Cryptanalysis of Simpira v1

Christoph Dobraunig, Maria Eichlseder, and Florian Mendel

Graz University of Technology, Austria
maria.eichlseder@iaik.tugraz.at

Abstract. Simpira v1 is a recently proposed family of permutations, based on the AES round function. The design includes recommendations for using the Simpira permutations in block ciphers, hash functions, or authenticated ciphers. The designers' security analysis is based on computer-aided bounds for the minimum number of active S-boxes. We show that the underlying assumptions of independence, and thus the derived bounds, are incorrect. For family member Simpira-4, we provide differential trails with only 40 (instead of 75) active S-boxes for the recommended 15 rounds. Based on these trails, we propose full-round collision attacks on the proposed Simpira-4 Davies-Meyer hash construction, with complexity $2^{82.62}$ for the recommended full 15 rounds and a truncated 256-bit hash value, and complexity $2^{110.16}$ for 16 rounds and the full 512-bit hash value. These attacks violate the designers' security claims that there are no structural distinguishers with complexity below 2^{128}.

Keywords: Simpira · permutation-based cryptography · cryptanalysis · hash functions · collisions

1 Introduction

The Advanced Encryption Standard AES and its underlying wide-trail design strategy are among the most popular building blocks for new symmetric designs. There are several good reasons for this. New AES-like designs profit both from the insights in efficient implementations and from the extensive cryptanalysis and well-understood security bounds of AES. In particular, if new designs not only reuse the general design ideas, but the AES block cipher itself or its round function, then Intel's AES-NI instruction set can provide high software performance on modern CPUs. However, while block ciphers are a versatile building block for other cryptographic primitives, the fixed block size of AES of 128 bits implies a certain limitation. Modern designs often require larger states for efficiency or security. Examples include permutation-based cryptography (hash functions, authenticated encryption, etc.), wide-block encryption, security beyond 2^{64} inputs without resorting to beyond-birthday-security schemes, and more.

These considerations have motivated the design of numerous cryptographic algorithms based on the AES round function. Notable recent examples of dedicated designs include several authenticated encryption algorithms with excellent software performance, such as the CAESAR round-2 candidates AEGIS [15] and

Tiaoxin [12], but also more specialized primitives like the Haraka hash function for short inputs [9]. Very recently, Jean and Nikolić [5] analyzed a more general family of AES-round-based building blocks that generalizes several of the previous dedicated designs. However, except for the last work, these dedicated designs target only specific state sizes, and do not offer scalable, easily reusable building blocks for other cryptographic applications.

Simpira is a recently proposed family of permutations designed by Gueron and Mouha [2] that aims to fill this gap. The design goal is to provide very efficient permutations for arbitrarily large input sizes of $b \cdot 128$ bits, $b \in \mathbb{N}^+$, while taking advantage of the Intel AES-NI instruction set for optimized software implementations. To achieve these goals, Simpira plugs the AES round function into a generalized Feistel construction. Additionally, the designers provide computer-aided bounds for the minimum number of active S-boxes, and argue that these bounds provide security against a wide range of attack vectors. To showcase the versatility of the Simpira permutations, the designers propose a number of application scenarios, including Even-Mansour block cipher constructions, or a keyless Davies-Meyer variant for hash functions with limited-length inputs.

Our contribution. We analyze members of the original Simpira v1 family [2]. We show that the underlying assumptions of independence, and thus the derived bounds on the minimum number of active S-boxes, are incorrect. We focus our analysis on family member Simpira-4 with its 512-bit state, but similar observations also apply to other family members with larger state sizes. For Simpira-4, we provide differential trails with only 40 (instead of 75) active S-boxes for the recommended 15 rounds. Based on these trails, we propose collision attacks on the proposed Simpira-4 Davies-Meyer hash construction. For 16 rounds of the permutation, we obtain collisions for the full 512-bit hash output with complexity $2^{110.16}$. We also adapt the attack to the originally recommended 15 rounds, providing second-order collisions and truncated collisions. We consider several truncation variants, and obtain, among others, collisions on truncated 384-bit output with complexity $2^{110.16}$, or collisions on the 256-bit output with complexity $2^{82.62}$ – the details depend on the implemented truncation variant. These attacks violate the designers' security claims that there are no structural distinguishers below 2^{128}.

Related work. Rønjom [14] independently analyzed Simpira v1, and identified invariant subspaces for any even number of rounds of Simpira-4. Both attacks on Simpira v1 exploit properties of the underlying Type-1.x Generalized Feistel Structure by Yanagihara and Iwata [16] and the sparse, structured round constants. In response to Rønjom's and our attacks, Gueron and Mouha proposed a new version of the design, Simpira v2 [3], which replaces both the Feistel construction and the round constant schedule. In the remaining document, Simpira always refers to Simpira v1.

Simpira is not the first AES-round-based design with problematic round constants. Other examples include the analysis of the hash function Haraka [9] by

Jean [4], the analysis of the withdrawn CAESAR round-1 candidate PAES [17] by Jean et al. [6, 7], or the analysis of SHAvite-3 [1] by Peyrin [13]. In all three cases, the structure of the round constants failed to break the symmetry properties of the unkeyed AES round function. However, our attack exploits different properties, in particular the incomplete diffusion of differences in the structured round constants.

Outline. We first describe the Simpira family of permutations in Sect. 2. We then propose our attacks in Sect. 3, beginning with an iterative truncated differential trail with fewer S-boxes than expected in Sect. 3.1. In Sect. 3.2, we select the bitwise differences of our truncated trail to obtain an 8-round differential trail with probability $2^{-110.16}$. Based on this trail, we propose a collision attack on the 16-round Simpira-4 hash construction in Sect. 3.3. Finally, in Sect. 3.4, we adapt our attack to the recommended 15-round design.

2 Description of Simpira

Simpira is a family of permutations designed by Gueron and Mouha [2]. By using the AES round function in a generalized Feistel construction, it can be adapted to any input size of $b \cdot 128$ bits, $b \in \mathbb{N}^+$. We refer to Simpira family members as Simpira-b.

2.1 F-Function

The Feistel update function $F = F_{c,b}$ applies two rounds of AES, where the Simpira family member b and the round counter c define the round constants. Like for AES, the 128-bit intermediate state of F is represented as a 4×4-matrix of bytes, labelled $s_0, \ldots, s_{15}$:

$$S = \begin{array}{|c|c|c|c|} \hline s_0 & s_4 & s_8 & s_{12} \\ \hline s_1 & s_5 & s_9 & s_{13} \\ \hline s_2 & s_6 & s_{10} & s_{14} \\ \hline s_3 & s_7 & s_{11} & s_{15} \\ \hline \end{array} .$$

We also refer to the value at byte position s_i in state S as $S[i]$.

The operations SubBytes, ShiftRows, and MixColumns are defined identically to AES, whereas AddConstant adds counters that define an invocation counter and the value b:

- SubBytes (SB): Applies the 8-bit AES S-box $\mathcal{S}$ to each of the 16 state bytes.
- ShiftRows (SR): Rotates row i of the state, $0 \leq i \leq 3$, by i bytes to the left.
- MixColumns (MC): Multiplies each byte column of the state by the MDS-matrix M over $\mathbb{K} = \mathbb{F}_2[\alpha]/(\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1)$,

$$M = \begin{pmatrix} \alpha & \alpha+1 & 1 & 1 \\ 1 & \alpha & \alpha+1 & 1 \\ 1 & 1 & \alpha & \alpha+1 \\ \alpha+1 & 1 & 1 & \alpha \end{pmatrix} = \begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix}$$

– AddConstant (AC): In the cth invocation of F for Simpira-b, xors the following round constant $C_{c,b}$ to the state:

$$C_{c,b} = \begin{array}{|c|c|c|c|} \hline c_0 & b_0 & 0 & 0 \\ \hline c_1 & b_1 & 0 & 0 \\ \hline c_2 & b_2 & 0 & 0 \\ \hline c_3 & b_3 & 0 & 0 \\ \hline \end{array} .$$

In the remaining paper, we focus on Simpira-4, so $b_0 = \texttt{04}$ and $b_1 = b_2 = b_3 = \texttt{00}$. Also, since the number of invocations of F is limited to 30 in Simpira-4, $c_1 = c_2 = c_3 = \texttt{00}$. This constant is only added in the first of the two AES rounds of F, while the second round adds 0.

To refer to intermediate states of F for an input S, we use the following notation:

$$S \xmapsto{\mathsf{SB}} S^{\mathsf{SB1}} \xmapsto{\mathsf{SR}} S^{\mathsf{SR1}} \xmapsto{\mathsf{MC}} S^{\mathsf{MC1}} \xmapsto{\mathsf{AC}} S^{\mathsf{AC}} \xmapsto{\mathsf{SB}} S^{\mathsf{SB2}} \xmapsto{\mathsf{SR}} S^{\mathsf{SR2}} \xmapsto{\mathsf{MC}} S^{\mathsf{MC2}} = F(S) \ .$$

2.2 Round Function and Permutation

The permutation Simpira-b keeps a state of $b \cdot 128$ bits. The generalized Feistel round function for $b \geq 4$, where $b \neq 6, 8$, is illustrated in Fig. 1. The final output of Simpira-b for $b \geq 4$, $b \neq 6, 8$, is the state after $6b - 9$ such rounds. Note that if the number of rounds is not a multiple of b, the state words are output in a permuted order to allow for more efficient implementations.

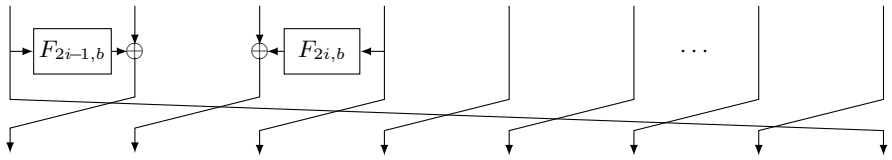


Fig. 1. Round function for round i of Simpira-b for $b \geq 4$, $b \neq 6, 8$.

In case of Simpira-4, we denote the 4 state words before round $i \geq 1$ by $S_i^A, S_i^B, S_i^C, S_i^D$, so the state update rule corresponds to

$$\begin{aligned} S_{i+1}^A &= F_{2i-1,4}(S_i^A) \oplus S_i^B, \\ S_{i+1}^B &= F_{2i,4}(S_i^D) \oplus S_i^C, \\ S_{i+1}^C &= S_i^D, \\ S_{i+1}^D &= S_i^A. \end{aligned}$$

The recommended number of rounds for Simpira-4 is 15, with output words $(S_{16}^B, S_{16}^C, S_{16}^D, S_{16}^A)$.

2.3 Permutation-based Hashing

Simpira's designers identify several application areas for the Simpira permutation, such as block ciphers via an Even-Mansour construction. One particular suggested application is permutation-based hashing for short inputs, where "short" means the state size of any Simpira variant. The proposal is to use a single-block, keyless Davies-Meyer-like construction with a feed-forward, and compute the hash $h(x)$ of x as

$$h(x) = \text{Simpira-}b(x) \oplus x.$$

This approach provides an efficient construction for hashing inputs of limited length, which is required by many applications, such as Lamport signatures [10].

3 Collision Attacks on Simpira-4 Hash

In this section, we show that the number of rounds recommended by the designers is not sufficient to obtain a secure permutation. In particular, we provide collisions for full-round Simpira-4 when used in the hash mode suggested by the designers. While our analysis is focused primarily on Simpira-4, the basic observations also apply to the larger Simpira variants with the same construction approach, that is, Simpira-b with $b \geq 4$, $b \neq 6, 8$.

3.1 Differential Trail with 40 Active S-Boxes over 15 Rounds

The analysis performed by Simpira's designers [2] relies on two basic bounds: full bit diffusion, and minimum number of active S-boxes. The recommended number of rounds for each variant is selected as 3 times the number of rounds necessary to prove full bit diffusion and a minimum number of 25 differentially or linearly active S-boxes. While the proofs for full bit diffusion are based on generic results on the underlying generalized Feistel construction by Yanagihara and Iwata [16], the bounds for active S-boxes were obtained with a Mixed-Integer Linear Programming (MILP) model [11]. For Simpira-4, both full bit diffusion and at least 25 active S-boxes are claimed to be provided by 5 rounds of the round function. For the full number of 15 rounds, this method would imply at least 75 active S-boxes.

The bound is derived under the assumption that all F-function inputs are processed independently. For example, if the F-functions were indeed independent, the 4-round differential trail illustrated in Fig. 2 would contain 20 independently active S-boxes. Since the trail is iterative, and adds 5 active S-boxes per round, this trail also demonstrates the tightness of the 15-round bound.

Of course, in an unkeyed primitive like a permutation or a hash function, the S-boxes are not really independent, since there are no random, independent round keys. Nevertheless, it is usually a reasonable assumption that the differential probabilities behave as if the values were actually independently random. We thus count S-boxes as independently active when it can reasonably be expected

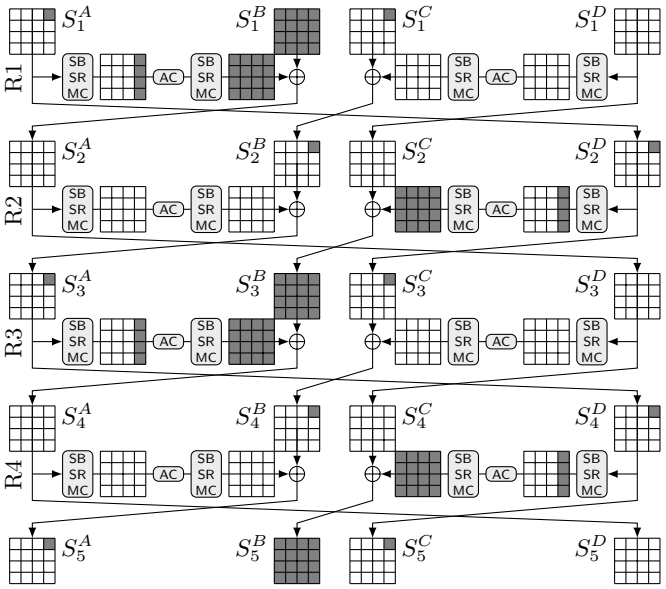


Fig. 2. Iterative 4-round trail for Simpira-4 with 10 independently active S-boxes.

that their multiplied differential probabilities give a good estimate for the overall differential probability of the trail.

However, for all instances of Simpira-b with $b \geq 4$, $b \neq 6, 8$, this independence is violated by the generalized Feistel construction, and the particular definition of F. Consider, for example, the inputs to the active F-functions in rounds 1 and 2, S_1^A and S_2^D. The input values to the two F-functions are identical. Recall the definition of $F = F_{c,b}$, in our case $F_{1,4}$ and $F_{3,4}$. The only difference between $F_{1,4}$ and $F_{3,4}$ is the round-constant addition at the end of the first AES round. This means that the inputs and outputs of the S-boxes of the first AES round must be identical, i.e., $S_1^{A,\mathsf{MC1}} = S_2^{D,\mathsf{MC1}}$. The round constant only differs in state byte s_0, so this means the S-box transitions in the second AES round will also be identical except in s_0. In fact, the outputs $S_1^{A,\mathsf{MC2}}$ of $F_{1,4}$ an $S_2^{D,\mathsf{MC2}}$ of $F_{3,4}$ will have identical values except for the first column.

Considering the 4-round trail of Fig. 2, this means that the entire output difference of $F_{3,4}$ will be identical to that of $F_{1,4}$ with probability 1, as illustrated in Fig. 3. Note that s_0 is not active in the second AES round, and the differential behaviour of MixColumns is independent of the actual values of s_0. Consequently, if we fix all full-state differences to the same bitwise difference pattern, all single-byte differences to the same difference pattern, and all columnwise differences to the same difference pattern, the actual cost of the iterative trail of Fig. 2 is equivalent to only 5 active S-boxes per 2 rounds, or 40 S-boxes overall for the recommended 15 rounds, which is about half as many as suggested by the MILP-based bound. In fact, the MILP model can be adapted to take this into

account by counting only the activity of the left-hand F-functions, and only S-box s_0 for the right-hand F-functions, except in the first round. With this modification, it is easy to prove that 40 active S-boxes is a tight bound for 25 rounds. The minimum number of rounds to achieve at least 25 active S-boxes is then 9, instead of 5.

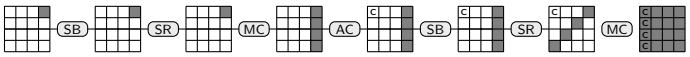


Fig. 3. Trail for the F-function with 5 active S-boxes.

3.2 Collision Attack on 8 Rounds

We now want to use this iterative differential trail of Fig. 2 to find collisions for the permutation-based hash construction suggested for Simpira permutations. Recall that in this short-input Davies-Meyer construction, the $b \cdot 128$-bit message is used as input to the Simpira permutation, and finally added as a feed-forward to the permutation output to produce the untruncated $b \cdot 128$-bit hash value. Our trail is incidentally very well suited to produce collisions for this feed-forward construction. Observe that if we fix all state differences to the same patterns as discussed in Sect. 3.1, the feed-forward will cancel out the message difference with probability 1 for any number of rounds that is a multiple of $b = 4$.

To optimize the complexity of the collision attack, we need to fix the bitwise difference patterns suitably. Recall that the AES S-box has maximum differential probability $\frac{4}{256} = 2^{-6}$. For each nonzero input difference, there is exactly one output difference with this probability (and vice versa), while the other probabilities are either $\frac{2}{256} = 2^{-7}$ or 0. We can easily choose difference patterns so that all S-box transitions have this optimal probability, at least for uniformly random round constants. For example, if we fix the one-byte input difference to `75`, the trail illustrated in Fig. 4 satisfies our requirements. The probability of the differential for the F-function is then at least 2^{-30}. Overall, the probability of such an 8-round trail is at least $2^{-30 \cdot 4} = 2^{-120}$, and the resulting complexity for finding the 512-bit collision is at most 2^{120}.

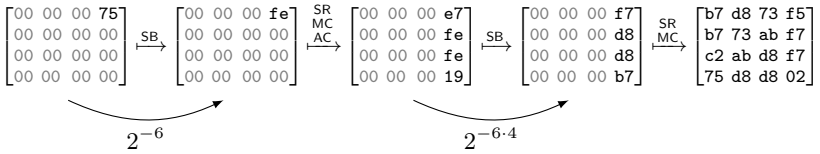


Fig. 4. Trail for the F-function with probability 2^{-30}

Note that we are actually not interested in the probability of the trail within the F-function, but just in the input-output differential from the fixed 1-byte

difference to the fixed 16-byte difference. The probability of this differential is typically higher than that of the trail, since several different trails can contribute to the same differential. In the case of 2-round AES, Keliher and Sui [8] proved that for a random round constant, the probability of the differential in Fig. 4 is actually $2^{-30} + 74 \cdot 2^{-35} \approx 2^{-28.272}$.

If we consider additionally that the round constant is not random, but in our case fixed to $(\mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00})^\top$ for the relevant state bytes, the transition probabilities can increase even further. For example, the differential in Fig. 5 is satisfied with probability $22 \cdot 2^{-32} \approx 2^{-27.54}$. With this differential, the probability of the 8-round trail is increased to $2^{4 \times 27.54} = 2^{-110.16}$.

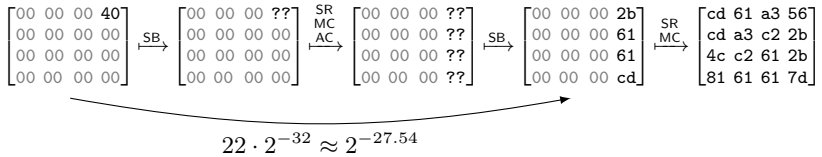


Fig. 5. Differential for F-function with probability $2^{-27.54}$

3.3 Collision Attack on 16 Rounds

Since the permutation involves no round keys, we can try to satisfy the conditions for some active F-functions with message modification. We will try to find messages (or rather, initial structures for intermediate Simpira states) such that the conditions for several rounds are satisfied "for free" with probability 1, and append the previous 8-round trails of Sect. 3.2 to be satisfied probabilistically. We first propose a simple initial structure covering 6 rounds, and then improve it to satisfy all conditions over 8 rounds, thus extending the previous 8-round trail to a 16-round trail with the same probability.

Initial structure for 6 rounds. It is sufficient to set the 4 bytes x_1, x_6, x_{11}, x_{12} of a state S_i^A to a suitable assignment in order to follow the trail for this F-function deterministically. We will refer to these 4 bytes as the diagonal in the following, and to a valid assignment as a valid diagonal. We can reuse one precomputed valid diagonal for all necessary diagonals.

We want to fix the values of the diagonals in S_1^A, S_3^A, and S_5^A to the valid diagonal. Observe that $S_1^A = S_3^C$, and $S_3^A = S_5^C$. Thus, by fixing the diagonals of S_5^A and S_5^C, we have already satisfied 2 F-trails. The remaining $12+16+12$ bytes of S_5^A, S_5^B, S_5^C can be filled arbitrarily, which will immediately determine the value of S_3^D and thus $S_3^{D,\mathsf{MC2}}$. If we now set the diagonal of S_3^C to the valid diagonal, and fill its remaining 12 bytes with arbitrary values, we completely determine S_5^D via S_4^B and S_4^A, and thus complete the state after 4 rounds. By varying the 52 arbitrary byte values, we can obtain the necessary $2^{110.16}$ candidates to satisfy

the 8-round trail. The approach is illustrated in rounds 1–6 of Fig. 6, where ▨ and ▧ mark the 52 arbitrary bytes.

Improved initial structure for 8 rounds by matching diagonals. With some additional effort, we can find initial structures that also satisfy the F-trail in round 7. We will again initialize the values of $S_5^A, S_5^B, S_5^C, S_3^C$ as in the previous 6-round initial structure. However, we can use the $12 + 12$ arbitrary bytes of S_5^A and S_5^C to obtain a valid diagonal in S_7^A. This will provide us with a 16-round collision attack with the same computational complexity as the 8-round trail in Sect. 3.2.

Our goal is to obtain a match between the diagonals of $S_5^{D,\mathsf{MC2}}$ and $S_6^{A,\mathsf{MC2}}$, as illustrated in Fig. 6. If these two diagonals sum to zero, the diagonal of S_7^A will take the exact same value as that of S_5^C, which is the valid diagonal. For this purpose, we want to initialize part of the initial structure to generate random values in $S_6^{A,\mathsf{MC2}}$, and independently a different part of the initial structure, to independently get random values in $S_5^{D,\mathsf{MC2}}$. Then, any match between the two corresponds to an initial structure that satisfies 4 F-trails.

Assume that S_3^C and S_5^B are already fixed to some arbitrary constants, with the valid diagonal in S_3^C. We first use the free bytes of S_5^A to randomize $S_6^{A,\mathsf{MC2}}$. Any complete assignment of S_5^A will directly determine $S_6^{A,\mathsf{MC2}}$ via $S_5^{A,\mathsf{MC2}}$ and S_6^A. We can assume the values are distributed reasonably close to uniformly random, since the values are processed by 4 AES rounds, and only 4 input bytes are fixed.

Independently, we can vary the 12 bytes of S_5^C to randomize the diagonal of $S_5^{D,\mathsf{MC2}}$. To see the independence of the values in S_5^A, consider the diagonal of $S_4^{A,\mathsf{MC2}}$. Its values will always be identical to that of $S_5^{D,\mathsf{MC2}}$, except for the first column, which is influenced by the round constant and will be considered separately in a moment. Since the diagonals of S_5^A and S_3^C are fixed and predetermined, these values can further be traced back right to $S_3^{D,\mathsf{MC2}}$. Thus, knowing the diagonal of $S_3^{D,\mathsf{MC2}}$ is equivalent to knowing the target diagonal of $S_5^{D,\mathsf{MC2}}$, except for 1 byte in s_1. This equivalent diagonal is derived easily from S_5^C, again by 4 AES rounds via $S_4^D, S_4^{D,\mathsf{MC2}}, S_4^C$.

Evaluating the missing match byte s_1 of $S_5^{D,\mathsf{MC2}}$. Now we still need to account for the missing byte s_1. Fortunately, with some minor modifications of our guessing strategy, this value can also be computed directly from $S_3^{D,\mathsf{MC2}}$. Instead of varying all 12 arbitrary bytes of S_5^A to produce our matching candidates, we will keep the first column (bytes s_0, s_2, s_3) fixed. In fact, for simplicity, we will set them to the exact same values as the first column of S_3^C:

$$S_5^A[0, \ldots, 3] = S_3^C[0, \ldots, 3].$$

This implies that the values of the first column and diagonal (bytes $s_0, \ldots, s_3$, s_6, s_{11}, s_{12}) must be identical between $S_3^{D,\mathsf{MC2}}$ and $S_4^{A,\mathsf{MC2}}$. By partially inverting

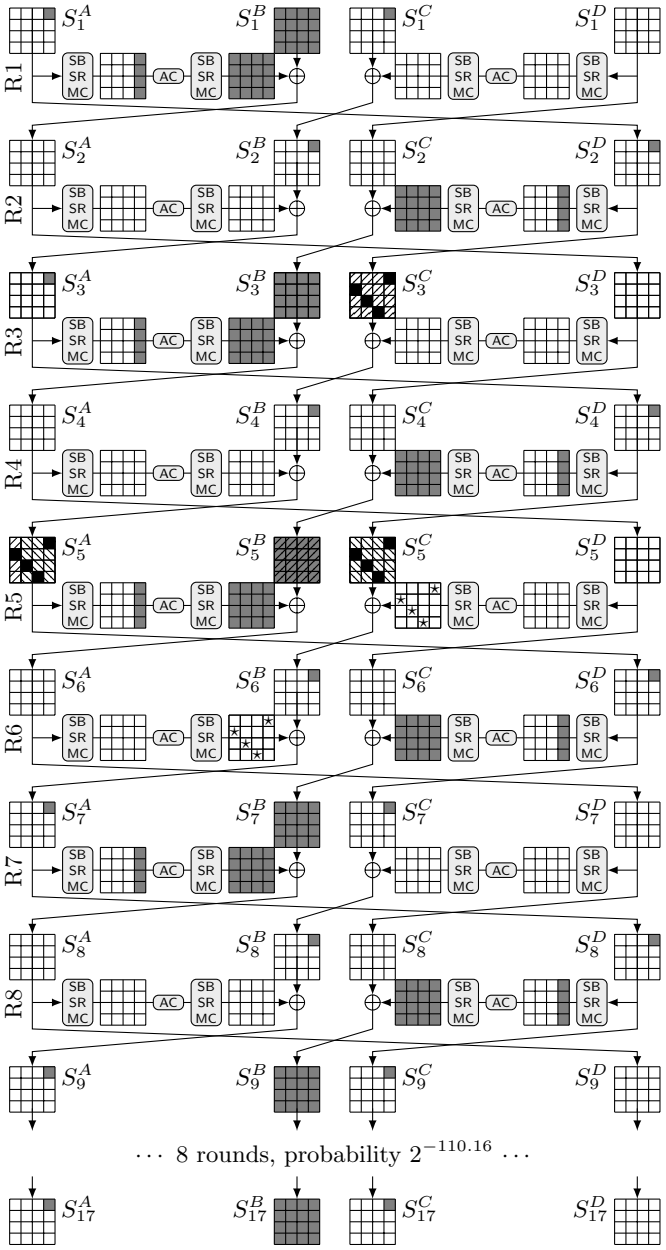


Fig. 6. 16-round collision attacks on Simpira-4 hash using 8-round initial structure. ■ fixed difference, ■ valid diagonal, ▨ arbitrary bytes, ▧ matching inputs, ⊡ match

the last few steps of F, we can also easily verify that this means that

$$S_3^{D,\mathsf{AC}}[0] = S_4^{A,\mathsf{AC}}[0].$$

To determine our target value s_1 in $S_5^{D,\mathsf{MC2}}$, consider a differential view of the intermediate variables in the computations $F(S_4^A)$ and $F(S_5^D)$. The input values are identical, but a difference in s_0 is introduced by $\mathsf{AddConstant}$. We are interested in how this difference ΔS^{AC} propagates to the target byte in ΔS^{MC2}. Since we only introduced a single-byte difference before the final $\mathsf{MixColumns}$, we get

$$\begin{aligned}\Delta S^{\mathsf{MC2}}[1] &= \mathtt{01} \cdot \Delta S^{\mathsf{SB2}}[0] \\ &= \mathcal{S}\left(S_4^{A,\mathsf{AC}}[0]\right) \oplus \mathcal{S}\left(S_4^{A,\mathsf{AC}}[0] \oplus \Delta S^{\mathsf{AC}}[0]\right).\end{aligned}$$

By using the previously established identities between $F(S_4^A)$ and $F(S_3^D)$, and observing $\Delta S^{\mathsf{AC}}[0] = \mathtt{07} \oplus \mathtt{0A} = \mathtt{0D}$, we finally obtain all our target match bytes in $S_5^{D,\mathsf{MC2}}$ directly from $F(S_3^D)$:

$$\begin{aligned}S_5^{D,\mathsf{MC2}}[1] &= S_4^{A,\mathsf{MC2}}[1] \oplus \Delta S^{\mathsf{MC2}}[1] \\ &= S_4^{A,\mathsf{MC2}}[1] \oplus \mathcal{S}\left(S_4^{A,\mathsf{AC}}[0]\right) \oplus \mathcal{S}\left(S_4^{A,\mathsf{AC}}[0] \oplus \mathtt{0D}\right) \\ &= S_3^{D,\mathsf{MC2}}[1] \oplus \mathcal{S}\left(S_3^{D,\mathsf{AC}}[0]\right) \oplus \mathcal{S}\left(S_3^{D,\mathsf{AC}}[0] \oplus \mathtt{0D}\right), \\ S_5^{D,\mathsf{MC2}}[6] &= S_3^{D,\mathsf{MC2}}[6], \\ S_5^{D,\mathsf{MC2}}[11] &= S_3^{D,\mathsf{MC2}}[11], \\ S_5^{D,\mathsf{MC2}}[12] &= S_3^{D,\mathsf{MC2}}[12].\end{aligned}$$

Complexity of generating initial structures. Summarizing, we can now generate a large number of initial structures as follows. First, fix the diagonals in S_3^C and S_5^C to any valid diagonal. Fix all remaining bytes of S_3^C and S_5^B to arbitrary values. Copy the valid diagonal and first column of S_3^C to S_5^A. Vary the remaining 9 bytes of S_5^A, storing the resulting values of the diagonal of $S_6^{A,\mathsf{MC2}}$ in a list. Independently vary the 12 bytes of S_5^C, derive the diagonal of $S_5^{D,\mathsf{MC2}}$, and store it in a second list. Any match between the two lists gives a valid initial structure that follows the differential trail up to round 8.

If we only wanted one match on the 4 bytes of the diagonal, we could try 2^{16} values each for S_5^A and S_5^C, and would expect roughly $2^{2\cdot 16-32} = 1$ match due to the birthday effect. However, consider using 2^{32} values each instead. The expected number of 4-byte matches is roughly $2^{2\cdot 32-32} = 2^{32}$. Now we evaluate the complexity for generating these 2^{32} solutions. Computing the match bytes requires to evaluate $2 \cdot 2 \cdot 2^{32} = 2^{34}$ F-functions. Since 16-round Simpira-4 evaluates more than $16 = 2^4$ F-functions, this corresponds to a complexity of about $2^{32-4} = 2^{30}$ Simpira-4 evaluations. Thus, we were able to produce solutions with amortized complexity less than 1. With this initial structure, we obtain a 16-round collision with computational complexity about $2^{4\times 27.54} = 2^{110.16}$. The memory requirements are only about $2^{32} \cdot 2$ AES states.

3.4 Collision Attack on 15 Rounds with Truncation

In Sect. 3.3, we actually attacked more than the recommended number of 15 rounds for the Simpira-4. In the following, we discuss the applicability of the analysis to the original 15-round design.

Permutation distinguisher. Clearly, the 16-round trail of Fig. 6 also immediately leads to a 15-round permutation distinguisher. With a computational complexity of $2^{110.16}$, we can find pairs of inputs with a fixed input difference such that the permutation outputs collide in 62 of 64 bytes, or actually in 510 of 512 bits, since we use the 1-byte differences of Fig. 5. This property implies, for example, second-order collisions for the hash construction with complexity $2 \cdot 2^{110.16}$, whereas the generic complexity bound is at least about $2^{512/4} = 2^{128}$. This distinguisher violates the security claims for Simpira-4.

Furthermore, if we impose no constraints on the active F-function in round 15 by allowing arbitrary constraints in $S_{15}^{A,\mathsf{MC2}}$ and thus in S_{16}^{A}, we still get a collision on at least 46 of 64 bytes, or in at least 382 of 512 bits, with a fixed input difference. Then, only the 3 active F-functions in rounds 9, 11, and 13 need to be satisfied probabilistically. The probability for this trail is $2^{-3\times 27.54} = 2^{-82.62}$.

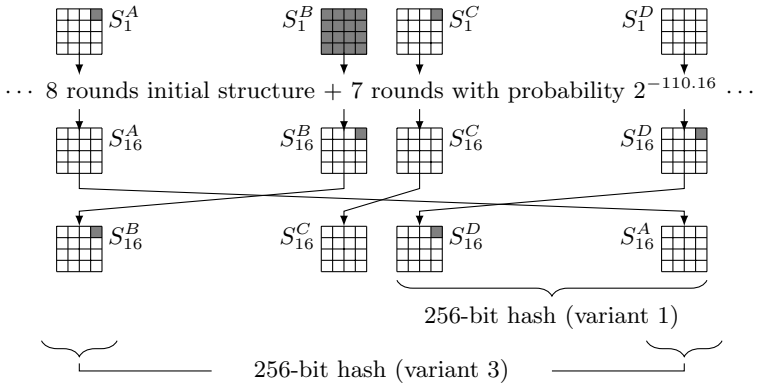
Truncated collisions. The trail no longer automatically leads to full-state collisions for the hash construction, since the 2 active state words we get after an odd number of rounds cannot cancel all 3 active state words at the input. However, we can consider truncated versions of the hash construction. Since the permutation-based Simpira-4 hash construction claims only 128-bit security, but the state size is 512 bits, Simpira's designers comment that "truncation of the output of Simpira may be required [...] to match the intended application". An obvious choice would be to truncate the state to 256 bits, so that the security claim matches the generic bound. The details and complexity of the collision attack then vary depending on the implementation of this truncation. Below, we consider 3 natural choices for truncation.

Truncation variant 1: Left/right half. The most intuitive choice is to simply truncate to the right (or left) half of the final state. Consider the rightmost 256 bits. With the previous 16-round trail of Fig. 6 and 7a, the permutation of the output words means that this conveniently corresponds to a hash output of

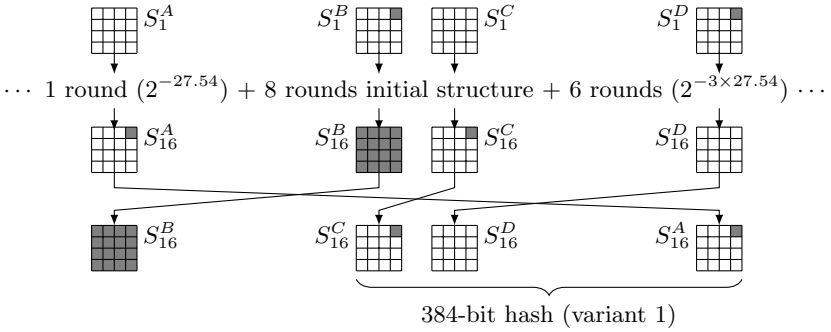
$$(S_1^C \oplus S_{16}^D,\ S_1^D \oplus S_{16}^A) = \Big($$  $$,$$  $$\Big) = \Big($$  $$,$$  $$\Big).$$

In fact, we can extend this to collisions up to the rightmost 384 bits if we just shift our iterative trail down by 1 round, as illustrated in Fig. 7b. The probabilistic part of the trail is then moved to rounds 1 (input S_1^D) and rounds 10, 12, and 14 (inputs S^A). For the same complexity of $2^{110.16}$, we get a 384-bit hash collision of the output

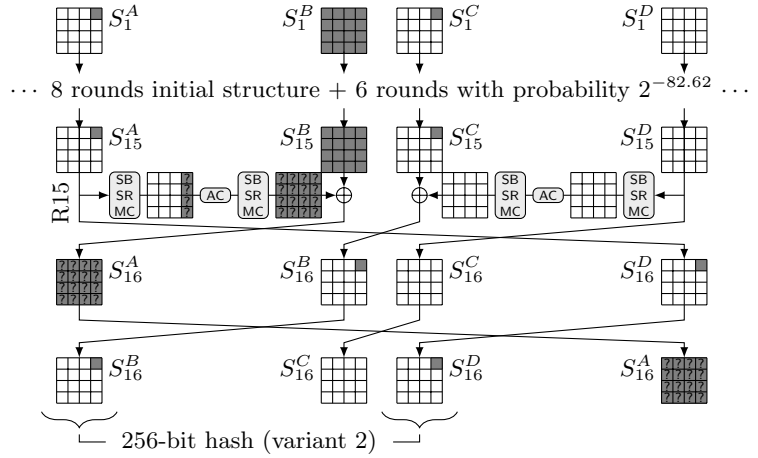
$$(S_1^B \oplus S_{16}^C,\ S_1^C \oplus S_{16}^D,\ S_1^D \oplus S_{16}^A).$$



(a) Truncation variants 1 and 3: 256-bit collisions with complexity $2^{110.16}$



(b) Truncation variant 1: 384-bit collisions with complexity $2^{110.16}$



(c) Truncation variant 2: 256-bit collisions with complexity $2^{82.62}$

Fig. 7. Collisions for truncated 15-round Simpira-4 hash.

Truncation variant 2: Every second word. Assume the truncation function selects every second word, that is, the 256-bit hash output is

$$(S_1^A \oplus S_{16}^B,\ S_1^C \oplus S_{16}^D).$$

Then, we can even take advantage of the improved permutation distinguisher with complexity $2^{82.62}$, as in Fig. 7c.

Truncation variant 3: Updated words. In the previous truncation variants, we took advantage of the fact that the output of one of the last round's two F-functions was truncated. Consequently, another good candidate for a truncation function is to select exactly the words that depend on the last round's F-outputs, S_{16}^A and S_{16}^B, so the hash output is

$$(S_1^A \oplus S_{16}^B,\ S_1^D \oplus S_{16}^A).$$

Nevertheless, the trail of Fig. 7a still provides hash collisions with complexity $2^{110.16}$.

4 Conclusion

In this paper, we analyzed the permutations Simpira-b, $b \geq 4$, $b \neq 6, 8$, of the Simpira v1 family, with a focus on Simpira-4. Due to properties of the underlying Type-1.x Generalized Feistel Structure and the sparse round constants, the computer-aided bounds given by the designers for the minimum number of active S-boxes are invalid. The count includes many pairs of S-boxes whose inputs are not independent, in particular, many actually share the exact same inputs. Based on differential trails that exploit this property, we propose full-round collision attacks on the proposed Simpira-4 Davies-Meyer hash construction, with complexities down to $2^{82.62}$ for the recommended full 15 rounds and the truncated 256-bit hash value, depending on the truncation rule, and complexity $2^{110.16}$ for 16 rounds and the full 512-bit hash value.

The attacks exploit Generalized Feistel Structures which apply multiple F-functions to a Feistel branch without xoring other F-outputs in between, as would be the case in a standard Feistel construction. While it is not clear whether this property could be exploited in general for independent F, it certainly becomes a problem when the F-functions differ only by using different, sparse round constants. In Simpira v1, this is the case for all family members $b \geq 4$, $b \neq 6, 8$. The consequence is that two branches of the state will be updated with two closely related F-outputs.

To address the problems described in this paper and by Rønjom [14], Gueron and Mouha subsequently tweaked their design [3]. The new Simpira v2 fixes the issue by replacing both the Feistel construction, to ensure disjoint F-inputs, and the round constants with denser values.

Acknowledgments. We thank the Simpira designers Shay Gueron and Nicky Mouha for verifying our results and providing useful suggestions.

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR) and from the Austrian Science Fund (project P26494-N15).

References

1. Biham, E., Dunkelman, O.: The SHAvite-3 hash function. Submission to NIST (SHA-3 Round 2) (2009), http://www.cs.technion.ac.il/~orrd/SHAvite-3/Spec.15.09.09.pdf
2. Gueron, S., Mouha, N.: Simpira: A family of efficient permutations using the AES round function. Cryptology ePrint Archive, Report 2016/122 (2016), http://eprint.iacr.org/2016/122/20160214:005409
3. Gueron, S., Mouha, N.: Simpira v2: A family of efficient permutations using the AES round function. Cryptology ePrint Archive, Report 2016/122 (2016), http://eprint.iacr.org/2016/122/20160306:231039
4. Jean, J.: Cryptanalysis of Haraka. Cryptology ePrint Archive, Report 2016/396 (2016), http://ia.cr/2016/396
5. Jean, J., Nikolić, I.: Efficient design strategies based on the AES round function. FSE 2016. See also: Cryptology ePrint Archive, Report 2016/299 (2016), http://ia.cr/2016/299
6. Jean, J., Nikolić, I., Sasaki, Y., Wang, L.: Practical cryptanalysis of PAES. In: Joux, A., Youssef, A.M. (eds.) SAC 2014. LNCS, vol. 8781, pp. 228–242. Springer (2014)
7. Jean, J., Nikolić, I., Sasaki, Y., Wang, L.: Practical forgeries and distinguishers against PAES. IEICE Transactions 99-A(1), 39–48 (2016)
8. Keliher, L., Sui, J.: Exact maximum expected differential and linear probability for two-round Advanced Encryption Standard. IET Information Security 1(2), 53–57 (2007), http://ia.cr/2005/321
9. Kölbl, S., Lauridsen, M.M., Mendel, F., Rechberger, C.: Haraka – efficient short-input hashing for post-quantum applications. Cryptology ePrint Archive, Report 2016/098 (2016), http://ia.cr/2016/098
10. Lamport, L.: Constructing digital signatures from a one-way function. Tech. Rep. SRI-CSL-98, SRI International Computer Science Laboratory (1979)
11. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Wu, C., Yung, M., Lin, D. (eds.) Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer (2011)
12. Nikolić, I.: Tiaoxin v2. Submission to the CAESAR competition (2015), http://competitions.cr.yp.to/round2/tiaoxinv2.pdf
13. Peyrin, T.: Chosen-salt, chosen-counter, pseudo-collision for the compression function of SHAvite-3. NIST mailing list (2009), http://ehash.iaik.tugraz.at/uploads/e/ea/Peyrin-SHAvite-3.txt
14. Rønjom, S.: Invariant subspaces in Simpira. Cryptology ePrint Archive, Report 2016/248 (2016), http://ia.cr/2016/248
15. Wu, H., Preneel, B.: AEGIS v1. Submission to the CAESAR competition (2014), http://competitions.cr.yp.to/round1/aegisv1.pdf
16. Yanagihara, S., Iwata, T.: Type 1.x generalized Feistel structures. IEICE Transactions 97-A(4), 952–963 (2014)
17. Ye, D., Wang, P., Hu, L., Wang, L., Xie, Y., Sun, S., Wang, P.: PAES v1. Submission to the CAESAR competition (2014), http://competitions.cr.yp.to/round1/paesv1.pdf