Simpira: A Family of Efficient Permutations Using the AES Round Function

Shay Gueron[1,2] and Nicky Mouha[3,4]

[1] University of Haifa, Israel.
[2] Intel Corporation, Israel Development Center, Haifa, Israel.
[3] Dept. Electrical Engineering-ESAT/COSIC, KU Leuven, Leuven and iMinds, Ghent, Belgium.
[4] Project-team SECRET, Inria, France.
shay@math.haifa.ac.il, nicky@mouha.be

Abstract. This paper introduces Simpira, a family of cryptographic permutations that supports inputs of $128 \times b$ bits, where b is a positive integer. Its design goal is to achieve high throughput on virtually all modern 64-bit processor architectures, that nowadays already have native instructions to support AES computations. To achieve this goal, Simpira uses only one building block: the AES round function. For $b = 1$, Simpira corresponds to 12-round AES with fixed round keys, whereas for $b \geq 2$, Simpira is a Generalized Feistel Structure (GFS) with an F-function that consists of two rounds of AES. From the security viewpoint, we claim that there are no structural distinguishers for Simpira with a complexity below 2^{128}, and analyze its security against a variety of attacks in this setting. From the efficiency viewpoint, we show that the throughput of Simpira is close to the theoretical optimum, namely, the number of AES rounds in the construction. For example, on the latest Intel Skylake processor, Simpira has throughput below 1 cycle per byte for $b \leq 4$ and $b = 6$. For larger permutations, where moving data in memory has a more pronounced effect, Simpira with $b = 32$ (512 byte inputs) evaluates 732 AES rounds, and performs at 802 cycles (1.56 cycles per byte), i.e., less than 10% off the theoretical optimum. The Simpira family offers an efficient solution for multiple usages where operating on wide blocks, larger than 128 bits, is desired.

Keywords. Cryptographic permutation, AES-NI, Generalized Feistel Structure (GFS), hash function, Lamport signature, wide-block encryption, Even-Mansour.

1 Introduction

The introduction of AES instructions by Intel (subsequently by AMD, and recently ARM) has changed the playing field for symmetric-key cryptography on modern processors, because it significantly reduced the encryption overheads. The performance of these instructions has been steadily improving in every new generation of processors. By now, on the latest Intel Architecture Codename

Skylake, the AESENC instruction that computes one round of AES has latency of 4 cycles and throughput of 1 cycle. The improved AES performance trend can be expected to continue, with the increasing demand for fast encryption of more and more data.

To understand the impact of the AES instructions in practice, consider for example, the way that Google Chrome browser connects to `https://google.com`. In this situation, Google is in a privileged position, as it controls both the client and the server side. To speed up connections, Chrome (the client) is configured to identify the processor's capabilities. If AES-NI are available, it would offer (to the server) to use AES-128-GCM for performing authenticated encryption during the TLS handshake. The high-end server would accept the proposed cipher suite, due to the high performance of AES-GCM on its side. This would capture any recent 64-bit PC, tablet, desktop, or even smartphone. On older processors, or architectures without AES instructions, Chrome resorts to proposing the ChaCha20-Poly1305 algorithm during the secure handshake negotiation.

An advantage of AES-GCM is that the message blocks can be processed independently for encryption. This allows pipelining of the AES round instructions, so that the observed performance is dominated by their throughput, and not by their latency [43, 44]. We note that even if a browser negotiates to use an inherently sequential mode such as CBC encryption, the web server can process multiple independent data buffers in parallel to achieve high throughput (see [43, 44]), and this technique is already used in the recent OpenSSL version 1.0.2. This performance gain by collecting multiple independent encryption tasks and pipelining their execution, is important for the design rationale of Simpira.

Setting. This paper should be understood in the following setting. We focus only on processors with AES instructions. Assuming that several independent data sources are available, we explore several symmetric-key cryptographic constructions with the goal of achieving a high throughput. Our reported benchmarks are performed on the latest Intel processor, namely Architecture Codename Skylake, but we expect to achieve similar performance on any processor that has AES instructions with throughput 1.

In particular, we focus here on applications where the 128-bit block size of AES is not sufficient, and support for a wider range of block sizes is desired. This includes various use cases such as permutation-based hashing and wide-block encryption, or just to easily achieve security beyond 2^{64} input blocks without resorting to (often inefficient) modes of operation with "beyond birthday-bound" security. For several concrete suggestions of applications, we refer to Sect. 7.

Admittedly, our decision to focus on only throughput may result in unoptimized performance in certain scenarios where the latency is critical. However, we point out that this is not only a property of Simpira, but also of AES itself, when it is implemented on common architectures with AES instructions. To achieve optimal performance on such architectures, AES needs to be used in a parallelizable mode of operation, or in a protocol that supports processing independent inputs. Similarly, this is the case for Simpira as well. In fact, for 128-bit inputs, Simpira is the same as 12-round AES with fixed round keys.

Origin of the name. Simpira is named after a mythical animal of the Peruvian Amazon. According to the legend, one of its front legs has the form of a spiral that can be extended to cover the entire surface of the earth [26]. In a similar spirit, the Simpira family of permutations extends itself to a very wide range of input sizes. Alternatively, Simpira can be seen as an acronym for "SIMple Permutations based on the Instruction for a Round of AES."

2 Related Work

Block ciphers that support wide input blocks have been around for a long time. Some of the earliest designs are Bear and Lion [2], and Beast [59]. They are higher-level constructions, in the sense that they use hash functions and stream ciphers as underlying components.

Perhaps the first wide-block block cipher that is not a higher-level construction is the Hasty Pudding Cipher [72], which supports block sizes of any positive number of bits. Another early design is the Mercy block cipher that operates on 4096-bit blocks [27]. More recently, low-level constructions that can be scaled up to large input sizes are the SPONGENT [17,18] permutations and the LowMC [1] block ciphers.

Our decision to use only the AES round function as a building block for Simpira means that some alternative constructions are not considered in this paper. Of particular interest are the EGFNs [6] used in Lilliput [7], the AESQ permutation of PAEQ [13], and Haraka [54]. The security claims and benchmark targets of these designs are very different from those of Simpira. We only claim security up to 2^{128}. However unlike Haraka, we consider all distinguishing attacks up to this bound. Also, we focus only on throughput, and not on latency. An interesting topic for future work is to design variants of these constructions with similar security claims, and to compare their security and implementation properties with Simpira.

3 Design Rationale of Simpira

AES [31] is a block cipher that operates on 128-bit blocks. It iterates the AES round function 10, 12 or 14 times, using round keys that are derived from a key of 128, 192 or 256 bits, respectively. On Intel (and AMD) processors, the AES round function is implemented by the `AESENC` instruction. It takes a 128-bit state and a 128-bit round key as inputs, and returns a 128-bit output that is the result of applying the `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` operations. An algorithmic description of `AESENC` is given in Alg. 1 of Sect. 4, where we give the full specification of Simpira.

A cryptographic permutation can be obtained by setting the AES round keys to fixed, publicly-known values. It is a bad idea to set all round keys to zero. Such a permutation can easily be distinguished from random: if all input bytes are equal to each other, the AES rounds preserve this property. Such problems are avoided when round constants are introduced: this breaks the symmetry inside every round, as well as the symmetry between rounds.

We decided to use two rounds of AES in Simpira as the basic building block. As the `AESENC` instruction includes an XOR with a round key, this can be used to introduce a round constant in one AES round, and to do a "free XOR" in the other AES round. An added advantage is that two rounds of AES achieve *full bit diffusion*: every output bit depends on every input bit, and every input bit depends on every output bit.

Another design choice that we made, is to use only AES round functions in our construction, and no other operations. Our hope is that this design would maximize the contribution of every instruction to the security of the cryptographic permutation. It also simplifies the analysis and the implementation. From the performance viewpoint, the theoretically optimal software implementation would be able to dispatch a new `AESENC` instruction in every CPU clock cycle. A straightforward way to realize this design strategy is to use a (Generalized) Feistel Structure (GFS) for $b \geq 2$ that operates on b input subblocks of 128 bits each, as shown in Fig. 1.

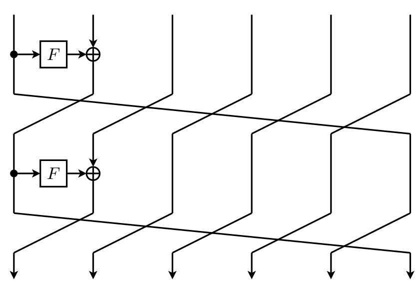

Fig. 1. Two common classes of Generalized Feistel Structures (GFSs) are the Type-1 GFS (left) and the Type-2 GFS (right). For each example, two rounds are shown of a GFS that operates on $b = 6$ subblocks. We will consider these GFSs in this paper, as well as other GFSs with a different number of F-functions per round, and other *subblock shuffles* at the end of every round.

As with any design, our goal is to obtain a good trade-off between security and efficiency. In order to explore a large design space, we use simple metrics to quickly estimate whether a given design reaches a sufficient level of security, and to determine its efficiency. In subsequent sections, we will formally introduce the designs, and study them in detail to verify the accuracy of our estimates.

3.1 Design Criteria

Our design criteria are as follows. The significance of both criteria against cryptanalysis attacks will be explained in Sect. 6.

- **Security**: We calculate the number of Feistel rounds to achieve either *full bit diffusion*, as well as the number of Feistel rounds to achieve at least 25 (linearly or differentially) active S-boxes. To ensure a sufficient security margin against known attacks, we require that the number of rounds is three times the largest of these two numbers.
- **Efficiency**: As explained in Sect. 1, we will only focus on throughput. Given that we use no other operations besides the AES round function, we will use the number of AES round functions as an estimate for the total number of cycles.

Suzaki and Minematsu [73] formally defined DRmax to calculate how many Feistel rounds are needed for an input subblock to affect all the output subblocks. We will say that *full subblock diffusion* is achieved after DRmax of the permutation or its inverse, whichever is greater. To achieve the strictly stronger criterion of *full bit diffusion*, one or two additional Feistel rounds may be required.

To obtain a lower bound for the minimum number of active S-boxes, we use a simplified representation that assigns one bit to every pair of bytes, to indicate whether or not they contain a non-zero difference (or linear mask). This allows us to use the Mixed-Integer Linear Programming (MILP) technique introduced by Mouha et al. [67] quickly find a lower bound for the minimum number of active S-boxes.

3.2 Design Space Exploration

For each input size of the permutation, we explore a range of designs, and choose the one that maximizes the design criteria. If the search returns several alternatives, it does not really matter which one we choose. In that case, we arbitrarily choose the "simplest" design. The resulting Simpira design is shown in Fig. 2.

Case $b = 1$. Full bit diffusion is reached after two rounds of AES, and four rounds of AES ensures at least 25 active S-boxes [31]. Following the design criteria, we select a design with 12 AES rounds.

Case $b = 2$. This is a (standard) Feistel structure. Full subblock diffusion is achieved after two Feistel rounds, and three Feistel rounds are needed to reach full bit diffusion. We find that five rounds ensures that there are at least 25 active S-boxes (see Fig. 4). Consequently, we select a design with 15 Feistel rounds.

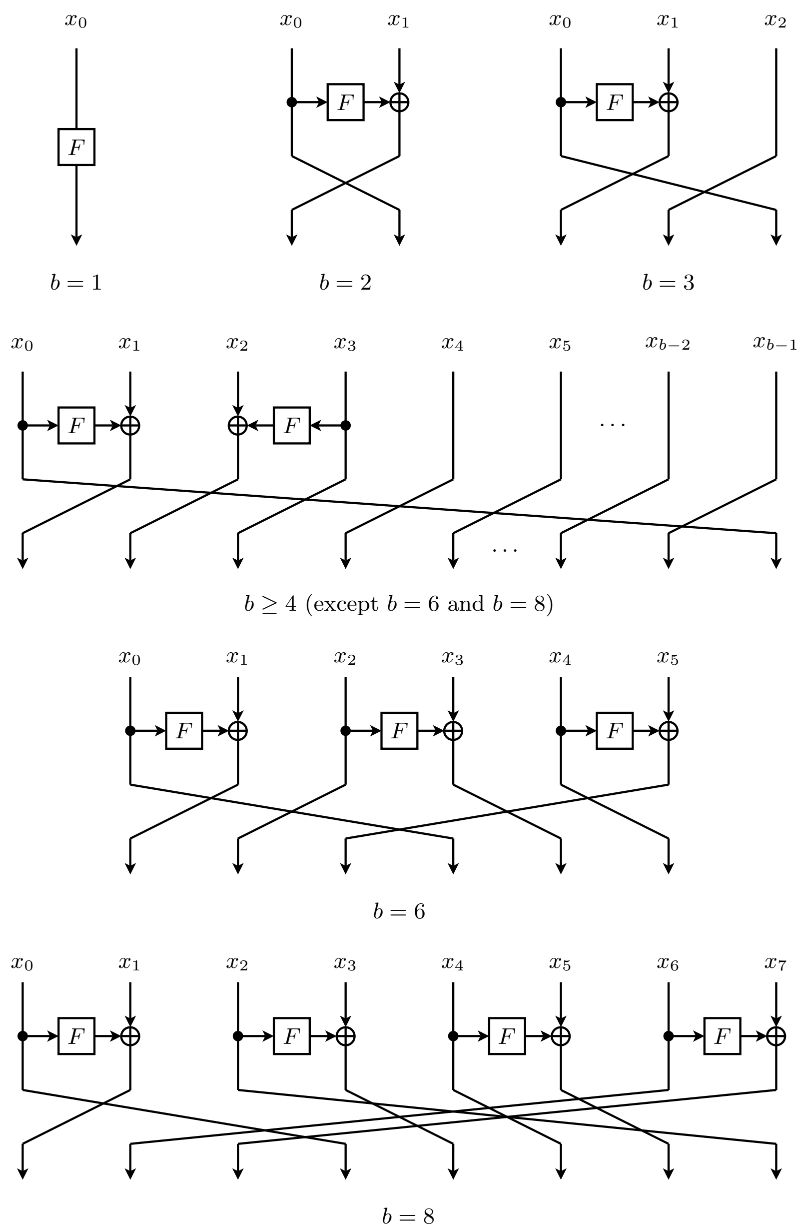

Fig. 2. One round of the Simpira construction for various choices of b. The total number of rounds is $6b + 3$ for $b \leq 3$ and $6b - 9$ for $b \geq 4$, with three exceptions: 6 for $b = 1$, 15 for $b = 6$, and 18 for $b = 8$. F is shorthand for $F_{c,b}$, where c is a counter that is initialized by one, and incremented after every evaluation of $F_{c,b}$. Every $F_{c,b}$ consists of two AES round evaluations, where the round constants that are derived from (c, b). The last round is special: the `MixColumns` is omitted when $b = 1$, and the subblock shuffle may be different when $b \geq 2$. See Sect. 4 for a full specification.

Case $b = 3$. There are several designs that are optimal according to our criteria. They have either one or two F-functions per Feistel round, and various possibilities exist to reorders the subblocks at the end of every Feistel round. We choose what is arguably the simplest design: a Type-1 GFS according to Zheng et al.'s classification [79]. Full subblock diffusion requires five Feistel rounds, and at least six Feistel rounds are needed to ensure that there are least 25 active S-boxes. As seven Feistel rounds are needed to achieve full bit diffusion, we select a design with 21 Feistel rounds.

Case $b \geq 4$. The Type-1 GFS does not scale well for larger b, as diffusion becomes the limiting factor. More formally, Yanagihara and Iwata [75, 76] proved that the number of rounds required to reach full subblock diffusion is (at best) quadratic in the number of subblocks, regardless of how the subblocks are reordered at the end of every Feistel round. In subsequent work, Yanagihara and Iwata [77] introduced a GFS with two F-functions per round, where the number of rounds for full subblock diffusion is linear b. More specifically, their Type-1.x (b,2) GFS reaches full subblock diffusion after $2b - 4$ rounds.

To simplify the implementation, we use a variant of their construction with a cyclic left shift of the subblocks at the end of every Feistel round. For $b = 4$, five Feistel rounds are sufficient for both full bit diffusion and 25 active S-boxes. When $b \geq 5$, full bit diffusion is reached after $2b - 4$ rounds. For $b \leq 128$, we used the MILP technique to verify that $2b - 3$ rounds are needed to ensure 25 active S-boxes. In fact, the tool shows that there will be at least 30 active S-boxes. We conjecture that this property will hold for larger b as well. Therefore, our criterion tells us to go for $6b - 9$ rounds. See Appendix A for linear and differential characteristics that were output by the tool.

Note that GFSs with more than two F-functions reach full subblock diffusion even quicker, but this seems to come at the cost of using more F-functions in total. Looking only at the tabulated values of DRmax(π) and DRmax(π^{-1}) in literature [73,75–77], we can immediately rule out almost all alternative designs. However, two improved Type-2 GFS designs by Suzaki and Minematsu [73] turned out be superior. Instead of a cyclic left shift, they reorder the subblocks in a different way at the end of every Feistel round. We now explore these in detail.

Case $b = 6$. Let the *subblock shuffle* at the end of every Feistel round be presented by a list of indices that indicates which input subblock is mapped to which output subblock, e.g. $\{b-1, 0, 1, 2, \ldots, b-2\}$ denotes a cyclic left shift. Suzaki and Minematsu's improved Type-2 GFS with subblock shuffle $\{3, 0, 1, 4, 5, 2\}$ reaches full block diffusion and full bit diffusion after five Feistel rounds. At least 25 active S-boxes (in fact at least 30) are reached after four Feistel rounds. Following the design criteria, we end up with a design with 15 Feistel rounds. As this design has three F-functions in every Feistel round, it evaluates $3 \cdot 15 = 45$ F-functions. This is less than the general $b \geq 4$ case that requires $6b - 9$ Feistel rounds with 2 F-functions per round, which corresponds to $(6 \cdot 6 - 9) \cdot 2 = 54$ F-functions.

Case $b = 8$. Suzaki and Minematsu's improved Type-2 GFS with subblock shuffle $\{3, 0, 7, 4, 5, 6, 1, 2\}$ ensures both full block diffusion and full bit diffusion after six rounds. After four Feistel rounds, there are at least 25 active S-boxes (in fact at least 30). According to the design criteria, we end up with a design with 18 Feistel rounds, or $18 \cdot 4 = 72$ F-functions in total. The general $b \geq 4$ design would have required $(6b - 9) \cdot 2$ F-functions, which for $b = 8$ corresponds to $(6 \cdot 8 - 9) \cdot 2 = 78$ F-functions.

3.3 Design Alternatives

Until now, the only designs we discussed were GFS constructions where the F-function consists of two rounds of AES. We now take a step back, and briefly discuss alternative design choices.

As explained earlier, it is convenient to use two rounds of AES as a building block. It not only means that we reach full bit diffusion, but also that a "free XOR" is available to add a round constant on Intel and AMD architectures.

It is nevertheless possible to consider GFS designs with an F-function that consists of only one AES round. A consequence of this design choice is that extra XOR instructions will be needed to introduce round constants, which could increase the cycle count. But this design choice also complicates the analysis. For example when $b = 2$, we find that 25 Feistel rounds are then needed to ensure at least 25 linearly active S-boxes. As shown in Fig. 3, this is because the tool can only ensure one active S-box for every Feistel round. Using two rounds of AES avoids this problem (see Fig. 4), and also significantly speeds up the tool: it makes bounding the minimum number of active S-boxes is rather easy, instead of becoming increasingly complicated for a reasonably large value of b.

Likewise, we could also consider designs with more than two AES rounds per F-function. In our experiments, we have not found any cases where this results in a design where the total number of AES rounds is smaller. The intuition is as follows: the number of Feistel rounds to reach full subblock diffusion is independent of the F-function, therefore adding more AES rounds to every F function is not expected to result in a better trade-off.

If we take another step back, we might consider to use other instructions besides `AESENC`. Clearly, `AESDEC` can be used as an alternative, and both the security properties and the benchmarks will remain the same. In fact, we use `AESDEC` when $b = 1$, to implement the inverse permutation. We do not use the `AESENCLAST` and `AESDECLAST` instructions, as they omit the `MixColumns` (resp. `MixColumns`) operation that crucial to the *wide trail design strategy* [30] of AES. We do, however, use only one `AESENCLAST` for the very last round of the $b = 1$ permutation, as this makes an efficient implementation of the inverse permutation possible on Intel architectures. This is equivalent to applying a linear transformation (`InvMixColumns`) to the output of the $b = 1$ permutation, therefore it does not reduce its cryptographic properties.

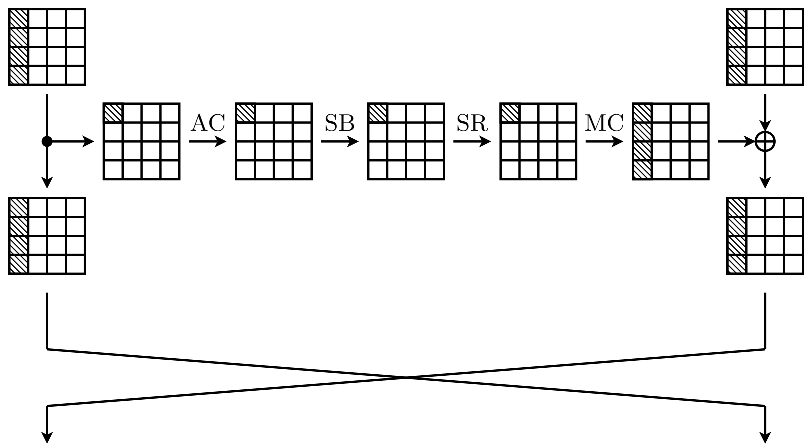

Fig. 3. A linear characteristic for an AES-based Feistel that uses only one round of AES inside its F-function. Crosshatches represent bytes with non-zero linear masks. The AES round consists of the `AddConstant` (AC), `SubBytes` (AC), `ShiftRows` (SR), and `MixColumns` (MC) operations. This round has only one active S-box. Therefore, 25 rounds are needed to ensure that there are least 25 linearly active S-boxes.

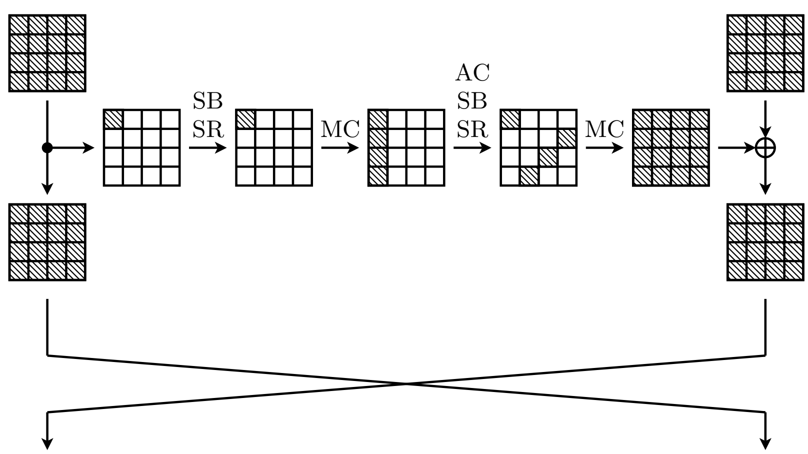

Fig. 4. A linear characteristic for one round of Simpira with $b = 2$ with 5 active S-boxes. Crosshatches represent bytes with non-zero linear masks. As Simpira uses two AES rounds per F-function, it can reach 25 active S-boxes in only 5 Feistel rounds, corresponding to 10 AES rounds in total.

Of course, it is possible to use non-AES instructions, possibly in combination with AES instructions. Actually, we do not need to be restricted to (generalized) Feistel designs for $b \geq 2$. However, such considerations are outside of the scope of this paper.

4 Specification of Simpira

An algorithmic specification of the Simpira design of Fig. 2 is given in Alg. 3-10. It uses one round of AES as a building block, which corresponds to the `AESENC` instruction on Intel processors (see Alg. 1). For byte ordering conventions and other implementation details, we refer to [44].

The F-function is specified in Alg. 2. It is parameterized by a counter c and by the number of subblocks b. Here, `SETR_EPI32` converts four 32-bit values into a 128-bit value, using the same byte ordering as the `_mm_setr_epi32()` compiler intrinsic.[5]

Both the input and output of Simpira consist of b blocks of 128 bits. The arrays use zero-based numbering, and array subscripts should be taken modulo the number of elements of the array. The subblock shuffle is done implicitly: we do not reorder the subblocks at the end of a Feistel round, but instead we apply the F-functions to other subblock inputs in the subsequent round. It is rather straightforward to implement the cyclic left shift in this way. For $b = 6$ and $b = 8$, the implementation of the subblock shuffle uses a decomposition into disjoint cycles.

As a result of this implementation choice, Simpira and its reduced-round variants are not always equivalent to a (generalized) Feistel with identical rounds. For example, for $b = 2$ the F-function is alternatingly applied from left to right and from right to left. When the number of rounds is odd, this is not equivalent to a Feistel with identical rounds: the two output subblocks will be swapped.

When $b = 1$, an extra `InvMixColumns` operation is applied to the output. This is equivalent to omitting the `MixColumns` operation in the last round, and is required to efficiently implement the inverse Simpira permutation using Intel's AES instructions. For details on how to efficiently implement both Simpira and Simpira^{-1} when $b = 1$, see Appendix B.

The design strategy of Simpira is intended to be very conservative. Because we think that the security of Simpira with very large b may not yet be well-understood, we recommend to use Simpira with $b \leq 65536$, corresponding to inputs of at most one megabyte. However, the external cryptanalysis of Simpira for any value of b is highly encouraged.

5 Benchmarks

We measured the performance of Simpira on the latest Intel processor, Architecture Codename Skylake. On this platform, the latency of `AESENC` is 4 cycles, and

[5] `_mm_setr_epi32(e3, e2, e1, e0)` yields a 128-bit destination: `dst[31:0] := e3 dst[63:32] := e2 dst[95:64] := e1 dst[127:96] := e0`

Algorithm 1 AESENC (see [44])

```
procedure AESENC(state, key)
    state ← SubBytes(state)
    state ← ShiftRows(state)
    state ← MixColumns(state)
    state ← state ⊕ key
    return state
end procedure
```

Algorithm 2 $F_{c,b}(x)$

```
procedure F_{c,b}(x)
    C ← SETR_EPI32(c, b, 0, 0)
    return AESENC(AESENC(x, C), 0)
end procedure
```

Algorithm 3 Simpira ($b \notin \{1, 6, 8\}$)

```
procedure SIMPIRA(x_0, ..., x_{b-1})
    if b ≤ 3 then
        R ← 6b + 3
    else
        R ← 6b − 9
    end if
    c ← 1

    for r = 0, ..., R − 1 do
        x_{r+1} ← x_{r+1} ⊕ F_{b,c}(x_r)
        c ← c + 1
        if b ≥ 4 then
            x_{r+2} ← x_{r+2} ⊕ F_{b,c}(x_{r+3})
            c ← c + 1
        end if
    end for
    return (x_0, x_1, ..., x_{b-1})
end procedure
```

Algorithm 4 Simpira^{-1} ($b \notin \{1, 6, 8\}$)

```
procedure SIMPIRA^{-1}(x_0, ..., x_{b-1})
    if b ≤ 3 then
        R ← 6b + 3
        c ← R
    else
        R ← 6b − 9
        c ← 2R
    end if
    for r = R − 1, ..., 0 do
        if b ≥ 4 then
            x_{r+2} ← x_{r+2} ⊕ F_{b,c}(x_{r+3})
            c ← c − 1
        end if
        x_{r+1} ← x_{r+1} ⊕ F_{b,c}(x_r)
        c ← c − 1
    end for
    return (x_0, x_1, ..., x_{b-1})
end procedure
```

Algorithm 5 Simpira ($b = 1$)

```
procedure SIMPIRA(x_0)
    R ← 6
    for c = 1, ..., R do
        x_0 ← F_{b,c}(x_0)
    end for
    InvMixColumns(x_0)
    return x_0
end procedure
```

Algorithm 6 Simpira^{-1} ($b = 1$)

```
procedure SIMPIRA(x_0)
    R ← 6
    MixColumns(x_0)
    for c = R, ..., 1 do
        x_0 ← F^{-1}_{b,c}(x_0)
    end for
    return x_0
end procedure
```

Algorithm 7 Simpira ($b = 6$)

procedure SIMPIRA($x_0, \ldots, x_5$)
  $R \leftarrow 15$
  $c \leftarrow 1$
  $s \leftarrow (0, 1, 2, 5, 4, 3)$
  for $r = 0, \ldots, R-1$ **do**
    $x_{s_{r+1}} \leftarrow x_{s_{r+1}} \oplus F_{b,c}(x_{s_r})$
    $c \leftarrow c + 1$
    $x_{s_{r+5}} \leftarrow x_{s_{r+5}} \oplus F_{b,c}(x_{s_{r+2}})$
    $c \leftarrow c + 1$
    $x_{s_{r+3}} \leftarrow x_{s_{r+3}} \oplus F_{b,c}(x_{s_{r+4}})$
    $c \leftarrow c + 1$
  end for
  return $(x_0, x_1, \ldots, x_5)$
end procedure

Algorithm 8 Simpira^{-1} ($b = 6$)

procedure SIMPIRA^{-1}($x_0, \ldots, x_5$)
  $R \leftarrow 15$
  $c \leftarrow 45$
  $s \leftarrow (0, 1, 2, 5, 4, 3)$
  for $r = R-1, \ldots, 0$ **do**
    $x_{s_{r+3}} \leftarrow x_{s_{r+3}} \oplus F_{b,c}(x_{s_{r+4}})$
    $c \leftarrow c - 1$
    $x_{s_{r+5}} \leftarrow x_{s_{r+5}} \oplus F_{b,c}(x_{s_{r+2}})$
    $c \leftarrow c - 1$
    $x_{s_{r+1}} \leftarrow x_{s_{r+1}} \oplus F_{b,c}(x_{s_r})$
    $c \leftarrow c - 1$
  end for
  return $(x_0, x_1, \ldots, x_5)$
end procedure

Algorithm 9 Simpira ($b = 8$)

procedure SIMPIRA($x_0, \ldots, x_7$)
  $R \leftarrow 18$
  $c \leftarrow 1$
  $s \leftarrow (0, 1, 6, 5, 4, 3)$
  $t \leftarrow (2, 7)$
  for $r = 0, \ldots, R-1$ **do**
    $x_{s_{r+1}} \leftarrow x_{s_{r+1}} \oplus F_{b,c}(x_{s_r})$
    $c \leftarrow c + 1$
    $x_{s_{r+5}} \leftarrow x_{s_{r+5}} \oplus F_{b,c}(x_{t_r})$
    $c \leftarrow c + 1$
    $x_{s_{r+3}} \leftarrow x_{s_{r+3}} \oplus F_{b,c}(x_{s_{r+4}})$
    $c \leftarrow c + 1$
    $x_{t_{r+1}} \leftarrow x_{t_{r+1}} \oplus F_{b,c}(x_{s_{r+2}})$
    $c \leftarrow c + 1$
  end for
  return $(x_0, x_1, \ldots, x_7)$
end procedure

Algorithm 10 Simpira^{-1} ($b = 8$)

procedure SIMPIRA^{-1}($x_0, \ldots, x_7$)
  $R \leftarrow 18$
  $c \leftarrow 72$
  $s \leftarrow (0, 1, 6, 5, 4, 3)$
  $t \leftarrow (2, 7)$
  for $r = R-1, \ldots, 0$ **do**
    $x_{t_{r+1}} \leftarrow x_{t_{r+1}} \oplus F_{b,c}(x_{s_{r+2}})$
    $c \leftarrow c - 1$
    $x_{s_{r+3}} \leftarrow x_{s_{r+3}} \oplus F_{b,c}(x_{s_{r+4}})$
    $c \leftarrow c - 1$
    $x_{s_{r+5}} \leftarrow x_{s_{r+5}} \oplus F_{b,c}(x_{t_r})$
    $c \leftarrow c - 1$
    $x_{s_{r+1}} \leftarrow x_{s_{r+1}} \oplus F_{b,c}(x_{s_r})$
    $c \leftarrow c - 1$
  end for
  return $(x_0, x_1, \ldots, x_7)$
end procedure

its throughput is 1 cycle. It follows that the software can be written in a way that fills the pipeline, by operating on four independent inputs. To obtain maximum throughput for all permutation sizes, we wrote functions that compute Simpira on four independent inputs. All Simpira permutations are benchmarked in the same setting, to makes the results comparable.

Note that when $b \geq 4$, Simpira uses two independent F-functions, which means that maximum throughput could already be reached with only two independent inputs. For $b = 8$, where Simpira has four independent F-functions, even a single-stream Simpira implementation would fill the pipeline.

The measurements are performed as follows. We benchmark a function that evaluates Simpira for four independent inputs, and computed the number of cycles to carry out 256 calls to this function, as a "unit." This provides us with the throughput of Simpira. The results were obtained by using the `RDTSCP` instruction, 250 repetitions as a "warmup" phase, averaging the measurement on subsequent 1000 runs. Finally, this experiment was repeated 30 times, and the best result was selected. The platform was set up with Hyperthreading and Turbo Boost disabled.

Table 1 shows the results obtained by our experiments. We present only benchmarks for the forward Simpira permutation; the benchmarks for Simpira^{-1} turned out to be very similar.

Table 1. Benchmarking results for the throughput of the Simpira permutations. For every b, we benchmark a function that applies the $128b$-bit permutation to four independent inputs. We give the number of cycles per input, as well as the overhead compared the theoretical optimum of performing only `AESENC` instructions.

b	bits	# `AESENC`	cycles	overhead
1	128	12	12	1.01
2	256	30	30	1.01
4	512	60	60	1.01
6	768	90	91	1.01
8	1,024	144	145	1.01
16	2,048	348	370	1.06
32	4,096	732	802	1.10
64	8,192	1,500	1,683	1.12
128	16,384	3,036	3,502	1.15
256	32,768	6,108	7,384	1.21

For comparison, we now provide the throughput of SHA-256, SHA-512, and Rijndael256 (with a 256-bit block size), measured on the same platform, and using the same methodology. In the case of SHA-256 and SHA-512, we wrote an optimized throughput-oriented implementation that uses the AVX2 architecture, available on the discussed platform. For SHA-256 and SHA-512, this implementation processes 4 and 8 independent (long) buffers respectively. For Rijndael256, we prepared optimized code that uses AES-NI (see details in [44]).

We measured is in ECB mode, operating on 8 blocks in parallel, to get the highest throughput possible on this platform.

Under this setup, the throughput of SHA-256, SHA-512, and Rijndael256 is 2.35, 3.13, and 1.54 cycles per byte, respectively. Therefore, for $b = 2$, it is clearly much faster to use the Simpira permutation, which requires only 0.94 cycles per byte. This permutation is to be used inside an Even-Mansour construction (for encryption), or with a Davies-Meyer feedforward (for hashing); but these operations not change the throughput in a noticeable way.

6 Cryptanalysis

The design criteria of Sect. 3 are not meant to be sufficient to guarantee security. In fact, it is not difficult to come up with trivially insecure constructions that satisfy (most of) the criteria. Rather, the design criteria are meant to assist us in identifying interesting constructions, which must then pass the scrutiny of cryptanalysis. Actually, during the design process of Simpira, we stumbled upon designs that were either insecure, or for which the security analysis was not so straightforward. When this happened, we adjusted the design criteria and repeated the search for constructions.

As such, we will not directly use the design criteria to argue the security of Simpira. Instead, we will use the fact that Simpira uses (generalized) Feistel structures and the AES round function, both of which have been extensively studied in literature. This allows us to focus our cryptanalysis efforts on the most promising attacks for this type of construction. We have tried to make this section easy to understand, which will hopefully convince the reader that Simpira should have a very comfortable security margin against all currently-known attacks.

Security claim. In what follows, we will only consider structural distinguishers [8] with a complexity up to 2^{128}. Simpira can be used in constructions that require a random permutation, however no statements can be made for adversaries that exceed 2^{128} queries. This type of security argument was first made by the SHA-3 [38] design team in response to high-complexity distinguishing attacks on the underlying permutation [19–21], and has since been reused for other permutation-based designs.

Symmetry attacks. As explained in Sect. 3, the round constants are meant to avoid symmetry inside a Simpira round, as well as symmetry between rounds. The round constants also depend on b, which means that Simpira permutations of different widths should be indistinguishable from each other. The round constants are generated by a simple counter: this not only makes the design easy to understand and to implement, but also avoids any concerns that the constants may contain a backdoor. Every F-function has a different round constant: this does not seem to affect performance on recent Intel platforms, but greatly reduces the probability that a symmetry property can be maintained over several rounds.

State collisions. For most block-cipher-based modes of operation, it is possible to define a "state," which is typically 128 bits long. This can be the chaining value for CBC mode, the counter for CTR mode, or the checksum in OCB. When a collision is found in this state, which is expected to happen around 2^{64} queries, the mode becomes insecure. For the Feistel-based Simpira ($b \geq 2$), there is no such concept of a "state." In fact: all subblocks receive an equal amount of processing after b rounds. This is ensured either by the cyclic left shift, or because every other round an F-function is applied to each subblock (for $b = 6$ and $b = 8$). This allows Simpira to reach security beyond 2^{64} queries after a sufficient amount of Feistel rounds.

Linear and differential cryptanalysis. Simpira's security argument against linear [12] and differential [60] cryptanalysis (up to attacks with complexity 2^{128}) is the same as the argument for AES, which is based on counting the number of active S-boxes. As explained in [31], four rounds of AES have at least 25 (linearly or differentially) active S-boxes. Then any four-round differential characteristic holds with a probability less than $2^{-6 \cdot 25} = 2^{-150}$, and any four-round linear characteristic holds with a correlation less than $2^{-3 \cdot 75} = 2^{-75}$.

Here, 2^{-6} refers to the maximum difference propagation probability, and 2^{-3} is the maximum correlation amplitude of the S-box used in AES. The aforementioned reasoning makes the common assumptions that the probabilities of every round of a characteristic can be multiplied, and that this leads to a good estimate for the probability of the characteristic, and also of the corresponding differential.

The number of rounds typically needs to be slightly higher to account for partial key guesses (for keyed constructions), and to have a reasonable security margin. For any of the Simpira designs, we have at least three times the number of rounds required to reach 25 active S-boxes. This should give a sizable security margin against linear and differential cryptanalysis, and even against more advanced variants such as saturation and integral cryptanalysis [29].

Boomerang and differential-linear cryptanalysis. Instead of using one long characteristic, boomerang [74] and differential-linear [11, 56] cryptanalysis combine two shorter characteristics. But even combined with partial key guesses, the fact that Simpira has at least three times the number of rounds that result in 25 active S-boxes, should be more than sufficient to protect against this type of attacks.

Truncated and impossible differential cryptanalysis. When full bit diffusion is not reached, it is easy to construct a truncated differential [52] characteristic with probability one. A common way to construct an impossible differential [9,10] is the *miss in the middle* approach. It combines two probability-one truncated differentials, whose conditions cannot be met together.

However, every Simpira variant has at least three times the number of rounds to reach full bit diffusion. This should not only prevent truncated and impossible differential attacks, but result in a satisfactory security margin against such attacks.

Meet-in-the-middle and rebound attacks. Meet-in-the-middle-attacks [34] separate the equations that describe a symmetric-key primitive into two or three groups. This is done in such a way that some variables do not appear into at least one of these groups. A typical rebound attack [62] also splits a cipher into three parts, and combines this with techniques from differential cryptanalysis. Low-probability differential characteristics satisfied by solving equations, and high-probability characteristics are satisfied in a probabilistic way.

With Simpira, splitting the construction in three parts will always result in one part that has at least 25 active S-boxes, or that reaches full bit diffusion. This should not only prevent meet-in-the-middle and rebound attacks, but also provide a large security margin against these attacks.

On Simpira with $b = 1$ (corresponding to 12-round AES with fixed round keys), the best known distinguisher is a rebound attack by Gilbert and Peyrin [42] that attacks 8 rounds out of 12.

Generic attacks. A substantial amount of literature exists on generic attacks of Feistel structures. In particular, we are interested in attacks in Maurer et al.'s indifferentiability setting [61], which is an extension of the indistinguishability notion for constructions that use publicly available oracles. In Simpira, the F-functions contain no secret key, and are therefore assumed to be publicly available.

Coron et al. [25] showed that five rounds of Feistel are not indifferentiable from a random permutation, and presented a indifferentiability proof for six rounds. Holenstein et al. [50] later showed that their proof is flawed, and provided a new indifferentiability proof for fourteen rounds. In very recent work, Dai and Steinberger [32] and independently Dachman-Soled et al. [28] announced an indifferentiability proof for the 10-round Feistel, which Dai and Steinberger subsequently improved to a proof for 8 rounds [33].

A problem with the aforementioned indifferentiability proofs is that they are rather weak: if the F-function is 128 bits wide, security is only proven up to about 2^{16} queries. The indistinguishability setting is better understood, where many proofs are available for not only Feistel, but also various generalized Feistel structures. But even in this setting, most proofs do not go beyond 2^{64} queries, and proving security with close to 2^{128} queries requires a very large number for rounds [49].

So although Simpira's Feistel-based permutations can be proven to be indistinguishable from random permutations using [63,79], it is an open problem to prove stronger security bounds for Simpira and other generalized Feistel structures. Nevertheless, no generic attacks are known for Simpira, even when up to 2^{128} are made.

Note that strictly speaking, there is an exception to the previous sentence for Simpira with $b = 1$. It is guaranteed to be an even permutation [22, Thm. 4.8], and therefore $2^{128} - 1$ queries can distinguish it from a random permutation with advantage 0.5. We only mention this for completeness; actually all of Simpira's permutations can be shown to be even, but this is typically not considered to be more than just a mathematical curiosity.

Other attacks. We do not consider brute-force-like attacks [70], such as the biclique attacks on AES [16]: they perform exhaustive search on a smaller number of rounds, and therefore do not threaten the practical security of the cipher. However, it will be interesting to investigate such attacks in future work, as they give an indication of the security of the cipher in the absence of other attacks. We also do not look into algebraic attacks, as AES seems to very resistant against such attacks.

7 Applications

Simpira can be used in various scenarios where AES does not permit an efficient construction with security up to 2^{128} evaluations of the permutation. We present a brief overview possible applications.

A block cipher without round keys. The (single-key) Even-Mansour construction [37, 39, 40] uses a secret key K to turn a plaintext P into a ciphertext C as follows:

$$C = E_K(P) = \pi(P \oplus K) \oplus K \ , \tag{1}$$

where π is an n-bit permutation. As argued by Dunkelman et al. [37], the construction is minimal, in the sense that simplifying it, for example by removing one of its components, will render it completely insecure. Mouha and Luykx [65] explained that the Even-Mansour is also minimal in the multi-key setting, where several keys are independently and uniformly drawn from the key space.

When D plaintext-ciphertexts are available, the secret key K of the Even-Mansour construction can be recovered in $2^n/D$ (off-line) evaluations of the permutation π [37]. This may be acceptable in lightweight authentication algorithms which rekey regularly, but may not be sufficient for encryption purposes [64, 65]. In order to achieve security up to about 2^{128} queries against all attacks in the multi-key setting, the Even-Mansour construction requires a permutation of at least 256 bits.

An important advantage of the Even-Mansour construction is that it avoids the need to precalculate round keys (and store them securely!) or to calculate them on the fly. But it also allows the easy construction of a tweakable block cipher. For a given tweak T, one can turn the Even-Mansour construction into a tweakable block cipher [57, 58]:

$$C = E_K(P) = \pi(P \oplus K \cdot T) \oplus K \cdot T \ , \tag{2}$$

that can be proven to be secure up to $2^{n/2}$ queries in the multi-key setting using the proof of [65,66]. For concreteness, we use the multiplication $K \cdot T$ in $GF(2^n)$, which restricts the tweaks to $T \neq 0$. However, any ϵ-AXU hash function can be used instead of this multiplication [23].

If the cipher is computed in a parallelizable mode of operation, independent blocks can be pipelined, and the performance would be dominated by Simpira with the relevant value of b, plus the overhead the key addition.

Permutation-based hashing. Achieving 128-bit collision resistance with a 128-bit permutation has been shown to be impossible [71]. Typically, a large permutation size is used to achieve a high throughput, for example 1600 bits in the sponge construction of SHA-3 [38]. The downside of using a large permutation is that performance is significantly reduced when many short messages need to be hashed, for example to compute a Lamport signature [55]. Simpira overcomes these problems by providing a family of efficient permutations with different input sizes.

In particular for hashing short messages, one may consider to use Simpira with a Davies-Meyer feed-forward: $\pi(x) \oplus x$. This construction has been shown to be optimally preimage and collision-resistant [14, 15], and even preimage aware [35], but not indifferentiable from a random oracle [24] as it is easy to find a fixed point: $\pi^{-1}(0)$. To match the intended application, padding of the input and/or truncation of the output of Simpira may be required.

Wide-block encryption and robust authenticated encryption. Wide-block encryption can be used to security against chosen ciphertext attacks when short (or even zero-length) authentication tags are used. In the context of full-disk encryption, there is usually no space to store an authentication tag. In an attempt to reduce the risk that ciphertext changes result in meaningful plaintext, a possibility is to use a wide block cipher to encrypt an entire disk sector, which typically has a size of 512 to 4096 bytes.

The same concern also exists when short authentication tags are used, and can be addressed by an encode-then-encipher approach [5]: add some bits of redundancy, and then encrypt with an arbitrary-input-length block cipher. Note that this technique achieves robust authenticated encryption [48].

Typical solutions for wide-block encryption such as the VIL [4], CMC [46] and EME [45, 47] modes of operation have the disadvantage that they are patented, and do not provide security beyond 2^{64} blocks of input. We are unaware of any patents related to Simpira.

When used in an Even-Mansour construction, Simpira with $b \geq 2$ can provide a wide block cipher that provides security up to 2^{128} blocks. When the block size exceeds the key size, the Even-Mansour construction can be generalized as follows:

$$C = E_K(P) = \pi(P \oplus (K \cdot T)\|0^*) \oplus (K \cdot T)\|0^*) \ , \quad (3)$$

where we set $T = 1$ if no tweak is provided. Note that this straightforward extension of the Even-Mansour construction appears in the proof for various sponge constructions. The first proof of security of this construction in the multi-key setting was given by Andreeva et al. [3].

8 Conclusion

We introduced Simpira, which is a family of cryptographic permutations that processes inputs of $128 \times b$ bits. It is intended to be a very conservative design that achieves high throughput on processors with AES instructions. We decided to use two rounds of AES as a building block, with the goal of simplifying the design space exploration, and making the cryptanalysis and implementation straightforward.

With this building block, we explored a large number of generalized Feistel structures, and calculated how many rounds are required to reach either full bit diffusion, or 25 linearly or differentially active S-boxes, whichever is greater. To ensure a large security margin, we multiplied this number of rounds by three. Of all designs that we considered, we selected the ones with the lowest amount of F-functions in total.

Following these design criteria, Simpira resulted in six different designs. For $b = 1$, we have AES with fixed round keys. Simpira uses a Feistel structure for $b = 2$, and a Type-1 GFS for $b = 3$. The $b \geq 4$ design corresponds to Yanagihara and Iwata's Type-1.x (b,2) GFS, which uses two F-functions per round. For $b = 6$ and $b = 8$, we use Suzaki and Minematsu's improved Type-2 GFS, as it has fewer F-functions than the Type-1.x (b,2) GFS.

The design criteria were only intended to quickly identify promising designs, and are not by themselves sufficient to guarantee a secure construction. We analyzed the security of Simpira against a wide variety of attacks, based on the large amount of literature that exists on AES and on generalized Feistel structures. We found that Simpira has a very comfortable security margin against currently-known attacks.

Our benchmarks on Intel Skylake showed that Simpira is close to the theoretical optimum of only executing `AESENC` instructions. For $b \leq 32$, corresponding to inputs of up to 512 bytes, Simpira is less than 10% away from this optimum. This justifies our assumption that the number of F-functions is a good metric for the throughput of the implementation.

It is unfortunate that many methods to encrypt wide input blocks, such as VIL, CMC, and EME, have not seen widespread adoption. The main obstacle appears to be that they are patented. We hope that Simpira can provide an interesting alternative: it is not only free from patent concerns, but offers security way beyond the 2^{64} limit for typical AES-based modes.

Acknowledgments. The authors would like to thank the organizers and participants of Dagstuhl Seminar 16021, where an early version of this work was presented. The detailed comments and suggestions of the seminar participants helped to improve this manuscript significantly. This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007), by Research Fund KU Leuven, OT/13/071, by the PQCRYPTO project, which was partially funded by the European Commission Horizon 2020 research Programme, grant #645622, and by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011, and the BRUTUS project under Contract ANR-14-CE28-0015. Nicky Mouha is supported by a Postdoctoral Fellowship from the Flemish Research Foundation (FWO-Vlaanderen), and by FWO travel grant 12F9714N.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald and Fischlin [69], pp. 430–454
2. Anderson, R.J., Biham, E.: Two Practical and Provably Secure Block Ciphers: BEARS and LION. In: Gollmann, D. (ed.) Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings. LNCS, vol. 1039, pp. 113–120. Springer (1996)
3. Andreeva, E., Daemen, J., Mennink, B., Assche, G.V.: Security of Keyed Sponge Constructions Using a Modular Proof Approach. In: Leander, G. (ed.) Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers. LNCS, vol. 9054, pp. 364–384. Springer (2015)
4. Bellare, M., Rogaway, P.: On the Construction of Variable-Input-Length Ciphers. In: Knudsen [53], pp. 231–244
5. Bellare, M., Rogaway, P.: Encode-Then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. In: Okamoto [68], pp. 317–330
6. Berger, T.P., Minier, M., Thomas, G.: Extended Generalized Feistel Networks Using Matrix Representation. In: Lange, T., Lauter, K.E., Lisonek, P. (eds.) Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. LNCS, vol. 8282, pp. 289–305. Springer (2013)
7. Berger, T., Francq, J., Minier, M., Thomas, G.: Extended Generalized Feistel Networks using Matrix Representation to Propose a New Lightweight Block Cipher: Lilliput. Computers, IEEE Transactions on Preprint, 1–15 (2015)
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic Sponge Functions, available at `http://sponge.noekeon.org/CSF-0.1.pdf`
9. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. LNCS, vol. 1592, pp. 12–23. Springer (1999)
10. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. J. Cryptology 18(4), 291–311 (2005)

11. Biham, E., Dunkelman, O., Keller, N.: Enhancing Differential-Linear Cryptanalysis. In: Zheng, Y. (ed.) Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings. LNCS, vol. 2501, pp. 254–266. Springer (2002)
12. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. J. Cryptology 4(1), 3–72 (1991)
13. Biryukov, A., Khovratovich, D.: PAEQ: Parallelizable Permutation-Based Authenticated Encryption. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S. (eds.) Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings. LNCS, vol. 8783, pp. 72–89. Springer (2014)
14. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung [78], pp. 320–335
15. Black, J., Rogaway, P., Shrimpton, T., Stam, M.: An Analysis of the Blockcipher-Based Hash Functions from PGV. J. Cryptology 23(4), 519–545 (2010)
16. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT. LNCS, vol. 7073, pp. 344–371. Springer (2011)
17. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. LNCS, vol. 6917, pp. 312–325. Springer (2011)
18. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: SPONGENT: The Design Space of Lightweight Cryptographic Hashing. IEEE Trans. Computers 62(10), 2041–2053 (2013)
19. Boura, C., Canteaut, A.: A zero-sum property for the KECCAK-f permutation with 18 rounds. In: ISIT. pp. 2488–2492. IEEE (2010)
20. Boura, C., Canteaut, A.: Zero-Sum Distinguishers for Iterated Permutations and Application to KECCAK-f and Hamsi-256. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography. LNCS, vol. 6544, pp. 1–17. Springer (2010)
21. Boura, C., Canteaut, A., Cannière, C.D.: Higher-order differential properties of KECCAK and *Luffa*. IACR Cryptology ePrint Archive 2010, 589 (2010)
22. Cid, C., Murphy, S., Robshaw, M.J.B.: Algebraic aspects of the advanced encryption standard. Springer (2006)
23. Cogliati, B., Lampe, R., Seurin, Y.: Tweaking Even-Mansour Ciphers. In: Gennaro and Robshaw [41], pp. 189–208
24. Coron, J., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In: Shoup, V. (ed.) Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. LNCS, vol. 3621, pp. 430–448. Springer (2005)
25. Coron, J., Patarin, J., Seurin, Y.: The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In: Wagner, D. (ed.) Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings. LNCS, vol. 5157, pp. 1–20. Springer (2008)
26. Cossíos, D.: Breve Bestiario Peruano. Editorial Casatomada, second edn. (2008)
27. Crowley, P.: Mercy: A Fast Large Block Cipher for Disk Sector Encryption. In: Schneier, B. (ed.) Fast Software Encryption, 7th International Workshop, FSE

2000, New York, NY, USA, April 10-12, 2000, Proceedings. LNCS, vol. 1978, pp. 49–63. Springer (2000)
28. Dachman-Soled, D., Katz, J., Thiruvengadam, A.: 10-Round Feistel is Indifferentiable from an Ideal Cipher. Cryptology ePrint Archive, Report 2015/876 (2015), http://eprint.iacr.org/
29. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. LNCS, vol. 1267, pp. 149–165. Springer (1997)
30. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings. LNCS, vol. 2260, pp. 222–238. Springer (2001)
31. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
32. Dai, Y., Steinberger, J.: Indifferentiability of 10-Round Feistel Networks. Cryptology ePrint Archive, Report 2015/874 (2015), http://eprint.iacr.org/
33. Dai, Y., Steinberger, J.: Indifferentiability of 8-Round Feistel Networks. Cryptology ePrint Archive, Report 2015/1069 (2015), http://eprint.iacr.org/
34. Diffie, W., Hellman, M.E.: Exhaustive Cryptanalysis of the NBS Data Encryption Standard. Computer 10(6), 74–84 (1977)
35. Dodis, Y., Ristenpart, T., Shrimpton, T.: Salvaging Merkle-Damgård for Practical Applications. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. LNCS, vol. 5479, pp. 371–388. Springer (2009)
36. Dunkelman, O. (ed.): Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers, LNCS, vol. 5665. Springer (2009)
37. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT. LNCS, vol. 7237, pp. 336–354. Springer (2012)
38. Dworkin, M.J.: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Inf. Process. Stds. (NIST FIPS) - 202 (August 2015)
39. Even, S., Mansour, Y.: A Construction of a Cipher From a Single Pseudorandom Permutation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT. LNCS, vol. 739, pp. 210–224. Springer (1991)
40. Even, S., Mansour, Y.: A Construction of a Cipher from a Single Pseudorandom Permutation. J. Cryptology 10(3), 151–162 (1997)
41. Gennaro, R., Robshaw, M. (eds.): Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, LNCS, vol. 9215. Springer (2015)
42. Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In: Hong and Iwata [51], pp. 365–383
43. Gueron, S.: Intel's New AES Instructions for Enhanced Performance and Security. In: Dunkelman [36], pp. 51–66
44. Gueron, S.: Intel® Advanced Encryption Standard (AES) New Instructions Set. Available at: https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set (September 2012), Revision 3.01

45. Halevi, S.: EME*: Extending EME to Handle Arbitrary-Length Messages with Associated Data. In: Canteaut, A., Viswanathan, K. (eds.) Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings. LNCS, vol. 3348, pp. 315–327. Springer (2004)
46. Halevi, S., Rogaway, P.: A Tweakable Enciphering Mode. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. LNCS, vol. 2729, pp. 482–499. Springer (2003)
47. Halevi, S., Rogaway, P.: A Parallelizable Enciphering Mode. In: Okamoto, T. (ed.) Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings. LNCS, vol. 2964, pp. 292–304. Springer (2004)
48. Hoang, V.T., Krovetz, T., Rogaway, P.: Robust Authenticated-Encryption AEZ and the Problem That It Solves. In: Oswald and Fischlin [69], pp. 15–44
49. Hoang, V.T., Rogaway, P.: On Generalized Feistel Networks. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. LNCS, vol. 6223, pp. 613–630. Springer (2010)
50. Holenstein, T., Künzler, R., Tessaro, S.: The equivalence of the random oracle model and the ideal cipher model, revisited. In: Fortnow, L., Vadhan, S.P. (eds.) Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011. pp. 89–98. ACM (2011)
51. Hong, S., Iwata, T. (eds.): Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers, LNCS, vol. 6147. Springer (2010)
52. Knudsen, L.R.: Truncated and Higher Order Differentials. In: Preneel, B. (ed.) Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings. LNCS, vol. 1008, pp. 196–211. Springer (1994)
53. Knudsen, L.R. (ed.): Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings, LNCS, vol. 1636. Springer (1999)
54. 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://eprint.iacr.org/`
55. Lamport, L.: Constructing Digital Signatures from a One Way Function. Tech. Rep. SRI-CSL-98, SRI International Computer Science Laboratory (October 1979)
56. Langford, S.K., Hellman, M.E.: Differential-Linear Cryptanalysis. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings. LNCS, vol. 839, pp. 17–25. Springer (1994)
57. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In: Yung [78], pp. 31–46
58. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. J. Cryptology 24(3), 588–613 (2011)
59. Lucks, S.: BEAST: A Fast Block Cipher for Arbitrary Blocksizes. In: Horster, P. (ed.) Communications and Multimedia Security II, Proceedings of the IFIP TC6/TC11 International Conference on Communications and Multimedia Security at Essen, Germany, 23-24 September 1996. IFIP Conference Proceedings, vol. 70, pp. 144–153. Chapman & Hall (1996)
60. Matsui, M.: Linear Cryptoanalysis Method for DES Cipher. In: EUROCRYPT. pp. 386–397 (1993)

61. Maurer, U.M., Renner, R., Holenstein, C.: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: Naor, M. (ed.) Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings. LNCS, vol. 2951, pp. 21–39. Springer (2004)
62. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: Dunkelman [36], pp. 260–276
63. Moriai, S., Vaudenay, S.: On the Pseudorandomness of Top-Level Schemes of Block Ciphers. In: Okamoto [68], pp. 289–302
64. Mouha, N.: The Design Space of Lightweight Cryptography. Cryptology ePrint Archive, Report 2015/303 (2015), `http://eprint.iacr.org/`
65. Mouha, N., Luykx, A.: Multi-key Security: The Even-Mansour Construction Revisited. In: Gennaro and Robshaw [41], pp. 209–223
66. Mouha, N., Mennink, B., Herrewege, A.V., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers. In: Joux, A., Youssef, A.M. (eds.) Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers. LNCS, vol. 8781, pp. 306–323. Springer (2014)
67. 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.) Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. LNCS, vol. 7537, pp. 57–76. Springer (2011)
68. Okamoto, T. (ed.): Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings, LNCS, vol. 1976. Springer (2000)
69. Oswald, E., Fischlin, M. (eds.): Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, LNCS, vol. 9056. Springer (2015)
70. Rechberger, C.: On Bruteforce-Like Cryptanalysis: New Meet-in-the-Middle Attacks in Symmetric Cryptanalysis. In: Kwon, T., Lee, M., Kwon, D. (eds.) Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers. LNCS, vol. 7839, pp. 33–36. Springer (2012)
71. Rogaway, P., Steinberger, J.P.: Security/Efficiency Tradeoffs for Permutation-Based Hashing. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. LNCS, vol. 4965, pp. 220–236. Springer (2008)
72. Schroeppel, R.: The Hasty Pudding Cipher – A Tasty Morsel. Available at: `http://richard.schroeppel.name:8015/hpc/` (1998), submission to the NIST AES competition
73. Suzaki, T., Minematsu, K.: Improving the Generalized Feistel. In: Hong and Iwata [51], pp. 19–39
74. Wagner, D.: The Boomerang Attack. In: Knudsen [53], pp. 156–170
75. Yanagihara, S., Iwata, T.: On Permutation Layer of Type 1, Source-Heavy, and Target-Heavy Generalized Feistel Structures. In: Lin, D., Tsudik, G., Wang, X. (eds.) Cryptology and Network Security - 10th International Conference, CANS 2011, Sanya, China, December 10-12, 2011. Proceedings. LNCS, vol. 7092, pp. 98–117. Springer (2011)

76. Yanagihara, S., Iwata, T.: Improving the Permutation Layer of Type 1, Type 3, Source-Heavy, and Target-Heavy Generalized Feistel Structures. IEICE Transactions 96-A(1), 2–14 (2013)
77. Yanagihara, S., Iwata, T.: Type 1.x Generalized Feistel Structures. IEICE Transactions 97-A(4), 952–963 (2014)
78. Yung, M. (ed.): Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings, LNCS, vol. 2442. Springer (2002)
79. Zheng, Y., Matsumoto, T., Imai, H.: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: Brassard, G. (ed.) Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. LNCS, vol. 435, pp. 461–480. Springer (1989)

A Differential and Linear Characteristics for $b = 6$

For $b = 6$, two differential characteristics that were output by the MILP tool are shown in Fig. 5-6, and two linear characteristics are shown in Fig. 7-8. Each of these characteristics have 30 (linearly or differentially) active S-boxes. The tool proved that no characteristics exist with fewer active S-boxes.

As the linear characteristics of Fig. 7-8 show, removing either the first or the last round (respectively) would result in a characteristic with only 10 linearly active S-boxes. Therefore, reducing the number of rounds would clearly violate the requirement that there must be at least 25 active S-boxes.

B Efficient Implementation For $b = 1$

We recall the four Intel instructions to implement one round of AES: `AESENC` (Alg. 11), `AESENCLAST` (Alg. 12), `AESDEC` (Alg. 13), and `AESENCLAST` (Alg. 14). The `AESIMC` instruction corresponds to the `InvMixColumns` operation. Then for $b = 1$, Simpira can be implemented as in Alg. 15, and Simpira^{-1} as in Alg. 16.

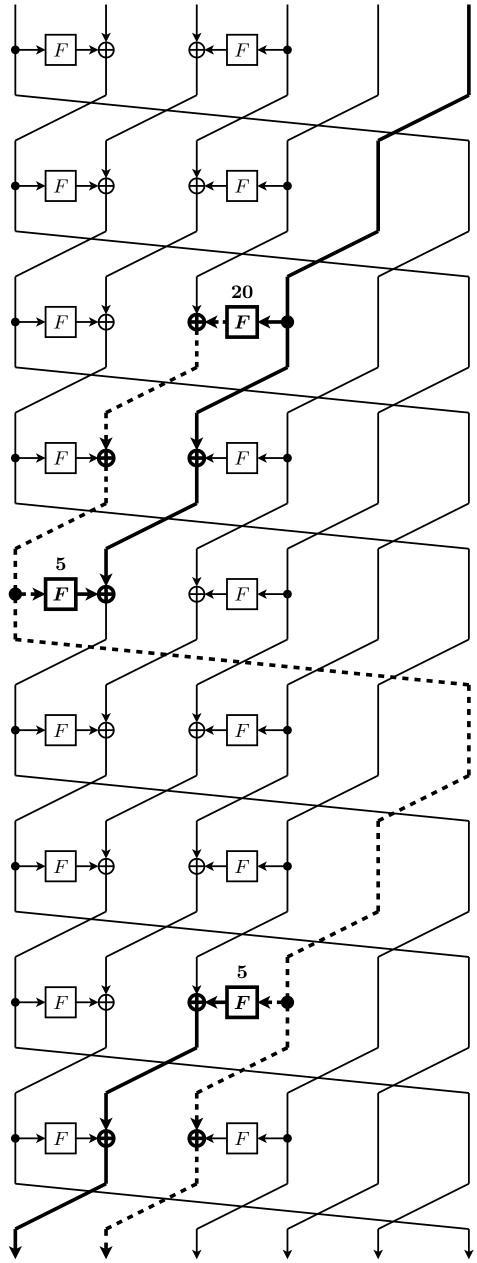

Fig. 5. A differential characteristic for $b = 6$ with 30 active S-boxes. A thick full line indicates a difference in every byte, a thick dotted line refers to a difference in only one byte – it does not matter which one. A normal line indicates that no difference is present. When non-zero, the number of active S-boxes is shown above every F-function.

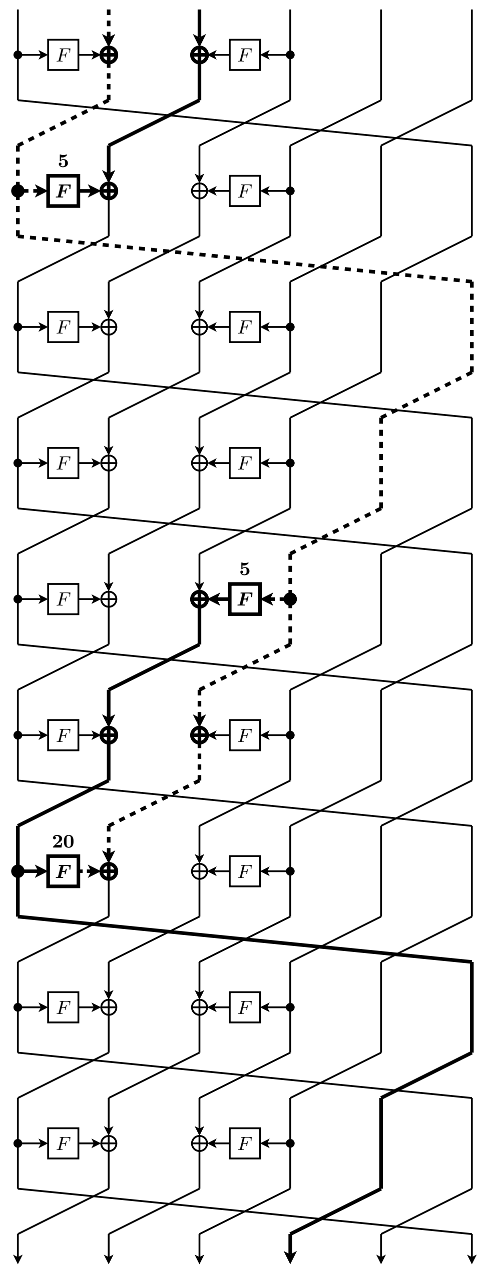

Fig. 6. Another differential characteristic for $b = 6$ with 30 active S-boxes. A thick full line indicates a difference in every byte, a thick dotted line refers to a difference in only one byte – it does not matter which one. A normal line indicates that no difference is present. When non-zero, the number of active S-boxes is shown above every F-function.

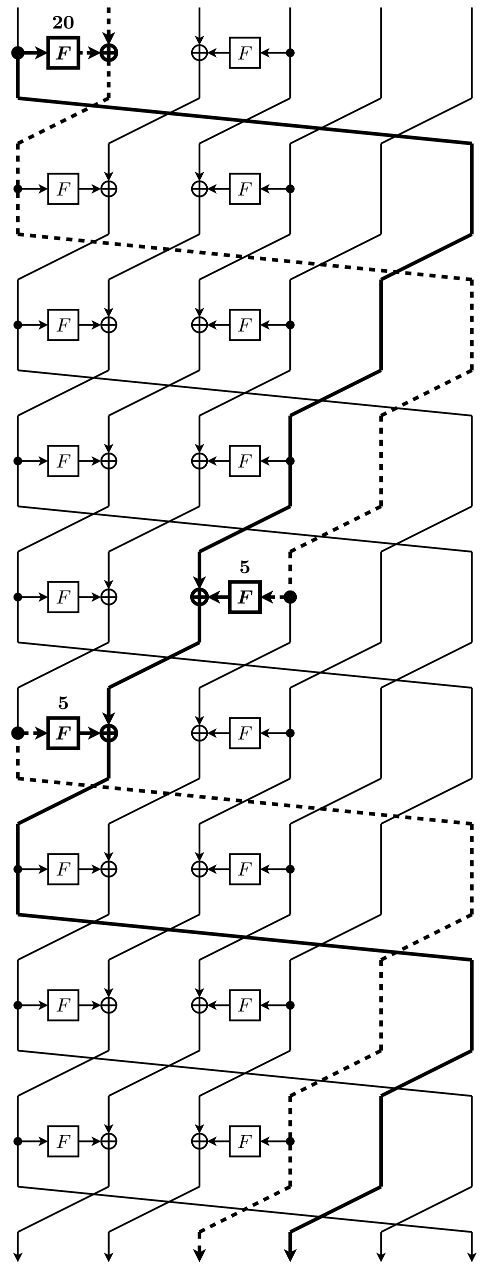

Fig. 7. A linear characteristic for $b = 6$ with 30 active S-boxes. A thick full line indicates a non-zero linear mask in every byte, a thick dotted line refers to a non-zero linear mask in only one byte – it does not matter which one. A normal line indicates that the linear masks of every byte are zero. When non-zero, the number of active S-boxes is shown above every F-function.

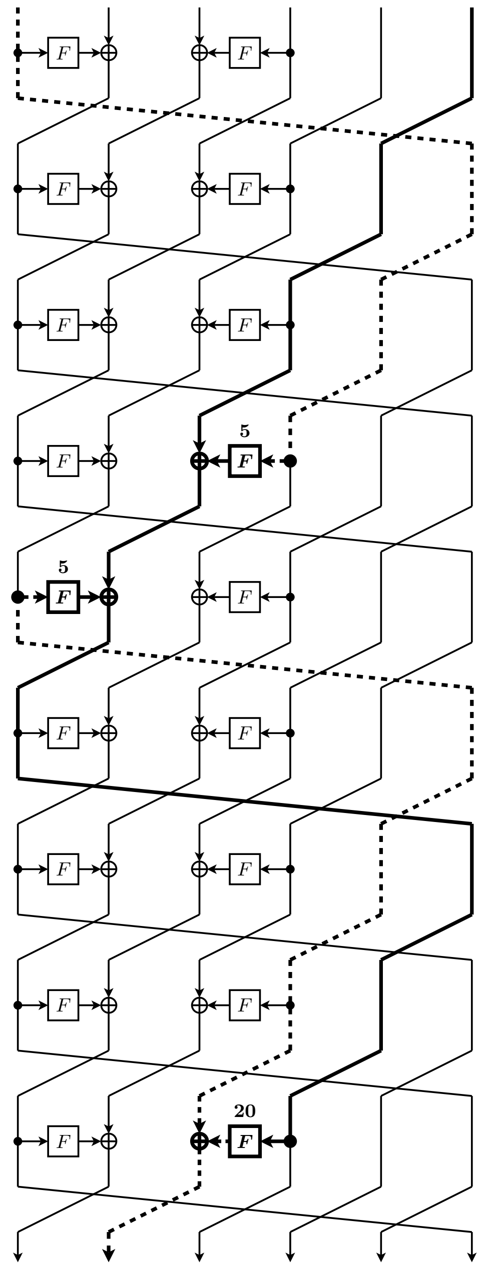

Fig. 8. Another linear characteristic for $b = 6$ with 30 active S-boxes. A thick full line indicates a non-zero linear mask in every byte, a thick dotted line refers to a non-zero linear mask in only one byte – it does not matter which one. A normal line indicates that the linear masks of every byte are zero. When non-zero, the number of active S-boxes is shown above every F-function.

Algorithm 11 `AESENC` (= Alg. 1)

procedure `AESENC`(state, key)
  state ← `SubBytes`(state)
  state ← `ShiftRows`(state)
  state ← `MixColumns`(state)
  state ← state ⊕ key
  return state
end procedure

Algorithm 12 `AESENCLAST`

procedure `AESENCLAST`(state, key)
  state ← `SubBytes`(state)
  state ← `ShiftRows`(state)
  state ← state ⊕ key
  return state
end procedure

Algorithm 13 `AESDEC`

procedure `AESDEC`(state, key)
  state ← `InvSubBytes`(state)
  state ← `InvShiftRows`(state)
  state ← `InvMixColumns`(state)
  state ← state ⊕ key
  return state
end procedure

Algorithm 14 `AESDECLAST`

procedure `AESDECLAST`(state, key)
  state ← `InvSubBytes`(state)
  state ← `InvShiftRows`(state)
  state ← state ⊕ key
  return state
end procedure

Algorithm 15 Simpira ($b = 1$) (= Alg. 5)

procedure SIMPIRA(x_0)
  $R \leftarrow 6$
  for $r = 1, \ldots, R-1$ **do**
    $C \leftarrow$ `SETR_EPI32`$(r, R, 0, 0)$
    $x_0 \leftarrow$ `AESENC`(x, C)
    $x_0 \leftarrow$ `AESENC`$(x, 0)$
    $c \leftarrow c + 1$
  end for
  $C \leftarrow$ `SETR_EPI32`$(R, R, 0, 0)$
  $x_0 \leftarrow$ `AESENC`(x, C)
  $x_0 \leftarrow$ `AESENCLAST`$(x, 0)$
  return x_0
end procedure

Algorithm 16 Simpira^{-1} ($b = 1$) (= Alg. 6)

procedure SIMPIRA(x_0)
  $R \leftarrow 6$
  for $r = R, \ldots, 1$ **do**
    $C \leftarrow$ `SETR_EPI32`$(r, R, 0, 0)$
    $C \leftarrow$ `AESIMC`(C)
    $x_0 \leftarrow$ `AESDEC`(x, C)
    $x_0 \leftarrow$ `AESDEC`$(x, 0)$
    $c \leftarrow c + 1$
  end for
  $C \leftarrow$ `SETR_EPI32`$(1, R, 0, 0)$
  $C \leftarrow$ `AESIMC`(C)
  $x_0 \leftarrow$ `AESDEC`(x, C)
  $x_0 \leftarrow$ `AESDECLAST`$(x, 0)$
  return x_0
end procedure