A shortened version of this paper will appear under the same title
in *Advances in Cryptology – CRYPTO 2003* (D. Boneh, ed.),
vol. 2729 of *Lecture Notes in Computer Science*, Springer-Verlag, 2003.

Cryptanalysis of SAFER++ [⋆]

Alex Biryukov[1 ⋆⋆], Christophe De Cannière[1 ⋆⋆⋆], and Gustaf Dellkrantz[1,2]

[1] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10,
B–3001 Heverlee, Belgium
{alex.biryukov, christophe.decanniere}@esat.kuleuven.ac.be
[2] Royal Institute of Technology,
Stockholm, Sweden
d98-gde@nada.kth.se

Abstract. This paper presents several multiset and boomerang attacks on SAFER++ up to 5.5 out of its 7 rounds. These are the best known attacks for this cipher and significantly improve the previously known results. The attacks in the paper are practical up to 4 rounds. The methods developed to attack SAFER++ can be applied to other substitution-permutation networks with incomplete diffusion.
Keywords: Cryptanalysis, Block ciphers, SAFER, Boomerang attack, Multiset attack.

1 Introduction

The 128-bit block cipher SAFER++ [9] is a 7-round substitution-permutation network (SPN), with a 128-bit key (the 256-bit key version[3] has 10 rounds). SAFER++ was submitted to the European pre-standardization project NESSIE [14] and was among the primitives selected for the second phase of this project.

SAFER [7] was introduced by Massey in 1993, and was intensively analyzed since then [4, 6, 8, 11, 13]. This resulted in a series of tweaks which lead to several ciphers in the family: SAFER-K (the original cipher), SAFER-SK (key schedule tweak), SAFER+ (key schedule and mixing transform tweak, increased number of rounds, AES candidate), SAFER++ (faster mixing tweak, key schedule tweak,

⋆ The work described in this paper has been supported in part by the Commission of the European Communities through the IST Programme under Contract IST-1999-12324 and by the Concerted Research Action (GOA) Mefisto-666.
⋆⋆ F.W.O. Researcher, sponsored by the Fund for Scientific Research – Flanders.
⋆⋆⋆ F.W.O. Research Assistant, sponsored by the Fund for Scientific Research – Flanders
[3] A legacy 64-bit block version was also proposed by the designers but is not studied in this paper.

fewer rounds due to more complex mixing). All these ciphers have common S-boxes derived from exponentiation and discrete logarithm functions. They share the Pseudo-Hadamard-like mixing transforms (PHT), although these are constructed in different ways in the different versions. The ciphers in the family also share the idea of performing key-mixing with two non-commutative operations.

The inventors claim that SAFER++ offers "further substantial improvement over SAFER+" [9]. The main feature is a new 4-point PHT transform in place of the 2-point PHT transform that was used previously in the SAFER family. The authors claim that "all 5-round characteristics have probabilities that are significantly smaller than 2^{-128}" and that SAFER++ is secure against differential cryptanalysis [1] after 5 rounds and against linear cryptanalysis [10] after 2.5 rounds.

The best previous attack on SAFER++ is linear cryptanalysis [12], which can break 3 rounds of SAFER++ (with 128-bit keys) with 2^{81} known plaintexts and 2^{101} steps for a fraction 2^{-13} of keys. For 256-bit keys the attack can break the 3.5-round cipher with 2^{81} known plaintexts and 2^{176} steps for a fraction 2^{-13} of keys.

In this paper we study only the 128-bit key version of SAFER++, since we would like to make our attacks as practical as possible. We design several very efficient multiset attacks on SAFER++ following the methodology of the structural attack on SASAS [2] and inspired by the collision attacks on RIJNDAEL [3]. These multiset attacks can break up to 4.5 rounds of SAFER++ with 2^{48} chosen plaintexts and 2^{94} steps, which is much faster than exhaustive search. Attacking 3 rounds is practical and was tested with an actual implementation running in milliseconds on a PC.

In the second half of the paper we show how to apply a cryptanalytic technique called the boomerang attack [15] to SAFER++. We start from ideas which are applicable to arbitrary SPNs with incomplete diffusion (such as RIJNDAEL, SAFER++ or SERPENT) and then extend our results using special properties of the SAFER S-boxes. The attacks thus obtained are more efficient then those we found via the multiset techniques, are practical up to 4 rounds and were confirmed experimentally on a mini-version of the cipher.

The average data complexity of the 5 round attack is 2^{78} chosen plaintexts/adaptive chosen ciphertexts with the same time complexity, most of which is spent encrypting the data. The attack completely recovers the 128-bit secret key of the cipher and can be extended to 5.5 rounds by guessing 30 bits of the secret key. See Table 1 for a summary of results presented in this paper and their comparison with the best previous attack.

This paper is organized as follows: Section 2 provides a short description of SAFER++ and Section 3 shows some interesting properties of the components. In Sections 4 and 5 we design our multiset attacks on SAFER++. Section 6 describes our application of boomerang techniques to SAFER++ reduced to 5 rounds and shows how to use the middle-round S-box trick to obtain even better results (a similar attack on 4 rounds is described in Appendix A). Finally, Section 7 concludes the paper.

Table 1. Comparison of our results with the best previous attack on SAFER++.

Attack	Key size	Rounds	Data[a]	Type[b]	Workload[c]	Memory[a]
Our Multiset attack	128	3 of 7	2^{16}	CC	2^{16}	2^{4}
Our Multiset attack	128	4 of 7	2^{48}	CP	2^{70}	2^{48}
Our Multiset attack	128	4.5 of 7	2^{48}	CP	2^{94}	2^{48}
Our Boomerang attack	128	4 of 7	2^{41}	CP/ACC	2^{41}	2^{40}
Our Boomerang attack	128	5 of 7	2^{78}	CP/ACC	2^{78}	2^{48}
Our Boomerang attack	128	5.5 of 7	2^{108}	CP/ACC	2^{108}	2^{48}
Linear attack[d] [12]	128	3 of 7	2^{81}	KP	2^{101}	2^{81}

[a] Expressed in number of blocks.
[b] KP – Known Plaintext, CP – Chosen Plaintext, ACC – Adaptive Chosen Ciphertext.
[c] Expressed in equivalent number of encryptions.
[d] Works for one in 2^{13} keys.

2 Description of Safer++

This section contains a short description of SAFER++. For more details, see [9]. In this paper, eXclusive OR (XOR) will be denoted by $\oplus$, addition modulo 256 by $\boxplus$ and subtraction modulo 256 by $\boxminus$. The notion of difference used is subtraction modulo 256. Throughout this paper we will number bytes and S-boxes from left to right, starting from 0.

SAFER++ is an iterated product cipher in which every round consists of an upper key layer, a nonlinear layer, a lower key layer and a linear transformation. Fig. 1 shows the structure of one SAFER++ round. After the final round there is an output transformation that is similar to the upper key layer. The upper and lower key layers together with the nonlinear layer make up the *keyed nonlinear layer*, denoted by S. The linear layer is denoted by A.

2.1 The Keyed Nonlinear Layer

The upper key layer combines a 16 byte subkey with the 16 byte block. Bytes 0, 3, 4, 7, 8, 11, 12 and 15 of the subkey are XORed to the corresponding bytes of the block and bytes 1, 2, 5, 6, 9, 10, 13 and 14 are combined using addition modulo 256.

The nonlinear layer is based on two 8-to-8-bit functions, X and L defined as

$$\begin{aligned} \mathrm{X}(a) &= (45^a \bmod 257) \bmod 256\,, \\ \mathrm{L}(a) &= \log_{45}(a) \bmod 257\,, \end{aligned}$$

with the special case that $\mathrm{L}(0) = 128$, making X and L mutually inverse. In the nonlinear layer, bytes 0, 3, 4, 7, 8, 11, 12 and 15 are sent through the function X, and L is applied to bytes 1, 2, 5, 6, 9, 10, 13 and 14.

The lower key layer applies a 16 byte subkey to the 16 byte block using addition modulo 256 for bytes 0, 3, 4, 7, 8, 11, 12 and 15 and XOR for bytes 1, 2, 5, 6, 9, 10, 13 and 14.

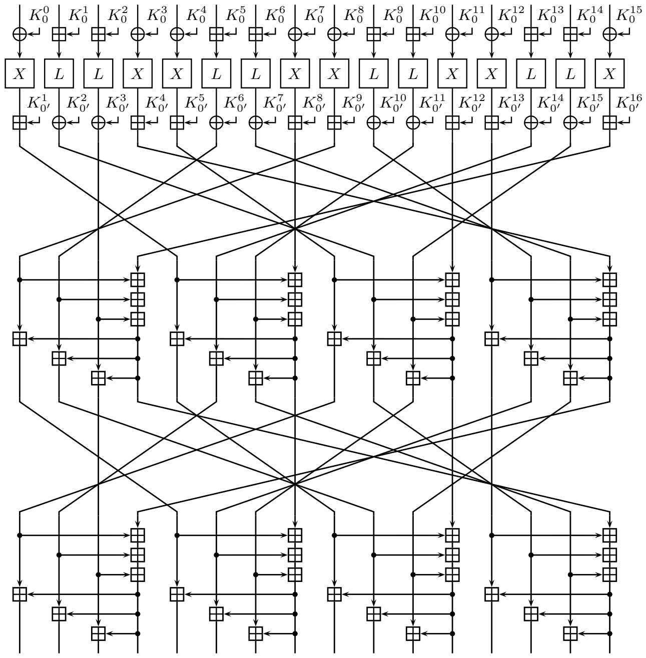

Fig. 1. One round of SAFER++.

2.2 The Linear Layer

The linear transformation of SAFER++ is built from a 4-point Pseudo Hadamard Transform (4-PHT) and a coordinate permutation. The 4-PHT can be implemented with six modular additions.

The linear layer first reorders the input bytes and then applies the 4-PHT to groups of four bytes. The output of the linear layer is obtained after iterating this operation twice.

The linear layer and its inverse can be represented by the matrices A and A^{-1}. Since the linear layer consists of two iterations of one linear function the matrix A can be written as the square of a matrix $\sqrt{A}$. These matrices are shown in Appendix B.

2.3 The Key schedule

The key schedule expands the 128 or 256-bit master key into the required number of subkeys. It consists of two similar parts differing only in the way the master key is used to fill the registers. The first part generates the subkeys for the upper key layer and the output transform and the second part generates subkeys for the lower key layer.

It can be noted that the key schedule provides no interaction between bytes of the key and furthermore, there is a big overlap between the key bytes used in different rounds. Therefore, we will not number the bytes of the subkeys according to the order in the subkeys, but according to which master key byte they depend on.

3 Properties of the Components

In this section we show some interesting properties of the components of SAFER++ which will be used later in our analysis.

3.1 Diffusion in the Linear Layer

In [8], the designers show that the choice of the components used in the linear layer provides "optimum transform diffusion" without sacrificing efficiency. In order to measure this diffusion, the authors compute the minimal number of output bytes that are affected by a change in a single input byte. In the case of SAFER++, for example, the linear layer guarantees that a single byte difference at the input of the layer will cause at least ten output bytes to be different.

While the "optimum transform diffusion" defined in this way is certainly a desirable property, it potentially allows some low-weight differentials that might still be useful for an attacker. For example, if two input bytes are changed simultaneously in SAFER++, the number of affected output bytes after the linear layer can be reduced to only three. The adversary might also consider to attack the layer in decryption direction, in which case single byte differences are only guaranteed to propagate to at least five bytes. Neither of these cases is captured by the diffusion criterion used in [8].

3.2 Symmetry of the Linear Layer

Due to the symmetry of the 4-PHT and the coordinate permutation used, there is a four byte symmetry in the linear layer. If the input difference to the linear layer is of the form

$$(a, b, c, d, a, b, c, d, a, b, c, d, a, b, c, d)$$

for any 8-bit values a, b, c, and d the output difference will be of the form

$$(x, y, z, t, x, y, z, t, x, y, z, t, x, y, z, t)$$

The nonlinear layer is symmetric in the same way and were it not for the subkeys, the property would hold for the whole SAFER++ cipher, with an arbitrary number of rounds.

A special illustration of this property are the two eigenvectors of the linear transformation corresponding to the eigenvalue 1:

$$(0, 1, -1, 0, 0, 1, -1, 0, 0, 1, -1, 0, 0, 1, -1, 0)$$
$$(1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0)$$

These vectors and all linear combinations of them are fixed points of the linear transform.

3.3 Properties of the S-boxes

The S-boxes of SAFER++ are constructed using exponentiations and logarithms as described previously. This provides us with some interesting mathematical properties. The following expressions hold for X and L:

$$\begin{aligned} \mathrm{X}(a) + \mathrm{X}(a \boxplus 128) &= (45^a \bmod 257) + (45^{a+128} \bmod 257) = 257 \\ &\equiv \quad 1 \quad \bmod 256\,, \\ \mathrm{L}(a) - \mathrm{L}(1 \boxminus a) &\equiv 128 \quad \bmod 256\,. \end{aligned}$$

This is a very useful property that will be exploited in the boomerang attacks in Section 6. Though we discovered this property independently, it was known to Knudsen [6].

4 Multiset Attack on 3 Rounds using Collisions

Multiset attacks are attacks where the adversary studies the propagation of multisets through the cipher, instead of actual values or differences as in other attacks. By multisets we mean sets where values can appear more than once. The multiplicity of each value is one important characteristic of a multiset.

In this and the following section we present two multiset attacks on 3 and 4 rounds of SAFER++. Both attacks exploit the asymmetry of the linear layer, but in different ways. The first attack considers the cipher in decryption direction and relies on the fact that the weaker diffusion in backward direction allows a stronger distinguisher. The second approach, described in Section 5, is built on a distinguisher in forward direction which, though weaker than the previous one, can more easily be extended with rounds at the top and the bottom. All multiset attacks presented in this paper are independent of the choice of the S-boxes.

4.1 A Two-round Distinguisher in Decryption Direction

As mentioned earlier, changes propagate relatively slowly through the linear layer in decryption direction. Most byte positions at the input of a decryption round affect only 6 bytes at the output, and conversely, most byte positions at the output are affected by only 6 bytes at the input. After one more round, this property is destroyed, i.e., a change in any input byte can induce changes in any output byte after two rounds (complete diffusion on byte level). The number of different paths through which the changes propagate is still small, however, and this property allows us to build an efficient distinguisher.

The structure of a 2-round distinguisher is shown in Fig. 2. First, a multiset of 2^{16} texts is constructed, constant in all bytes except in byte 4 and 6, in which it takes all possible values. After one decryption round, these texts are still constant in 8 positions, due to the weak diffusion properties of the linear layer. We now focus on byte 13 at the top of the distinguisher. This byte is affected by 6 bytes of the preceding round, 5 of which are constant. This implies that

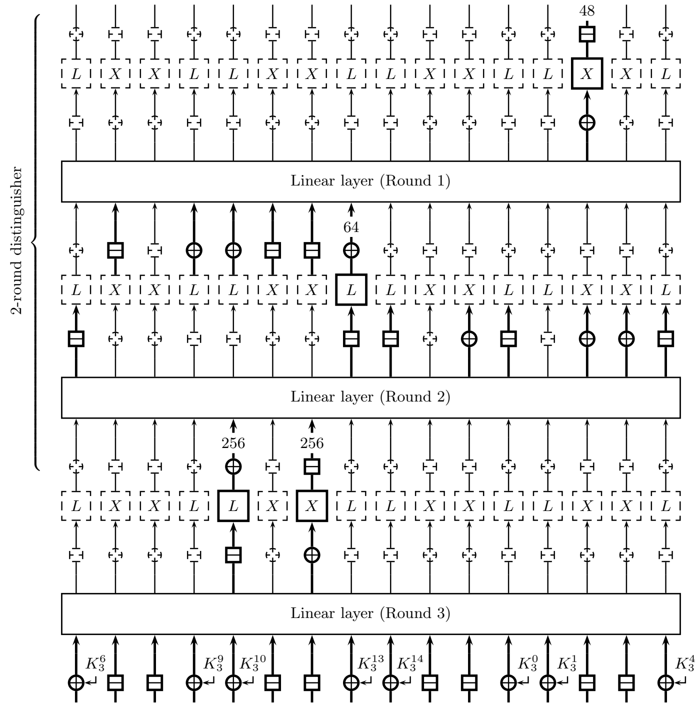

Fig. 2. A 3-round attack.

the changes in the two bytes at the input essentially propagate through a single path.

As a closer look reveals, this path has some additional properties: in Round 2, the two input bytes are first summed and then multiplied by 4 before they reach S-box 7. The byte at this position is then again multiplied by 4 in Round 1. The effect of these multiplications (modulo 256) is that the two most significant bits are lost, reducing the number of different values that can be observed at the top to 64. This is extremely unlikely to happen if the cipher were a random permutation, in which case all 256 possible values would almost certainly be observed. Moreover, due to the interaction with the special S-boxes of SAFER++, the number of different values is exactly 48.

4.2 Adding a Round at the Bottom

The distinguisher above is very strong, but it is hard to use it as such when a round is added at the bottom. Causing the multiset described above at the input of the distinguisher (see Fig. 2) would require half of the key bytes (those that are XORed) to be guessed in the final key addition layer.

In this paper we use a better approach and consider small multisets of 2^4 texts that are constant except in the two most significant bits of the fifth and the seventh byte. In order to cause such multisets from the bottom, we only need to guess the second most significant bit of the 8 key bytes that are XORed in the final key addition layer and generate sets of 2^4 ciphertexts of the form

$$(x \cdot A \boxplus y \cdot B \boxplus C) \oplus K \tag{1}$$

with $(x, y) \in \{0, 64, 128, 192\}^2$, C any fixed 128-bit word, and

$$\begin{aligned} A &= (4,2,2,2,1,1,2,1,1,1,1,1,1,2,1,1)\,, \\ B &= (1,1,1,1,1,2,1,1,2,2,4,2,2,1,1,1)\,, \\ K &= (K_3^6, 0, 0, K_3^9, K_3^{10}, 0, 0, K_3^{13}, K_3^{14}, 0, 0, K_3^0, K_3^1, 0, 0, K_3^4)\,. \end{aligned}$$

Note that only the second most significant bits of the K_3^i are relevant (because of carries) and that the effect of the other bits can be absorbed in the constant C.

Now the question arises whether we can still distinguish the set of 16 values obtained in byte 13 of the plaintexts from a random set. An interesting characteristic that can be measured is the number of collisions in the sets, should they appear at all. If this number depends in a significant way on whether the key bits were correctly guessed or not, then this would allow us to recover the key. This question is easily answered by deriving a rough estimation of the number of collisions.

First, we deduce the expected number of collisions μ_0 in case the guess was correct. Assuming that all 48 possible values are equally likely, we obtain

$$\mu_0 = \binom{16}{2} \cdot \frac{1}{48} \approx 2.50 \qquad \text{and} \qquad \sigma_0 = \sqrt{\binom{16}{2} \cdot \frac{1}{48}\left(1 - \frac{1}{48}\right)} \approx 1.56\,,$$

with σ_0 the expected deviation. In order to estimate the number of collisions given a wrong guess, one could be tempted to assume that the sets at the output of the distinguisher are random. This is not the case however.[4] One can easily see, for example, that whether K_3^6 is correctly guessed or not matters only for half of the (x, y) pairs, i.e., when the second most significant bit of $4 \cdot x \boxplus y$ is set. In all these (and only these) cases an incorrect carry bit will appear in the most significant bit. If we absorb this wrong bit in a new constant C', we obtain two subsets of 8 texts, both of which still satisfy (1), but with two different fixed values C and C'. Hence we find

$$\begin{aligned} \mu_1 &= 2 \cdot \binom{8}{2} \cdot \frac{1}{48} + 8^2 \cdot \frac{1}{256} && \approx 1.42\,, \\ \sigma_1 &= \sqrt{2 \cdot \binom{8}{2} \cdot \frac{1}{48}\left(1 - \frac{1}{48}\right) + 8^2 \cdot \frac{1}{256}\left(1 - \frac{1}{256}\right)} && \approx 1.19\,. \end{aligned}$$

The value of μ_1 is indeed considerably higher than what we would expect for a random set (about 0.47 collisions). Exactly the same situation occurs for 17

[4] Note that this property is useful if one does not care about recovering the key, but just needs a 3-round distinguisher.

other combinations of wrong key bits, and similar, but less pronounced effects can be expected for other guesses.

The result above can now be used to predict the complexity and success probability of our 3-round attack. The attack consists in running through all 2^8 possible partial key guesses and accumulating the total number of collisions observed after decrypting α sets of 16 texts. The maximum number of collisions is then assumed to correspond to the correct key. Taking this into consideration, we obtain the estimations below:

$$\text{Time and data complexity} \approx 2^8 \cdot 16 \cdot \alpha \,,$$
$$\text{Success probability} \approx 1 - 17 \cdot \Phi\left(\sqrt{\alpha} \cdot \frac{\mu_0 - \mu_1}{\sqrt{\sigma_0^2 + \sigma_1^2}}\right) .$$

Evaluating these expressions for $\alpha = 16$, we find a complexity of 2^{16} and a corresponding success probability of 77%. Similarly, for $\alpha = 32$ we get a complexity of 2^{17} and an expected probability as high as 98%. In order to verify these results, we performed a series of simulations and found slightly lower success probabilities: 70% and 89% for $\alpha = 16$ and $\alpha = 32$ respectively. This difference is due to the fact that our estimation only considers wrong guesses that have a high probability of producing many collisions.

5 Multiset Attack on 4 Rounds using Structural Approach

The chosen ciphertext attack presented in the previous section is particularly efficient on 3 rounds, but extending it to 4 rounds turns out to be rather difficult. Either we could try to add a round at the bottom, which would require us to cause small bit changes after crossing a decryption round, or we could add a round at the top and try to recover the value of a particular byte from the output of this round. Both cases, however, are hindered by the asymmetry of the linear layer and this motivates the search for a distinguisher in forward direction.

Before describing the distinguisher we introduce some convenient notations to represent different types of multisets:

C – **Constant multiset.** A set containing a single value, repeated multiple times.
P – **Permutation multiset.** A set containing all possible values once, in an arbitrary order.
E – **Even multiset.** A set containing different values, each occurring an even number of times.
B – **Balanced multiset.** A set containing elements with arbitrary values, but such that their sum (modulo 256) is zero. We will write "B_0", if this property only holds for the least significant bit. Note that multisets satisfying the conditions C, P or E satisfy condition B_0 as well.

5.1 A Two-round Distinguisher in Encryption Direction

Fig. 3 is a graphical representation of the 2-round distinguisher using the notations above. The input to the distinguisher consists of a permutation multiset (P) at position 12, and constant multisets (C) elsewhere. This pattern is preserved after the first layer of S-boxes. It then crosses the linear layer, where it is transformed into a series of P and E multisets. Again, these properties remain unchanged after applying the second layer of S-boxes. Finally, we note that all multisets at this point satisfy condition B_0, and this property is preserved both by the second linear layer and the final key addition. The sixteen B_0-multisets at the output provide us with a 16-bit condition and allows the two round cipher to be distinguished from a random permutation.

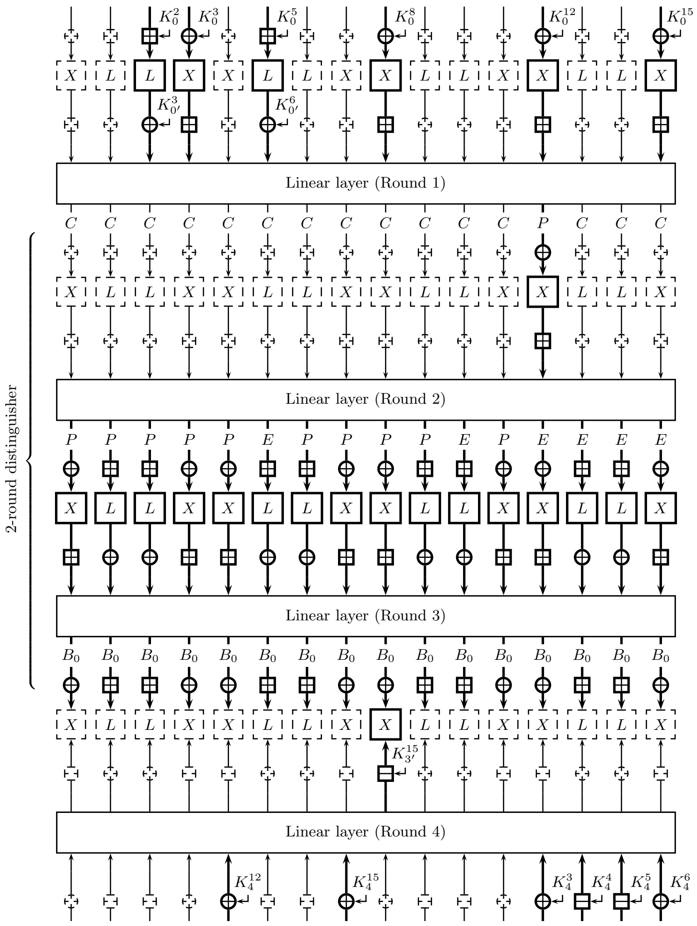

Fig. 3. A 4-round multiset attack.

5.2 Adding a Round at the Top

In order to add a round at the top, we must be capable of keeping all bytes constant after one encryption round, except for byte 12. This would have been very hard in decryption direction, but is fortunately relatively easy in this case. Due to the fact that the diffusion is incomplete in backward direction, we only need to cross the nonlinear layer in 6 active bytes (see Fig. 3). Moreover, we do not care about constants added to byte 12. This implies that 6 key bytes in the upper key layer (K_0^2, K_0^3, K_0^5, K_0^8, K_0^{12}, and K_0^{15}) and 2 XORed key bytes in the lower key layer ($K_{0'}^3$ and $K_{0'}^6$) need to be guessed. As K_0^3 and $K_{0'}^3$ are easily derived from each other, guessing 7 bytes will suffice.

In order to distinguish the wrong guesses from the correct one, 4 different multisets need to be examined for each key (yielding a 8-byte condition). Since only six bytes are active at the top, all multisets can be constructed from a pool of 2^{48} plaintexts. The time complexity of this 3-round attack is about $4 \cdot 2^{7 \cdot 8} = 2^{58}$ steps.

5.3 Adding a Round at the Bottom

The complexity derived above is a bit disappointing compared to the results in the previous section, but the good news is that adding an extra round at the bottom comes almost for free. This time, we will test a single byte at the output of the distinguisher (byte 8) and exploit the fact that this byte is only affected by 6 bytes at the output of the fourth round (see Fig. 3). The additional obstacles are 7 key bytes which need to be guessed. However, due to the key schedule and the special choice for the position of the active byte at the input of the distinguisher (byte 12), 6 of these key bytes can directly be determined from the keys guessed at the top. As a consequence, the total number of key bytes to guess is only increased by 1. Taking into account that each multiset provides a 1-bit condition in this case, we obtain a time complexity of $8 \cdot 8 \cdot 2^{8 \cdot 8} = 2^{70}$ steps.

The 4-round attack described above can easily be extended by another half round at the bottom (i.e., an S-box layer followed by a key addition). This would in principle require 6 more key bytes to be guessed, but again, 3 of them depend on previous guesses. Accordingly, the complexity increases to about $11 \cdot 8 \cdot 2^{11 \cdot 8} \approx 2^{94.5}$ steps.

6 Boomerang attacks on Safer++

In this section we describe an attack on Safer++ reduced to 5 rounds and then extend it to an attack on 5.5 rounds. The attack is a boomerang attack [15] using a combination of truncated differentials [5] and conventional differentials [1].

Contrary to conventional differentials which require full knowledge of the input difference and predict the full output difference, truncated (or wildcard) differentials restrict only parts of the input difference and may predict only parts of the output difference. It is thus natural to consider truncated differentials for ciphers which use operations on small sub-blocks (for example, bytes or nibbles).

Two natural ways of placing partial restrictions on differences are: to fix the difference in certain sub-blocks to constant values (a popular choice is zero) while allowing arbitrary differences in the other sub-blocks; or to place constraints on differences without restricting them to specific values. For example, one can consider differences of the form $(x, 0, -x, 2x, 0, -y, y, 4y)$, where the values x, y are not restricted and thus give *degrees of freedom*.

Truncated differentials have several distinct properties which make them different from conventional differentials. First of all, the probability of truncated differentials is usually not the same in forward and in backward direction. Secondly, due to the "wildcard" nature of truncated differentials plaintexts for truncated differentials may be efficiently packed into pools in which a large fraction of pairs satisfies the truncated difference. Such a *pool effect* often significantly reduces the data requirements for detecting truncated differentials. The data requirements of the truncated differential attack may be smaller than the inverse of the differential probability (which is usually a good measure for the data complexity of a conventional differential attack), and thus truncated differential attacks may still work in cases where the probability is lower than 2^{-n} where n is the block size. Finally, due to the large number of pairs in the pools and due to the partial prediction of the output difference for the *right pair*, filtration of wrong pairs (those pairs that have the predicted partial output difference, but do not follow the partial difference propagation expected by the truncated differential) becomes a crucial issue for truncated differential attacks.

The idea of the boomerang attack is to find good conventional (or truncated) differentials that cover half of the cipher but can not necessarily be concatenated into a single differential covering the whole cipher. The attack starts with a pair of plaintexts P and P' with a difference Δ which goes to difference Δ^* through the upper half of the cipher. The attacker obtains the corresponding ciphertexts C and C', applies the difference ∇ to obtain ciphertexts $D = C + \nabla$ and $D' = C' + \nabla$ and decrypts them to plaintexts Q and Q'. The choice of ∇ is such that the difference propagates to the difference ∇^* in the decryption direction through the lower half of the cipher. For the *right quartet* of texts, difference Δ^* is created in the middle of the cipher between partial decryptions of D and D' which propagates to the difference Δ in the plaintexts Q and Q'. This can be detected by the attacker.

Moreover, working with quartets (pairs of pairs) provides boomerang attacks with additional filtration power. If one partially guesses the keys of the top round one has two pairs of the quartet to check whether the uncovered partial differences follow the propagation pattern, specified by the differential. This effectively doubles the attacker's filtration power.

6.1 A Boomerang Distinguisher for 4.5 Rounds

In this part we build a distinguisher for 4.5 rounds of SAFER++. As mentioned before, the affine layer is denoted by A and the keyed nonlinear layer by S. Using this notation, the distinguisher will have the structure $[ASA - S - ASASA]$ (see Fig. 5, in which layers of rectangular boxes correspond to S-layers). The top part $[ASA-]$ will be covered top-down and the bottom part $[-ASASA]$ in the bottom-up direction. The part $[-S-]$ in the middle will come for free using the middle S-box trick.

Pairs of plaintexts (P, P'), that have difference $(0, x, 0, 0, x, x, 0, 0, 0, 0, 0, -4x, 0, 0, x, -x)$ are used from the top. This difference propagates to a single active S-box after one linear layer and is then diffused to up to 16 active bytes by the next linear layer.

We obtain the two ciphertexts C, C', modify them into $D = C \boxplus \nabla$, $D' = C' \boxplus \nabla$ and decrypt them to two new plaintexts Q and Q'. The difference ∇ is $(0, 0, 80_x, 0, 80_x, 80_x, 0, 0, 0, 0, 0, 0, 80_x, 0, 0, 0)$ which causes difference $(0, 0, 0, 0, 0, x, -x, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ with approximate probability 2^{-7} after one decryption round. The probability is 2^{-7} since the input difference 80_x can only cause odd output differences through X. The two active bytes then cause bytes $3, 9, 11$ and 14 to be active after the next linear layer.

In a standard boomerang attack, we would now hope that the pairs (C, D) and (C', D') both have the same difference in the middle round, which would cause the difference of the pair (D, D') in the middle round to be the same as that for (P, P'). For the right quartet we would then have the same difference going back in the pair (D, D') as going forward in (P, P'), thus making (Q, Q') active in the same bytes as (P, P'). This can be detected by the attacker and used to recover parts of the key.

This version of the attack is reasonably efficient against four rounds of SAFER++ and can be applied to any substitution-permutation network with incomplete diffusion. However, the probability of the event used in the middle, where the parts of the boomerang meet, depends heavily on the number of active bytes and is often low. For the SAFER-family of ciphers we can do better by using special properties of the S-boxes.

6.2 The Middle Round S-box Trick

Using the relation $\mathrm{X}(a) \boxplus \mathrm{X}(a \boxplus 80_x) = 1$, which holds with probability 1 for the exponentiation-based S-boxes (see Section 3.3), we can make the boomerang travel for free through the middle S-box layer.

Consider the middle S-box layer where the top and bottom part of the boomerang meet. Suppose that the difference coming from above and entering an X-box, is 80_x, i.e., a pair of values a and $a \boxplus 80_x$. After the X-box the values will always be of the form b and $1 \boxminus b$. Assume that on the two other faces of the boomerang we also have difference 80_x coming from below (see Fig. 4). Then the fourth face must have values $b \boxplus 80_x$ and $1 \boxminus b \boxplus 80_x$ on the lower side of the X-box. However, these values again sum to 1, since $80_x \boxplus 80_x = 0$ (actually, any pair of values that sum to 0 would work instead of 80_x here). As a consequence, we will observe values of the form c and $c \boxplus 80_x$ at the input of the X-box on the fourth face. See Fig. 4 for an illustration of this effect. The same reasoning holds for the L-box since it is the inverse of X and the same differences are coming from the top and the bottom.

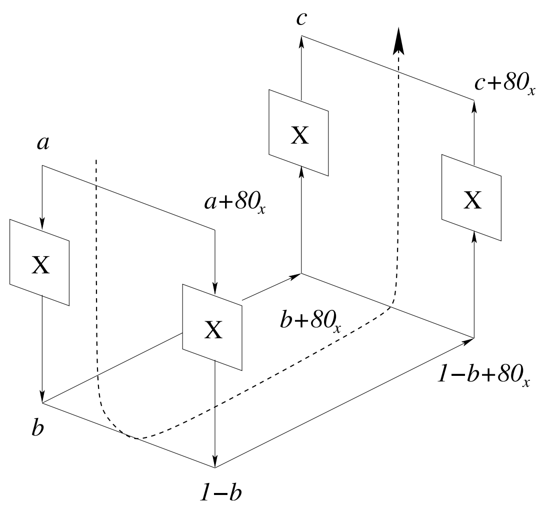

Fig. 4. The free pass of the boomerang through the middle Safer++ S-boxes.

This shows that if we manage to produce texts with difference 80_x coming from both the top and the bottom in the boomerang, the boomerang travels through the middle S-box layer for free due to the special properties of the S-boxes.

6.3 Breaking 5 and 5.5 Rounds

We can break 5 rounds of Safer++ with the distinguisher described in the previous sections. The truncated differentials used are shown in Fig. 5. From the top we use a difference with six active bytes that propagates to one active byte after one round with probability 2^{-40}. This one-byte difference then causes a difference of 80_x in bytes 0, 1, 2, 3, 8, 9, 11, 13, 14 and 15 after an additional round with probability 2^{-8}. All other bytes have zero-difference. This completes the upper part of the boomerang.

From the bottom we start with changes in the most significant bits of four bytes, which cause two active bytes after one decryption round. This then propagates to four active bytes with probability 2^{-7} and further to difference 80_x in all bytes except bytes 2, 4, 9 and 12 with probability $2^{-30.4}$. The total probability of the lower part of the boomerang is thus $\left(2^{-7} \cdot 2^{-30.4}\right)^2$, since we need the differential to hold in two pairs.

The top part of the boomerang in the decryption direction propagates with probability 1, due to the middle S-box trick. The total probability of the boomerang is thus $2^{-40} \cdot 2^{-8} \cdot \left(2^{-7} \cdot 2^{-30.4}\right)^2 = 2^{-122.8}$

The procedure for attacking 5 rounds is as follows:

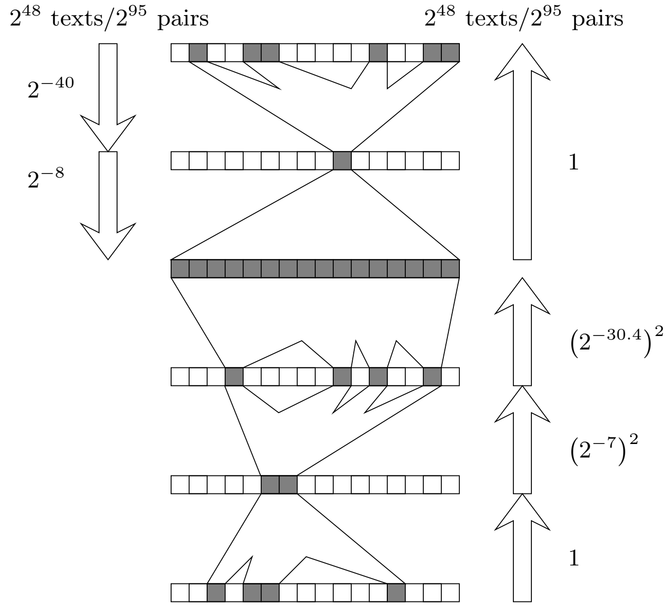

Fig. 5. The boomerang quartet for SAFER++ reduced to 5.5 rounds.

1. Prepare a pool of 2^{48} plaintexts P_i, $i = 0, \ldots, 2^{48} - 1$ that have all possible values in bytes 1, 4, 5, 11, 14 and 15 and are constant in the other bytes. Encrypt the pool to get a pool of 2^{48} ciphertexts C_i.
2. Create a new pool of ciphertexts D_i from the pool C_i by changing the most significant bits in bytes $2, 4, 5$ and 12. Decrypt the pool D_i to obtain a pool Q_i of 2^{48} plaintexts.
3. Sort the pool Q_i on the bytes that correspond to the constant bytes in the pool P_i and pick the pairs Q_j, Q_k that have zero difference in those ten bytes.
4. For each of the possibly good quartets P_j,P_k,Q_j,Q_k, guess the 3 key bytes K_0^4, K_0^{11} and K_0^{15}, do a partial encryption and check that the difference in the 3 bytes after the first nonlinear layer is of the form $(x, -4x, -x)$ with x odd, both in the P-pair and in the Q-pair. Note that we do not need to guess the key bytes added at the bottom of this layer, as they do not influence the subtractive difference here. Keep the quartets that have the right difference, together with the key bytes that they suggest.
5. Guess the key byte K_0^{14}, do a partial encryption and check that the difference after XORing the key $K_{0'}^{15}$ (which we know from the previous step) at the bottom of the nonlinear layer is consistent with the difference found in Step 4. If no quartets survived, go to Step 1.
6. Guess the key bytes K_0^1 and $K_{0'}^2$, do a partial encryption and check that the difference after the first keyed nonlinear layer is the right one. Repeat this for the key bytes K_0^5 and $K_{0'}^6$.

7. Keep collecting suggestions for the 8 key bytes until one appears twice. For this suggestion, do an exhaustive search for the 8 remaining key bytes using trial encryption. If no consistent key is found, go to Step 1.

We now analyze the complexity of this attack. From the pool of 2^{48} plaintexts created in Step 1 we get approximately 2^{95} pairs. Since the probability of the boomerang is $2^{-122.8}$, the probability is approximately $2^{-27.8}$ that a pool contains a boomerang.

After Step 3 we expect to have about 2^{15} wrong quartets left since we have an 80-bit condition on 2^{95} pairs. Step 4 reduces the number of wrong quartets to 2^5, because after guessing 3 bytes of the key, we obtain a 17-bit restriction on each side of the boomerang, resulting in a 34-bit condition. Similarly, after Step 5, only about 2^{-3} quartets remain per pool. On the average, Step 6 will suggest 1 value for the 4 guessed key bytes per remaining quartet.

In order for the right key to be suggested twice, we need two boomerangs in Step 7. This will occur after having analyzed about 2^{29} pools on average. During this process, 2^{26} wrong keys will be suggested as well, but the chance that one of these 64-bit values is suggested twice is small (the probability is about $2^{51}/2^{64} = 2^{-13}$). This implies that the exhaustive search in Step 7 only needs to be performed for a single suggestion on average.

Since each of the 2^{29} pools required by the attack consist of 2^{48} chosen plaintexts and 2^{48} adaptively chosen ciphertexts, we obtain a data complexity of 2^{78}. Just collecting these texts will be the most time-consuming part of the attack, and the time complexity of the attack is therefore expected to be 2^{78}. These figures reflect the average case, but the complexities will exceed 2^{79} in only 5% of the cases.

The attack on 5 rounds can be extended to 5.5 rounds by guessing 30 bits of the key in positions $2, 4, 5$ and 12 at the end of the added half round. This increases the data and time complexities to 2^{108}. Note that 2 key bits have been saved by using the special properties of the L-box.

7 Conclusions

In this paper we have applied novel multiset attack techniques to round-reduced SAFER++ inspired by the recent structural analysis of the SASAS scheme and partial-function collision techniques. These multiset attacks are very efficient up to 4.5 rounds and practical up to 3 rounds. This significantly improves the previously known results.

In the second half of the paper we applied boomerang attacks to SAFER++ which allow for even more efficient attacks because they exploit special properties of the exponentiation and logarithmic S-boxes and their interaction with the PHT-mixing layer. We presented an attack on 5.5 out of 7 rounds of SAFER++ requiring 2^{108} data blocks and steps of analysis. A 4-round variant of this boomerang attack is practical and was tested on a 64-bit mini-version of the cipher. See Table 1 for a summary of results presented in this paper and their comparison with previously known attacks.

Finally, note that the methods developed in the second half of this paper can be applied to arbitrary SPNs with incomplete diffusion, with the exception of the middle round trick which exploits special properties of the Safer++ S-boxes.

References

[1] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[2] A. Biryukov and A. Shamir, "Structural cryptanalysis of SASAS," in *Advances in Cryptology – EUROCRYPT 2001* (B. Pfitzmann, ed.), vol. 2045 of *Lecture Notes in Computer Science*, pp. 394–405, Springer-Verlag, 2001.

[3] H. Gilbert and M. Minier, "A collision attack on seven rounds of Rijndael," in *Proceedings of the Third AES Candidate Conference*, pp. 230–241, National Institute of Standards and Technology, Apr. 2000.

[4] J. Kelsey, B. Schneier, and D. Wagner, "Key-schedule cryptanalysis of 3-WAY, IDEA, G-DES, RC4, SAFER, and Triple-DES," in *Advances in Cryptology – CRYPTO'96* (N. Koblitz, ed.), vol. 1109 of *Lecture Notes in Computer Science*, pp. 237–251, Springer-Verlag, 1996.

[5] L. R. Knudsen, "Truncated and higher order differentials," in *Fast Software Encryption, FSE'94* (B. Preneel, ed.), vol. 1008 of *Lecture Notes in Computer Science*, pp. 196–211, Springer-Verlag, 1995.

[6] L. R. Knudsen, "A detailed analysis of SAFER K," *Journal of Cryptology*, vol. 13, no. 4, pp. 417–436, 2000.

[7] J. L. Massey, "SAFER K-64: A byte-oriented block-ciphering algorithm," in *Fast Software Encryption, FSE'93* (R. J. Anderson, ed.), vol. 809 of *Lecture Notes in Computer Science*, pp. 1–17, Springer-Verlag, 1994.

[8] J. L. Massey, "On the optimality of SAFER+ diffusion," in *Proceedings of the Second AES Candidate Conference*, National Institute of Standards and Technology, Mar. 1999.

[9] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian, "Nomination of SAFER++ as candidate algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE)." Primitive submitted to NESSIE by Cylink Corp., Sept. 2000.

[10] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Advances in Cryptology – EUROCRYPT'93* (T. Helleseth, ed.), vol. 765 of *Lecture Notes in Computer Science*, pp. 386–397, Springer-Verlag, 1993.

[11] S. Murphy, "An analysis of SAFER," *Journal of Cryptology*, vol. 11, no. 4, pp. 235–251, 1998.

[12] J. Nakahara Jr, *Cryptanalysis and Design of Block Ciphers*. PhD thesis, Katholieke Universiteit Leuven, June 2003.

[13] J. Nakahara Jr, B. Preneel, and J. Vandewalle, "Linear cryptanalysis of reduced-round versions of the SAFER block cipher family," in *Fast Software Encryption, FSE 2000* (B. Schneier, ed.), vol. 1978 of *Lecture Notes in Computer Science*, pp. 244–261, Springer-Verlag, 2001.

[14] NESSIE Project – New European Schemes for Signatures, Integrity and Encryption. `http://cryptonessie.org`.

[15] D. Wagner, "The boomerang attack," in *Fast Software Encryption, FSE'99* (L. R. Knudsen, ed.), vol. 1636 of *Lecture Notes in Computer Science*, pp. 156–170, Springer-Verlag, 1999.

A A 4-round Boomerang Attack

This appendix describes a 4-round boomerang attack which is very similar to the attack of Section 6.3. The top part of the boomerang is identical to the one used for the 5-round attack, but another differential is used in the bottom part.

The idea is to create ciphertexts with difference $(1,2,1,1,1,1,1,1,4,2,2,2,1,1,2,1)$ at the bottom of the last linear layer. This difference causes a difference in a single byte (byte 0) after the linear layer and propagates to 80_x differences in five bytes after one more decryption round with probability 2^{-7}.

In order to cross the output transform, we must take into account that the key bytes which are XORed might disturb the differences. In these positions we will therefore XOR the corresponding differences instead of adding them. This will result in a correct additive difference at the bottom of the linear layer for a fraction 2^{-8} of the ciphertexts.

The total probability of the differential used in the bottom part is 2^{-15}. Combining this with the results of Section 6.3, we find a probability of 2^{-78} for the complete boomerang.

The procedure for attacking 4 rounds is as follows:

1. Prepare a pool of 2^{40} plaintexts P_i, $i = 0, \ldots, 2^{40} - 1$ that have different values in bytes 1, 4, 5, 11, 14 and 15 and are constant in the other bytes. Encrypt the pool to get a pool of 2^{40} ciphertexts C_i.
2. Create a new pool of ciphertexts D_i from the pool C_i by adding $(0,2,1,0,0,1,1,0,0,2,2,0,0,1,2,0)$ and XORing $(1,0,0,1,1,0,0,1,4,0,0,2,1,0,0,1)$ to each C_i. Decrypt the pool D_i to obtain a pool Q_i of 2^{40} plaintexts.
3. Sort the pool Q_i on the bytes that correspond to the constant bytes in the pool P_i and pick the pairs Q_j, Q_k that have zero difference in those ten bytes.
4. For each of the possibly good quartets P_j,P_k,Q_j,Q_k, guess the 3 key bytes K_0^4, K_0^{11} and K_0^{15}, do a partial encryption and check that the difference in the 3 bytes after the first nonlinear layer is of the form $(x, -4x, -x)$ with x odd, both in the P-pair and in the Q-pair. Keep the quartets that have the right difference, together with the key bytes that they suggest. If no quartets survived, go to Step 1.
5. Guess the key byte K_0^{14}, do a partial encryption and check that the difference after XORing the key $K_{0'}^{15}$ (which we know from the previous step) at the bottom of the nonlinear layer is consistent with the difference found in Step 4. If no quartets survived, go to Step 1.
6. Guess the key bytes K_0^1 and $K_{0'}^2$, do a partial encryption and check that the difference after the first keyed nonlinear layer is the right one. Repeat this for the key bytes K_0^5 and $K_{0'}^6$.

From the pool of 2^{40} plaintexts created in Step 1 we get approximately 2^{79} pairs. Since the probability of the boomerang is 2^{-78}, each pool contains two boomerangs on average.

After the sorting phase (Step 3), we expect 2^{-1} wrong quartets to survive since we have an 80-bit condition on 2^{79} pairs. With very high probability, these wrong pairs will be eliminated during the subsequent key recovery steps.

The 4-round attack described above requires 2^{40} plaintexts and adaptively chosen ciphertexts and has a time complexity of 2^{41}. The attack recovers only 8 bytes of the key, but once these bytes are known, the attacker can efficiently recover the remaining key bytes, for example by mounting a second boomerang attack.

B The Linear Layer

This appendix contains the matrices corresponding to half of the linear layer, the linear layer and the inverse of the linear layer.

$$\sqrt{A} = \begin{pmatrix}
0&0&0&0&2&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&2&1&1&0&0&0&0\\
1&1&2&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&2&1&1&1&0&0&0&0\\
1&2&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&2&1\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
2&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&2&1&1\\
0&0&0&0&1&1&2&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&2&1&1&1\\
0&0&0&0&1&2&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&2&1&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

$$A = \begin{pmatrix}
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 2 & 1 \\
2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 4 & 2 & 2 \\
2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 \\
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 2 & 4 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 \\
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 2 & 4 & 2 \\
2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 \\
2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 \\
2 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

$$A^{-1} = \begin{pmatrix}
0 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -4 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & -1 & 16 & -1 & 0 & -4 & 1 & 0 & -4 & 0 & 1 & -4 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & -1 & 0 & 1 & 1 & 0 \\
-4 & 0 & 0 & 1 & 0 & 0 & 0 & 16 & -1 & -1 & -4 & 1 & 0 & -4 & -1 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & -1 \\
0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & 0 \\
-1 & -4 & 0 & 1 & -4 & -1 & -1 & 1 & 0 & 0 & 0 & 16 & 0 & 0 & -4 & 1 \\
0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -4 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -4 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & -4 \\
0 & -1 & -4 & 1 & 0 & -4 & 0 & 1 & -4 & 0 & -1 & 1 & -1 & 0 & 0 & 16
\end{pmatrix}$$