear combinations. Each linear combination is initialized with a 1-bit variable, representing the corresponding unknown state difference bit in the first round $\Delta(X_0)$ (before the first Sbox layer). Additionally, we allocate a linear equation system E_i (which is empty at first), and describes linear constraints on the characteristic, which are imposed by the inactive Sboxes. At the end of the algorithm (after the final round, r), the subspace of all the possible characteristics is described by the null-space of E_r.

The following round-linearization algorithm describes how we update ST_i and E_i to ST_{i+1} and E_{i+1}, according to the activity pattern of the Sboxes in round $i+1$ (starting from round $i=0$).

1. Allocate and initialize $ST_{i+1} \leftarrow ST_i, E_{i+1} \leftarrow E_i$.
2. For each non-active Sbox (according to the pattern of round i):
   (a) Add 8 equations to the system E_{i+1}, which equate the corresponding 8 bits in ST_{i+1} to zero. If the dimension of the null-space of E_{i+1} is 0 (i.e., there is no non-zero solution to system, and thus no matching characteristic), return ST_{i+1} and E_{i+1} as $NULL$, and exit.
3. For each active Sbox (according to the pattern of round i):
   (a) Replace the corresponding 8 linear combinations in ST_{i+1} with newly allocated variables.
4. Set $ST_{i+1} \leftarrow L(ST_{i+1})$, namely update the symbolic state ST_{i+1} according to the linear function of the cipher, L.

Given a pattern, the linear subspace of all possible characteristics for r rounds is calculated with the following algorithm:

1. Initialize ST_0 with 128 new variables, and E_0 with an empty set of equations.
2. For $i=0$ to $i=r-1$:
   (a) Run the round-linearization algorithm for round $i+1$, in order to calculate ST_{i+1} and E_{i+1}, if they are $NULL$, return $NULL$ and exit.

3. Output a basis B for all the possible characteristics of the pattern using the null space of E_r. This basis is represented as a set of b free (unconstraint) linear variables, and linear combinations of these variables, as follows: the 128 linear combinations of the initial state ST_0, and the $16 \cdot a$ linear combinations of all the inputs/outputs of the a active Sbox transitions (according to the pattern).

Post-Filtering the Linear Subspace of a Pattern Once we obtain a basis B for all the possible characteristics of the pattern, we apply a simple post-filtering algorithm. Recall that B is represented as a set of b free (unconstraint) linear variables, and linear combinations of these variables.

1. For of the 2^b possible values of the free variables:
   (a) For each active Sbox transition:
      i. Calculate the actual input/output for the Sbox transition by plugging in the values of the free variables.
      ii. Check in the difference distribution table of the cipher (e.g., Zorro) whether the differential transition is possible, and if not, go back to Step 1.
   (b) Output the full characteristic according to the current value of the free variables.

We note that it is possible to optimize the post filtering in various situations by choosing the free variables to be input/output bits of a restricted set of Sboxes. This enables us the iterate in advance only over the input/output difference transitions, which are possible according to the difference distribution table of these Sboxes. The optimization can be particularly useful when the filtered linear subspace is of a relatively large dimension (and thus we have less restrictions on the choice of free variables).

A.2 The Full Characteristic Search Algorithm

We describe a simple algorithm that outputs all the possible differential characteristics for r-round Zorro with at most a active Sboxes.

1. For each of the possible $\binom{4r}{\leq a}$ distinct differential patterns:
   (a) Apply the pattern analysis algorithm above to output all the differential characteristics with the current pattern.

A.3 Optimizing the Search Algorithm Using Pattern-Prefix Search

We describe an optimization of the characteristic search algorithm, which is based on the observation that we can analyze together many common patterns with the same prefix. This allows us (for example) to dispose of all the patterns whose common prefix is not possible (instead of analyzing and disposing each one separately). In the particular case of Zorro, there is no characteristic in which the first 16 Sboxes (in 4 rounds) are non-active, and we can dispose all of the patterns in which the first 16 Sboxes are non-active after analyzing only 4 rounds. An additional advantage of this algorithm is that it reduces the average amount of work (mostly linear algebra) performed for each pattern.

The algorithm PPS (Pattern-Prefix Search) iterates over the tree of possible prefixes of patterns using the DFS (Depth First Search) algorithm. The global parameters of PPS are the number of rounds to analyze, r, and the maximal number of active Sboxes, a. The parameters which are passed to each node of the tree are: the round number i, the current Sbox index in the round $s \in \{0, 1, 2, 3\}$, the current number of active Sboxes in the prefix, ca, and ST_i, E_i (as in the standard pattern-analysis algorithm). Thus, the PPS algorithm is initially called with parameters $PPS(i, s, ca, ST_0, E_0)$, where $i = 0$, $s = 0$, $ca = 0$, ST_0 is initialized with 128 new variables and E_0 is an empty set of equations.

1. If $i = r$ (i.e., we finished iterating over all the Sboxes of the pattern), then the r-round pattern is fully determined by the path to the root of the tree. Thus, calculate the basis B for all the possible characteristics of the pattern (using E_r). Finally, post-filter the characteristics (as in the pattern-analysis algorithm), and return them.
2. Allocate a node n_1 for the case that Sbox with index s in round i is inactive (duplicating the current ST_i, E_i): For this node, add 8 equations to the system E_i, which equate the corresponding 8 bits in ST_i to zero. Denote the (yet undetermined) output set of this node as OUT_1.
   – If the dimension of the null-space of E_i is 0 (i.e., there is no non-zero solution to system, and thus no matching characteristic), delete this node and set $OUT_1 = \emptyset$.
   – Otherwise, the dimension of the null-space is greater than 0. If $s = 3$ (i.e., we finished iterating over all the Sboxes of the current round i), then set $ST_{i+1} = L(ST_i)$ (i.e., update the symbolic state ST_{i+1} according to the linear function of the cipher, L), also set $E_{i+1} = E_i$. Recursively call $PPS(i + 1, 0, ca, ST_{i+1}, E_{i+1})$ and set OUT_1 according to the returned output.
   – Otherwise, the dimension of the null-space is greater than 0, and $s < 3$. Recursively call $PPS(i, s+1, ca, ST_i, E_i)$ and set OUT_1 according to the returned output.
3. If $ca = a$ (i.e., we have reached the maximum number of active Sboxes), return OUT_1.
4. Otherwise ($ca < a$) allocate a node n_2 for the case that Sbox with index s in round i is active (duplicating the current ST_i, E_i): For this node, replace the corresponding 8 linear combinations in ST_i with newly allocated variables. Denote the (yet undetermined) output set for this node as OUT_2.
   – If $s = 3$ (i.e., we finished iterating over all the Sboxes of the current round i), then set $ST_{i+1} = L(ST_i)$ and $E_{i+1} = E_i$. Recursively call $PPS(i+1, 0, ca+1, ST_{i+1}, E_{i+1})$ and set OUT_2 according to the returned output.
   – Otherwise, $s < 3$. Recursively call $PPS(i, s + 1, ca + 1, ST_i, E_i)$ and set OUT_2 according to the returned output.

5. Return $OUT_1 \bigcup OUT_2$.

Complexity Analysis Let $T(node)$ be the average complexity of evaluating a node in the recursive tree, without iterating and post-filtering the solutions. As the number of evaluated nodes is proportional to $\binom{4r}{\leq a}$, the complexity of the algorithm can be estimated by the formula $\binom{4r}{\leq a} \cdot T(Node) + SOL$, where SOL is the total number of solutions that we need to post-filter.[8] Since we cannot determine in advance the value of SOL, we will estimate it according to the total number of characteristics which remain *after* post-filtering (i.e., the actual output size), which we denote by OUT.

In order to relate SOL and OUT for the particular case of Zorro, we note that an arbitrary input-output transition for the Sbox is possible with probability of about $2^{-1.5}$, and thus if we have at most a active Sboxes, then we expect that $OUT \geq SOL \cdot 2^{-1.5a}$, or $SOL \leq OUT \cdot 2^{1.5a}$. Consequently, the time complexity of the algorithm can be upper bounded by $\binom{4r}{\leq a} \cdot T(Node) + OUT \cdot 2^{1.5a}$. Assuming that the the output size OUT is not too big, then the complexity of the algorithm is proportional to $\binom{4r}{\leq a}$.

B Details of the Improved Key-Recovery for Differential Attacks on Zorro-Like Ciphers

In this section, we give the details of the key-recovery algorithm for differential attacks on Zorro-like ciphers, using some specific notation defined in Section 5.

The Main Key-Recovery Algorithm The algorithm makes use of 2 auxiliary matrices, A_1 and A_2, which are independent of the actual key and data, and are computed during preprocessing.

– Given the 96×128 matrix A_1, and Δ_{r+4}, the 96-bit vector $A_1 \cdot \Delta_{r+4}$ describes all the $12 \cdot 8 = 96$ unknown output differences for the Sboxes of rounds $r+1$, $r+2$ and $r+3$. Note that once the output differences of these 12 Sboxes are known, computing the full Δ_{r+1}, Δ_{r+2} and Δ_{r+3} can be done by simple linear algebra.
– Given the $128 \times (128 + 256)$ matrix A_2, and a $(128 + 256)$-bit vector v (comprised of the 128-bit ciphertext, and $2 \cdot (32 \cdot 4) = 256$-bit input-output values of all the Sboxes of the last 4 rounds), the product $A_2 \cdot v$ gives a suggestion of the 128-bit key K.

The full algorithm is as follows:

1. Compute the matrices A_1 and A_2 (as described at the end of this section).
2. Ask for the encryptions of p^{-1} plaintext pairs with input difference Δ_0. For each pair (P, C) and (P', C'):

[8] As post-filtering a solution is very simple, we assume it can be done in unit time.

(a) Compute $\Delta_{r+4} = C \oplus C'$, and then calculate $A_1 \cdot \Delta_{r+4}$. This allows to compute the input-output differences of the 16 Sboxes in rounds $r + 1, r + 2, r + 3, r + 4$.
(b) Check for each of the 16 Sboxes, whether the input-output difference transition is possible according to the difference distribution table. If it is impossible, then dispose this pair and analyze the next pair by going back to Step 2.
(c) Compute according to the difference distribution table, a list of vectors *List*, containing $2 \cdot (32 \cdot 4) = 256$-bit vectors, specifying all the possible input-output values of all the 16 Sboxes of the last 4 rounds.
(d) For each 256-bit vector in *List*, denoted by w:
  i. Denote by v the $(128 + 256)$-bit vector, comprised of the 128-bit ciphertext C, and the 256-bit vector w (specifying the input-output values for all the Sboxes of the last 4 rounds). Obtain a suggestion for the key K by computing product $A_2 \cdot v$.
  ii. Test the key using a trial encryption, and if it succeeds, return it.

Complexity Analysis The data complexity of the attack is $2 \cdot p^{-1}$ chosen plaintexts. For each plaintext-ciphertext pair, we perform some simple linear algebra operations, whose complexity is generally proportional to a full cipher evaluation.[9] As noted in Section 5, we expect to test only 1 key per plaintext pair, and thus we can estimate the time complexity of the attack to be slightly higher than $2 \cdot p^{-1}$ cipher evaluations (given that the preprocessing complexity is negligible compared to p^{-1}).

The memory complexity of the attack is less than 2^{10} words of 128 bits, required in order to store A_1 and A_2. Note that the elements of *List* can be generated “on-the-fly”, and we do not need to store them.

Calculating the Differential Transitions From the Output Difference This preprocessing algorithm is given as an input Δ_r (which is known from the characteristic) and computes a 96×128 matrix A_1, such that given Δ_{r+4}, the 96-bit vector $A_1 \cdot \Delta_{r+4}$ describes all the $12 \cdot 8 = 96$ unknown output differences for the Sboxes of rounds $r + 1$, $r + 2$ and $r + 3$. Once the output differences of these 12 Sboxes are known, computing the full Δ_{r+1}, Δ_{r+2} and Δ_{r+3} can be done by simple linear algebra.

The algorithm symbolically maintains the state difference of round i (Δ_i), denoted by ST_i (which is initialized for $i = r$ with the known Δ_r).

1. For each round $i \in \{r, r + 1, r + 2, r + 3\}$:
  (a) Given ST_i, compute ST_{i+1} by allocating $4 \cdot 8 = 32$ new linear variables for the output of the 4 Sboxes of round $i + 1$, and then symbolically applying the linear layer L, obtaining $ST_{i+1} = L(ST_i)$ (i.e., a symbolic representation of Δ_{i+1}).

[9]We can further reduce the complexity of the linear algebra using various low-level techniques (e.g., by using Gray-Codes), but these are out of the context of this paper.

2. Given the 128 computed symbolic expressions ST_{r+4} (as functions of a total of $4 \cdot 32 = 128$ linear variables), invert the 128×128 matrix.
   This gives a matrix which calculates the Sbox output differences of rounds $r+1$, $r+2$ and $r+3$ (and $r+4$) as functions of $\varDelta_{r+4}$ (note that we do not actually need to allocate the 32 variables for $\varDelta_{r+4}$ in order to compute this matrix). Denote by A_1 the first 96 rows of this matrix (calculating the Sbox output differences of rounds $r+1$, $r+2$ and $r+3$).

Calculating the Key From the Ciphertext and Sbox Transition Values This preprocessing algorithm computes a $128 \times (128+256)$ matrix A_2, such that given a $(128+256)$-bit vector v, comprised of the 128-bit ciphertext C, and $2 \cdot (32 \cdot 4) = 256$ input and output values of all the Sboxes of the last 4 rounds, the product $A_2 \cdot v$ gives a suggestion for the 128-bit key K.

The algorithm first symbolically describes all the $(32 \cdot 4) = 128$ Sbox output values as linear combinations of the 128 bit variables of C, the 128 bit variables of K, and the $(32 \cdot 4) = 128$ input values of all the intermediate Sboxes. This is done by computing iteratively the symbolic description of rounds $r+4, r+3, r+2, r+1$ (from the decryption side), and expressing for each round the outputs of the Sbox transitions as linear combinations of the previous variables. Finally, the algorithm performs Gaussian elimination to express the 128 variables of the key, as linear combinations of the other $128+256$ variables, giving the matrix A_2.

As the idea of this algorithm is similar to the previous algorithm (which computes A_1), its full description will be given in a future version of this paper.

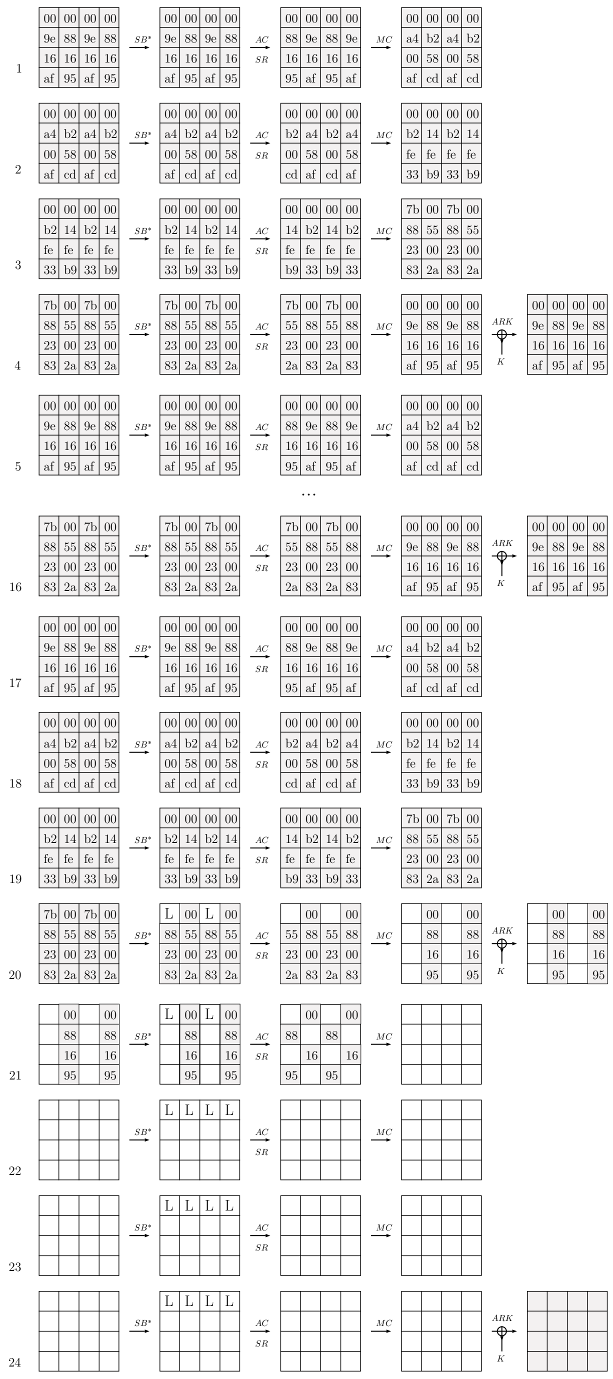


Fig. 5. The Differential Attack on Full Zorro