Farfalle: parallel permutation-based cryptography

Guido Bertoni¹, Joan Daemen¹,², Seth Hoffert,
Michaël Peeters¹, Gilles Van Assche¹, and Ronny Van Keer¹

¹ STMicroelectronics
² Radboud University

Abstract. In this paper, we introduce Farfalle, a new permutation-based construction for building a pseudorandom function (PRF). The PRF takes as input a key and a sequence of arbitrary-length data strings, and returns an arbitrary-length output. It has a compression layer and an expansion layer, each involving the parallel application of a permutation. The construction also makes use of LFSR-like rolling functions for generating input and output masks and for updating the inner state during expansion. On top of the inherent parallelism, Farfalle instances can be very efficient because the construction imposes less requirements on the underlying primitive than, e.g., the duplex construction or typical block cipher modes. Farfalle has an incremental property: compression of common prefixes of inputs can be factored out. Thanks to its input-output characteristics, Farfalle is really versatile. We specify simple modes on top of it for authentication, encryption and authenticated encryption, as well as a wide block cipher mode. As a showcase, we present Krãoável, a very efficient instance of Farfalle based on Keccak-p[1600, nr] permutations and formulate concrete security claims against classical and quantum adversaries. The permutations in the compression and expansion layers of Krãoável have only 6 and 4 rounds respectively and the rolling function is lightweight. We provide a rationale for our choices and report on software performance.

Keywords: pseudorandom function, permutation-based crypto, Keccak

1 Introduction

Until recently, symmetric cryptography was dominated by block ciphers. With the exception of some dedicated stream ciphers, standards and commercial products performed encryption, authentication, authenticated encryption and even hashing with block-cipher-based modes. After the introduction of the AES [22], a lot of innovation has been brought to block-cipher-based cryptography in the form of the definition of new modes, particularly for authentication and authenticated encryption.

Upon closer inspection of these modes, one can see a tendency towards modes that do not use the inverse mapping of the block cipher. Remarkably, the support for this inverse mapping imposes a separation of the processing of the $b = k + n$ bits of the input, with $k$ the key length and $n$ the block size. Namely, the key is processed in a key schedule and the data in the data path, and there can be no diffusion from the data path to the key schedule. This strongly limits the potential diffusion as illustrated by the birthday bounds in both block size and key length. So we think using a block cipher in a mode that does not use its inverse is a waste. Well-known examples of block-cipher based functions that do not use the inverse mapping are the NIST hash function standards SHA-1 and SHA-2 [13,16] but also counter mode encryption and any block-cipher based authentication.

As opposed to block ciphers, cryptographic permutations do not make a distinction between key and data input and hence do not suffer from this limitation. Their use gained popularity in the years 2000, in particular during the SHA-3 competition, as several candidates were based on this type of primitive. Furthermore, the selection of the permutation-based Keccak sponge function as the SHA-3 standard gave increased visibility to this type
of cryptographic primitive \[10,47\]. We introduced the concept of sponge functions in 2007, initially aimed at hashing, although soon after we developed modes for efficient encryption, authentication and authenticated encryption. For the latter we had to do some refactoring, leading to the (full-state) keyed duplex construction \[21\]. Today, sponge-based cryptography has become a full-fledged alternative to its block-cipher-based counterpart.

Yet, the sponge and duplex constructions are inherently serial. The bulk of the computation consists of the repeated evaluation of a permutation, and for every evaluation we need the output of the previous one. Modern high-end CPUs are so powerful that evaluating \( n \) permutations simultaneously is faster than evaluating them in sequence. For hashing, optimal performances can be obtained by using tree hashing. A concrete example of a hash function that can exploit a high degree of parallelism is KangarooTwelve \[14\]. For separate authentication or encryption, similar techniques can be applied. However, for duplex-based authenticated encryption, the amount of available parallelism must be known in advance. An example of a mode for authenticated encryption that supports configurable parallelism is Motorist, the mode underlying Keyak \[13\].

Also, the sponge and duplex constructions limit the number of output bits per evaluation of the permutation to \( r = b - c \), with \( b \) the width of the permutation and \( c \) a parameter, called the capacity, that determines the security strength. Depending on the mode of use and the adversary’s capabilities, in order to achieve \( s \) bits of security, one must take \( c \) somewhere between \( s \) and \( 2s \). Consequently, for a security strength of 128 bits, the rate \( r \) is 128 to 256 bits smaller than the permutation width. This restricts the underlying permutation to have some minimum width, and for relatively small widths the efficiency is not optimal.

To address these concerns, we set out to define a parallelizable counterpart for sponge-based cryptography. The ambition quickly became to have permutation-based modes for all keyed operations in symmetric cryptography that can exploit arbitrary parallelism and that can make use of permutations as small as the birthday bound \((b = 2s)\). We called the result Farfalle.

Similarly to sponges, the Farfalle offering is built around a (composite) primitive and modes on top of it. This primitive is a pseudorandom function (PRF) that takes as input a key and a string (or a sequence of strings), and produces an arbitrary-length output. To an adversary not knowing the key, these output bits look like independent uniformly-drawn random bits. Such a PRF is a powerful primitive that can readily be used as a message authentication codes (MAC), a stream cipher or a key derivation function. With some very simple modes, one can turn it into an authenticated encryption scheme and even a block cipher supporting variable block length.

In more details, Farfalle builds a PRF from a \( b \)-bit cryptographic permutation, or a family of permutations possibly with different number of rounds. The constructed PRF takes as input a key and a sequence of arbitrary-length data strings, and it generates an arbitrary-length output. It consists of a mask derivation, a compression layer and an expansion layer, each of them involving the parallel application of a permutation. The compression layer applies one of the permutations to input blocks, each blinded with a rolling \( b \)-bit input mask, and it (bitwise) adds their outputs in a \( b \)-bit accumulator. For the expansion layer, the accumulator is subject to a permutation and then used as a rolling state to generate the output. Each output block is the sum of the (rolling) output mask and the result of a permutation applied to the rolling state.
We depict the compression and expansion stages of Farfalle in Figure 1. The construction aims for simplicity and efficiency and has some features in common with the sponge construction [1]. As in sponges, the inverse of the permutation is not used. Another interesting feature it has in common with sponges is the ability to compute it for incremental inputs. While in sponges this is modeled by the duplex construction [8], in Farfalle this is achieved by the fact that the contribution of input strings to the accumulator only depends on their value and position in the input.

Farfalle can be seen as a parallelizable counterpart of the sponge for keyed applications. In particular, its permutation calls can be performed in parallel as soon as the input masks have been generated. This can be exploited on many platforms, including on modern processors with single-instruction multiple-data (SIMD) units. Moreover, it can be made very efficient as the number of rounds in the permutations can be taken much smaller than in sponge-based modes, thanks to the fact that in Farfalle an adversary never has access to both the input and the output of a permutation call.

Thanks to its input-output characteristics, Farfalle is really versatile. We specify concrete modes on top of it for authentication, encryption and authenticated encryption, as well as a wide block cipher mode.

Farfalle can be instantiated with any cryptographic permutation. In particular, we instantiate it with the Keccak-p permutations and with a rolling function similar to those proposed in [28], attach concrete security claims to it and call the result Kravatte. Reference and optimized code for Kravatte will be soon made available in KeccakTools and in the Keccak code package, respectively [11,16].

1.1 Overview of the paper

After introducing our notation and the main components of Farfalle in Section 2, we specify the Farfalle construction in Section 3. In Section 4 we define (authenticated) encryption modes: a session-supporting mode for authenticated encryption (AE), a synthetic initial value (SIV) AE mode and a wide block cipher. Section 5 gives a rationale for the basic construction and Section 6 discusses some prior art for Farfalle and its modes. Finally, in Section 7 we specify Kravatte, a concrete instance of Farfalle making use of Keccak-p, the permutation underlying Keccak, make a security claim and provide some rationale.

Appendix A is dedicated to an analysis of linear rolling functions that can be used in conjunction with any permutation with low algebraic degree. In Appendix B we list the changes since the preliminary version of Farfalle.

2 Notation and components

In this section we introduce notation related to strings and the two types of functions used in Farfalle: permutations and rolling functions.

2.1 Strings

Farfalle operates on strings of bits, that we will just call strings in the following. Inside Farfalle, strings are processed in chunks of $b$ bits, where $b$ is the width of the underlying permutations. We use uppercase characters for arbitrary-length strings and lowercase characters for $b$-bit strings.
We denote the length of a string \( X \) by \( |X| \). The set of strings of length \( n \) is \( \mathbb{Z}_2^n \) and the set of strings of any length is \( \mathbb{Z}_2^* \).

When applied to strings, the + operator is the bitwise addition, a.k.a. modulo 2 addition or exclusive-or (XOR). The \( \| \) operator is the concatenation.

Converting an input string to an array of \( b \)-bit strings requires a padding rule. We use simple padding that appends a single 1-bit followed by the minimum number of 0-bits resulting in a string with a length that is a multiple of \( b \). We write \( P = \text{pad}10^*(M) \). The string \( P \) can be seen as an array of \( b \)-bit blocks \( p_i \). It is convenient to have index \( i \) start from a value of our choice \( I \). We denote this as \( P = p_I, p_{I+1} \ldots p_{I+n-1} \), with \( n \) denoting the number of \( n \)-bit blocks in \( P \).

A sequence of \( m \) strings \( M^{(0)} \) to \( M^{(m-1)} \) is denoted \( M^{(m-1)} \circ \ldots \circ M^{(1)} \circ M^{(0)} \). The notation deliberately reminds of the composition of functions. The set of all sequences of at least one string is \((\mathbb{Z}_2^*)^+\).

Farfalle yields a PRF that returns a string of arbitrary output length. In our notation, we let the length of this string be determined by the context. In particular, \( X + F_K(M) \) means that the required output length of \( F_K(M) \) is \( |X| \). In some cases it is convenient to skip the first \( q \) bits of the output. For this we use the notation \( X + F_K(M) \ll q \) to indicate that we take the output starting from offset \( q \), i.e., the bits \( q \) to \( q + |X| - 1 \) of the output produced by \( F_K(M) \).

### 2.2 Permutations

Farfalle makes use of four cryptographic permutations, each operating on \( b \)-bit strings:

- \( p_b \) for deriving the initial mask from the key \( K \)
- \( p_c \) in the compression layer
- \( p_d \) between the compression and expansion layer
- \( p_e \) in the expansion layer

Specific security requirements apply for each of them, see Section 5. It is however not a security requirement that they are different. One may specify instances of Farfalle where the four permutations are the same. We think the way to instantiate the most efficient Farfalle instances is by taking a permutation with variable number of rounds and tuning the rounds for the four different cases to optimize the ratio efficiency vs. safety margin.

### 2.3 Rolling functions

A rolling function, denoted as roll, is a permutation of \( \mathbb{Z}_2^b \). Farfalle makes use of three rolling functions, each operating on \( b \)-bit strings:

- \( \text{roll}_c \) for generating masks that are added to the input blocks in the compression layer
- \( \text{roll}_e \) to update the internal state during expansion
- \( \text{roll}_f \) for generating masks that are added to the permutation outputs in the expansion

We write \( \text{roll}(k) \) for the result after applying the rolling function once and \( \text{roll}^i(k) \) for the result after applying it \( i \) times.
Typically, rolling functions are lightweight linear functions with huge order, like updating functions of linear feedback shift registers. The main security requirement for the rolling function is the following. Informally, an adversary not knowing \(k\) shall not be able to predict the mask value \(\text{roll}^i(k)\) for any \(i\) in a reasonable range nor the difference between any pair of mask values \(\text{roll}^i(k) + \text{roll}^j(k)\) for any \(i \neq j\) in that range. The combination with permutations of low algebraic degree introduces an additional security requirement: the set \(\{\text{roll}^i(k)\}_{0 \leq i < n}\) shall not contain high-dimensional affine spaces for \(n\) a reasonable value (see Section A.1). Finally, to preserve the parallelism, as explained later, we wish the computation of roll to be lightweight.

3 Specification of Farfalle

We define Farfalle, a PRF construction that takes as input a variable-length secret key \(K\) and a data string sequence, and returns an extract of the output stream at a desired offset and for a desired length. It makes use of four permutations and three rolling functions. Informally, Farfalle consists of three parts. First, the key derivation computes a \(b\)-bit mask \(k\) from the key \(K\). Then, the compression layer computes a \(b\)-bit accumulator from the data string sequence using the mask \(k\). And finally, the expansion layer computes a \(b\)-bit rolling state from the accumulator and then generates the output from the rolling state and a mask derived from \(k\). We provide the definition in Algorithm 1 and an illustration in Figure 1.

The compression layer applies a permutation \(p_c\) to \(b\)-bit blocks, each the sum of a data block and a rolling mask \(\text{roll}_c^i(k)\) with \(i\) the index of the block in the sequence and (bitwise) adds them into the accumulator \(x\). It enjoys a rather powerful incremental property. The block index \(i\) only depends on the length and number of input blocks accumulated already. Clearly, if multiple Farfalle computations share the same data block \(x\) with the same block index, their contribution to the accumulator is the same and \(p_c(x + \text{roll}_c^i(k))\) needs to be computed only once. A special case of this is Farfalle applied to multiple string sequences with a common prefix and the same initial mask \(k\).

Note that to separate input strings, the value of the index for the first block of \(M^{(i)}\) is 2 higher than that for the last block of \(M^{(i-1)}\). There is hence a block index value for which there is no contribution to the accumulator. We call this a blank index, a concept elucidated in Figure 2.

While the application of the rolling function in the compression layer of Farfalle is essentially serial, the application of \(p_c\) for index \(i\) can be done as soon as the input block has been formed and \(\text{roll}_c^i(k)\) is available. Hence if \(\text{roll}_c\) is relatively lightweight, the main part of the computation, namely the evaluations of \(p_c\), can be done in parallel.

The compression and expansion layers of Farfalle require a secret mask \(k\) of exactly \(b\) bits. We wish to accommodate variable-length keys and for that purpose we generate the initial mask \(k\) by applying \(p_b\) to the padded user key \(K\).

The expansion layer computes the rolling state \(y\) from the accumulator \(x\) simply by applying a permutation \(p_d\) and uses the rolling state in a stream generator. For an output block with index \(j\), it applies a permutation \(p_e\) to the rolling state \(\text{roll}_d^j(y)\) and (bitwise) adds the rolling mask \(\text{roll}_f^j(k')\) to the result before presenting it at the output.
Algorithm 1 Definition of Farfalle \( [p_b, p_c, p_d, p_e, \text{roll}_c, \text{roll}_e, \text{roll}_f] \)

**Parameters:** b-bit permutations \( p_b, p_c, p_d \) and \( p_e \) and rolling functions \( \text{roll}_c, \text{roll}_e \) and \( \text{roll}_f \).

**Input:**
- key \( K \in \mathbb{Z}_2^b \), \( |K| \leq b - 1 \)
- input string sequence \( M^{(m-1)} \circ \cdots \circ M^{(0)} \in (\mathbb{Z}_2^b)^+ \)
- requested length \( n \in \mathbb{N} \) and offset \( q \in \mathbb{N} \)

**Output:** string \( Z \in \mathbb{Z}_2^n \)

\[ K' = \text{pad}10^*(K) \]

\[ k \leftarrow p_b(K') \text{ [mask derivation]} \]

\[ x \leftarrow 0^b \]

\[ I \leftarrow 0 \]

**for** \( j \) running from 0 to \( m - 1 \) **do**

\[ M = \text{pad}10^*(M^{(j)}) \]

Split \( M \) in b-bit blocks \( m_I \) to \( m_{I+\mu-1} \)

\[ x \leftarrow x + \sum_{i=I}^{I+\mu-1} p_c(m_i + \text{roll}_c^i(k)) \]

\[ I \leftarrow I + \mu + 1 \text{ [skip the blank index]} \]

\[ k' \leftarrow \text{roll}_c^I(k) \]

**y \leftarrow p_d(x) \]

**while** all the requested \( n \) bits are not yet produced **do**

produce b-bit blocks as \( z_j = p_c(\text{roll}_c^j(y)) + \text{roll}_f^j(k') \)

\[ Z \leftarrow n \text{ successive bits from concatenation of } z_0 || z_1 || z_2 \ldots \text{ starting from bit with index } q. \]

**return** \( Z = 0^n + F_K \left( M^{(m-1)} \circ \cdots \circ M^{(0)} \right) \ll q \)
Fig. 2: A closer look at the compression of example string sequence $B \odot A$ in Farfalle.

4 Modes of use

Farfalle can be readily used for MAC computation, key derivation and keystream generation. It allows amortizing the computation of $k$ among different computations with the same key $K$.

We now define three (authenticated) encryption modes on top of Farfalle. These modes are parameterized by tag length $t$ and/or an alignment unit length $\ell$. We assume that for any given key the parameter values are fixed. If there are instances of any of these modes with different parameters or different modes altogether, we assume their keys have been chosen independently. In other words, we claim no security for different mode instances with the same key or with different modes with the same key.

4.1 Session-supporting authenticated encryption scheme

In many use cases where one wishes confidentiality, authentication is required too and it makes sense to offer a scheme that provides both: an authenticated encryption scheme. Doing this with a PRF is simple: one enciphers the plaintext by adding to it the output of a PRF applied to a nonce and computes a tag on the ciphertext (and possibly metadata). Often, one does not only want to protect a single message, but rather a session where multiple messages are exchanged, such as in the Transport Layer Security (TLS) protocol [26] or the Secure Shell (SSH) protocol [50]. Examples of session-supporting authenticated encryption schemes are the CAESAR submissions Keyak [13] and KETJE [12]. They require only a nonce at the startup of the session and each tag authenticates all messages already sent in the session.
We define a session-supporting authenticated encryption scheme similar to the Motorist mode defined in [13]. The session keeps track of a history that is presented to an (incrementable) PRF for generating tags and keystream. Starting a session initializes the history to a nonce $N$. From then on, it supports messages consisting of metadata $A$ and/or plaintext $P$. The wrapping of a message consists of three phases. First, Farfalle-SAE adds to the plaintext, if non-empty, the output of the incremental PRF applied to the history in order to generate the ciphertext. Second, it appends the metadata and/or ciphertext to the history. Finally, it generates the tag by applying the PRF to the history. Note that a tag authenticates the complete history of the session. Unwrapping is similar.

Farfalle-SAE has two parameters: the tag length $t$ and an alignment unit length $\ell$. It reserves the first $t$ bits of the PRF output for tags and takes keystream from the PRF output stream from an offset that is the smallest multiple of $\ell$ not shorter than $t$. Moreover, Farfalle-SAE applies domain separation between metadata and ciphertext strings in the history to skip the first phase for plaintext-only messages or the second phase for metadata-only or even empty messages. We provide a formal specification in Algorithm 2.

**Algorithm 2** Definition of Farfalle-SAE[$F, t, \ell$]

<table>
<thead>
<tr>
<th>Parameters:</th>
<th>PRF $F$, tag length $t \in \mathbb{N}$ and alignment unit length $\ell \in \mathbb{N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong></td>
<td>taking key $K \in \mathbb{Z}_2^t$ and nonce $N \in \mathbb{Z}_2^n$, and returning tag $T \in \mathbb{Z}_2^t$&lt;br&gt;offset $= \ell \left\lceil \frac{t}{\ell} \right\rceil$; the smallest multiple of $\ell$ not smaller than $t$&lt;br&gt;history $\leftarrow N$&lt;br&gt;$T \leftarrow 0^t + F_K$ (history)&lt;br&gt;<strong>return</strong> $T$</td>
</tr>
<tr>
<td><strong>Wrap</strong></td>
<td>taking metadata $A \in \mathbb{Z}_2^t$ and plaintext $P \in \mathbb{Z}_2^n$, and returning ciphertext $C \in \mathbb{Z}_2^{\lVert P \rVert}$ and tag $T \in \mathbb{Z}_2^t$&lt;br&gt;$C \leftarrow P + F_K$ (history) $\ll$ offset&lt;br&gt;if $</td>
</tr>
<tr>
<td><strong>Unwrap</strong></td>
<td>taking metadata $A \in \mathbb{Z}_2^t$ ciphertext $C \in \mathbb{Z}_2^t$ and tag $T \in \mathbb{Z}_2^t$, and returning plaintext $P \in \mathbb{Z}_2^{\lVert C \rVert}$ or an error&lt;br&gt;$P \leftarrow C + F_K$ (history) $\ll$ offset&lt;br&gt;if $</td>
</tr>
</tbody>
</table>

Clearly, this mode requires the PRF to have the incremental property. When instantiated with Farfalle, the mask derivation must be done only once and in every call to Farfalle only the recently appended string must be compressed. Like in Motorist [13], the initialization returns a tag that can be sent along with the nonce (sender) or verified at the beginning of a session (receiver).
4.2 SIV authenticated encryption scheme

Farfalle-SAE, as well as Motorist, require the management of nonces: each session shall be started with a unique combination key and nonce. Nonce management is perceived as a difficult task by some, and for that audience we define a so-called synthetic initial value (SIV) authenticated encryption scheme [49, 66]. SIV authenticated encryption schemes have the feature that one can securely encipher different plaintexts with the same key without requiring the overhead of nonce management. It simply uses the tag computed over the message as a nonce for the encryption function and security only breaks down when two messages have the same tag. SIV modes were originally proposed for key transport, but system architects that are not sure about their ability to manage nonces now also propose it for other use cases. An example is the mode GCM-SIV [31] that was proposed to the Internet Research Task Force (IRTF) and the Crypto Forum Research Group (CFRG) as specified in a RFC [30].

Farfalle-SIV takes as input a secret key $K$, an arbitrary-length plaintext $P$ and arbitrary-length metadata $A$ and returns a ciphertext $C$ with the same length as the plaintext and a fixed-length tag $T$. It first computes the tag by applying a PRF to $P \circ A$ and enciphers $P$ by adding to it the output of the PRF applied to $T \circ A$. We provide a formal specification in Algorithm 3.

**Algorithm 3 Definition of Farfalle-SIV$[F, t]$**

Parameters: a PRF $F$ and tag length $t \in \mathbb{N}$

Wrap taking metadata $A \in \mathbb{Z}_2^2$ and plaintext $P \in \mathbb{Z}_2^2$, and returning ciphertext $C \in \mathbb{Z}_2^{\lfloor P \rfloor}$ and tag $T \in \mathbb{Z}_2^t$

$T \leftarrow 0^t + F_K (P \circ A)$

$C \leftarrow P + F_K (T \circ A)$

return $C, T$

Unwrap taking metadata $A \in \mathbb{Z}_2^2$, ciphertext $C \in \mathbb{Z}_2^t$ and tag $T \in \mathbb{Z}_2^t$, and returning plaintext $P \in \mathbb{Z}_2^{\lfloor C \rfloor}$ or an error

$P \leftarrow C + F_K (T \circ A)$

$T' \leftarrow 0^t + F_K (P \circ A)$

if $T' = T$ then

return $P$

else

return error!

The security of this mode relies on Farfalle to be a PRF. For confidentiality, it requires that all messages that are encrypted with the same key and that have the same metadata $A$ result in a different tag $T$. Two such messages with colliding tags will use the same keystream to encipher their plaintexts and hence the sum of their ciphertexts will be equal to the sum of their plaintexts. For a chosen value of the parameter $t$, the probability of a tag collision occurring in $n$ messages is upper bounded by $n^2 / 2^{t+1}$. For example, if in an application it is reasonable to assume that only $2^{40}$ messages with the same (or empty) metadata and the same key will be processed, and one is willing to accept a risk of a collision up to $2^{-40}$, tags of length 128 bits will do the job. For tag collisions between messages with different metadata $A$ there is no security problem as the keystream also depends on the metadata $A$.

When instantiated with Farfalle, the mask derivation and the compression of $A$ that are in common to both calls to $F_K ()$ must be done only once, thanks to the incremental property.
4.3 Wide block cipher

There are use cases where it would be practical to have a block cipher with a custom block size, or where the block size is adaptable to the task at hand and that supports next to the key an additional diversification parameter, called a tweak. Examples include disk encryption, where the block size would equal the size of sectors. Another example is encryption in the Tor anonymity network [43]. Here every block of data must be encrypted recursively multiple times in such a way that the cryptogram is not longer than the plaintext and this joint encryption must achieve a certain authentication. This can be achieved with a tweakable wide block cipher with a width of 509 bytes.

We define a tweakable wide block cipher based on two PRFs. The global construction is an instantiation of the HHFHFH mode as presented by Dan Bernstein at the Symmetric Cryptography Dagstuhl seminar in January 2016 [5], that is in turn based on work of Naor and Reingold [44], that is based on a paper by Stefan Lucks [40], that builds further on work of Luby and Rackoff [39].

It takes as input a secret key $K$, an arbitrary-length plaintext $P$ and an arbitrary-length tweak $W$ and returns a ciphertext $C$ of same length as the plaintext. It performs a 4-round Feistel network to the plaintext. The latter is split in a left and a right part, with length determined by the function split[$b$, $\ell$]. This function takes the block length $n$ and returns the length of the left part $n_L$, which is a multiple of the given alignment unit length $\ell$. The function $G$ (corresponding to $F \circ H$ in HHFHFH) used in the two middle rounds must be a PRF and takes as input part of the intermediate result and the tweak. The function $H$ used in the first and last round does not necessarily have to be a PRF but must be differentially uniform. We provide a formal specification of the split function in Algorithm 4 and of the Feistel network in Algorithm 5. A rationale for the split function is given at the end of this section.

A (tweakable) wide block cipher can be converted to an authenticated encryption scheme by applying a very simple mode [32]. The metadata is used as tweak and as encipherment input one uses the plaintext with some agreed verifiable redundancy, such as 8 bytes equal to zero appended to the end. The cryptogram is the encipherment output. One can authenticate the cryptogram by verifying that the decipherment output ends in the agreed fixed string. This verification process can be performed before full decipherment is completed, allowing for early rejection of unauthentic cryptograms. As observed in [29], in comparison to SIV, a wide block cipher mode has the advantage of smaller overhead for the same forgery resistance. We provide a formal specification in Algorithm 6.

The wide block cipher in principle supports any length. However, the generic security it can achieve is limited by the ability to generate collisions in the left or right part of the intermediate result. Such collisions become likely as soon as the number of processed blocks reaches $2^{\min(n_L,n_R)/2} \leq 2^n/4$. For that reason, one cannot claim a security level higher than the width divided by 4 and hence the width cannot be taken too small.

When instantiating $H$ and $G$ with two instances of Farfalle that have the same permutation for $p_b$, the mask derivation can be pre-computed. Similarly, the compression of the tweak $W$ can be shared among the computations of $G_K (R||1 \circ W)$ and $G_K (L||0 \circ W)$.

**Rationale for the split function** The function split[$b$, $\ell$] returns a value $n_L$ that minimizes $m_L + m_R$, where $m_L$ is the number of $b$-bit blocks to hold the $L$ string in Algorithm 5 and $m_R$ is defined similarly for $R$. We further require that $L$ is aligned on $\ell$ bits, i.e., that
Algorithm 4 Definition of split[b, ℓ]

Parameters: permutation width b ∈ ℤ and alignment unit length unitLength ∈ ℤ, satisfying ℓ ≥ 2 and ℓ|b

Input: block length n ∈ ℤ

if n ≤ 2b − (ℓ + 2) then

\[ n_L = \ell \left\lfloor \frac{n + \ell}{2\ell} \right\rfloor \]

else

\[ q = \left\lfloor \frac{n + \ell + 2}{b} \right\rfloor, \text{ i.e., } q \text{ is smallest multiple of } b \text{ that } n + \ell + 2 \text{ fits in} \]

\[ x = \left\lfloor \log_2(q - 1) \right\rfloor, \text{ i.e., } x \text{ is largest integer such that } 2^x < q \]

\[ n_L = (q - 2^x)b - \ell \]

return \( n_L = \text{split}(n) \)

Algorithm 5 Definition of encryption in Farfalle-WBC[H, G, ℓ]

Parameters: two PRFs H and G and alignment unit length ℓ \∈ \mathbb{N}

Encryption taking key \( K \in \mathbb{Z}_2^\ell \), tweak \( W \in \mathbb{Z}_2^\ell \) and plaintext \( P \in \mathbb{Z}_2^\ell \) and returning ciphertext \( C \in \mathbb{Z}_2^{|P|} \)

\( L \) gets first split(|P|) bits of \( P \) and \( R \) the remaining ones

\( R_0 \leftarrow R_0 + H_K(|1|0) \), with \( R_0 \) the first \( \min(b, |R|) \) bits of \( R \)

\( L \leftarrow L + G_K(|1|0 \circ W) \)

\( R \leftarrow R + G_K(|0|0 \circ W) \)

\( L_0 \leftarrow L_0 + H_K(|1|1), \) with \( L_0 \) the first \( \min(b, |L|) \) bits of \( L \)

return \( C \leftarrow \text{Encryption}(K, A, P') \)

Algorithm 6 Definition of Farfalle-WBC-AE[H, G, t, ℓ]

Parameters: two PRFs H and G, the expansion length \( t \in \mathbb{N} \) and alignment unit length \( \ell \in \mathbb{N} \)

Wrap taking key \( K \in \mathbb{Z}_2^\ell \), metadata \( A \in \mathbb{Z}_2^\ell \) and plaintext \( P \in \mathbb{Z}_2^\ell \) and returning ciphertext \( C \in \mathbb{Z}_2^{|P|+t} \)

\( P' \leftarrow P||0^t \)

return \( C \leftarrow \text{Encryption}(K, A, P') \)

Unwrap taking key \( K \in \mathbb{Z}_2^\ell \), metadata \( A \in \mathbb{Z}_2^\ell \) and ciphertext \( C \in \mathbb{Z}_2^t \), and returning plaintext \( P \in \mathbb{Z}_2^{|C|-t} \)

or an error

\( L \leftarrow L_0 + H_K(|1|1 \circ A) \), with \( L_0 \) the first \( \min(b, |L|) \) bits of \( L \)

if \( |R| > b + t \) then

if the last \( t \) bits of \( R \neq 0^t \) then return error!

\( L \leftarrow L + G_K(|1|0 \circ A) \)

\( R_0 \leftarrow R_0 + H_K(|0|0), \) with \( R_0 \) the first \( b \) bits of \( R \)

else

\( L \leftarrow L + G_K(|0|0 \circ A) \)

\( R_0 \leftarrow R_0 + H_K(|0|0), \) with \( R_0 \) the first \( \min(b, |R|) \) bits of \( R \)

if the last \( t \) bits of \( L||R \neq 0^t \) then return error!

\( P' \leftarrow L||R \)

return \( P \leftarrow \) the first \(|C| - t \) bits of \( P' \)
$n_L$ is a multiple of $\ell$. More precisely, the number of blocks at the left is $m_L = \left\lceil \frac{n_L + 2}{b} \right\rceil$, since Algorithm 5 requires appending one frame bit for domain separation and Farfalle adds one bit of padding. As $\ell \geq 2$ and we want that $n_L | \ell$, these two bits are absorbed in a buffer of at least $\ell$ bits that ensures the alignment, and we can equivalently write $m_L = \left\lceil \frac{n_L + 2}{b} \right\rceil$. Similarly, the number of blocks at the right is $m_R = \left\lceil \frac{n_R + 2}{b} \right\rceil$. Because of the domain separation bits, padding bits and buffer for alignment, the total number of blocks $m_L + m_R$ is at least $q$ as defined in Algorithm 4. An optimal solution means $m_L + m_R = q$.

This function split $[b, \ell]$ addresses an additional requirement, namely that $\min(n_L, n_R)$ is not too small, to take into account the difference uniformity of $HK$ with output truncated to $n_L$ or $n_R$ bits. The function split distinguishes between two cases.

- For $n \leq 2b - (\ell + 2)$, we have $m_L = m_R = 1$. The function split maximizes $\min(n_L, n_R)$ within the constraint that $n_L$ is a multiple of $\ell$. More precisely, it is easy to show that $|n_L - n_R| \leq \ell$.

- For $n > 2b - (\ell + 2)$, we first translate the requirement on the lengths to $\min(n_L, n_R) \geq b - \ell$. Algorithm 5 realizes this by taking $n_L = m_L b - \ell \geq b - \ell$ for some number of blocks $m_L \geq 1$ on the left side. For the right side, we have $q \geq 3$, so that $m_R \geq 2$ and $n_R \geq b - 1$. Second, $n_L$ is a multiple of $\ell$ bits since $b$ is a multiple of $\ell$. Then, we reach optimality since $n_R = n - (m_L b - \ell)$ and thus $m_R = \left\lceil \frac{n_R + 2}{b} \right\rceil = q - m_L$. (Note that this works independently of the choice of $m_L$.) Finally, the function splits these $q$ blocks into the largest power of two for $R$. As the parallelism degree is often a power of two, this enables its optimal exploitation.

5 Rationale for Farfalle

A Farfalle function loaded with a secret key $K$ can be distinguished from a random function in several ways. We list here the types of attack that have played an important role in shaping Farfalle and that impose criteria on the used permutations and rolling functions:

1. accumulator collision: finding two input strings leading to identical accumulators;
2. weaknesses in the mask derivation;
3. distinguishing the output for a single input from a random string
   - retrieving rolling state and mask using algebraic attacks
   - detecting bias in the output sequence using higher-order differentials
   - detecting bias in the output sequence using correlation propagation
4. distinguishing the output for multiple inputs from random strings;
5. finding the values of $y$ and $k$ from input-output pairs.

We discuss these attacks in the following subsections.

5.1 Accumulator collision

Recall that the accumulator is the value $x$ that sums the contributions of $p_c(m_i + \text{roll}_i^j(k))$. Finding two string sequences $M$ and $M'$ leading to the same value of the accumulator can be used to attack Farfalle. The difficulty of finding collisions in the accumulator is
based on the differential propagation properties of $p_c$ and is helped by the fact that before being subject to $p_c$, each input blocks $m_i$ is whitened with the rolling input mask $\text{roll}_i^{\delta}(k)$ and that the accumulator cannot be directly observed. One may attempt to generate such collisions in several ways. We describe them in this subsection.

**Sets of input blocks that contribute 0 to the accumulator** The simplest method, at least conceptually, is to form $M'$ by appending a block at the end of $M$ and hope its $b$-bit contribution to the accumulator is zero. Succeeding in doing this is equivalent to successfully guessing $k$.

One may try to append two blocks where the input to the permutation is equal: $m_i + \text{roll}_i^{\delta}(k) = m_{i+1} + \text{roll}_{i+1}^{\delta}(k)$). The ability of the adversary to do this critically relies on her ability to predict $\text{roll}_i^{\delta}(k) + \text{roll}_{i+1}^{\delta}(k)$, imposing the requirement for the rolling function that it shall be hard to predict $k + \text{roll}(k)$ for unknown $k$. This attack can be generalized by not only appending blocks to $M$, but also putting input blocks in $M'$ where there are blank indices in $M$. Then it shall be hard to predict $k + \text{roll}_i^{\delta}(k)$ for unknown $k$ and any offset $\delta$ in a reasonable range.

Appending multiple blocks and saying something meaningful on their joint contribution to the accumulator is possible if the permutation has sufficiently low algebraic degree and the rolling function allows predicting mask differences. Basically, if one chooses the input blocks such that the corresponding inputs to $p_c$ form an affine space of sufficient dimension, their joint contribution is zero. We refer to Section [A.1] for more explanation. This imposes a requirement on the rolling function that a sequence of masks with reasonable index values shall not contain an affine space of dimension higher than the algebraic degree of the permutation.

**Input block variants swapping $p_c$ inputs** One may try to construct $M'$ from $M$ by modifying two input blocks $m_i$ and $m_j$ into $m'_i$ and $m'_j$, such that the input to $p_c$ for $m_i$ equals that of $m'_j$ and vice versa. This will just swap the contribution of blocks $i$ and $j$ to the accumulator. As the addition is associative, the joint contribution is the same in both cases. This boils down to finding $m_i + \text{roll}_i^{\delta}(k) = m'_j + \text{roll}_j^{\delta}(k)$ and a similar expression for $m_j$ and $m'_j$. The ability of the adversary to come up with such input blocks again critically depends on her ability to predict $k + \text{roll}_i^{\delta}(k)$ for unknown $k$ and any offset $\delta$ in a reasonable range.

**High-probability differentials in $p_c$** Another approach exploits high-probability differentials in $p_c$. One applies two different inputs $M$ and $M'$ that have the same length but differ in a limited number of blocks, denoted as active. Due to the invertibility of $p_c$, the smallest number of active blocks that may lead to a collision is 2. Let the difference in the first block be $\Delta$ and that in the second block $\Delta'$. We have a collision if these differences propagate to the same difference through the permutation. Assuming the adversary does not know the mask values for the active blocks, the probability of a collision is

$$\Pr(\text{collision}) = \sum_{\gamma} \text{DP}(\Delta, \gamma) \text{DP}(\Delta', \gamma),$$

with $\text{DP}(x,y)$ the differential probability of differential $(x,y)$ over $p_c$. The differential probabilities $\text{DP}(\Delta, \gamma)$ for fixed $\Delta$ and varying $\gamma$ can be seen as a vector with $2^b$ components, all positive and summing to 1. Equation (1) can be seen as an inner product of such
vectors. From this it follows that the highest values can be obtained by taking $\Delta = \Delta'$, yielding:

$$\Pr(\text{collision}) = \sum_\gamma \text{DP}^2(\Delta, \gamma).$$

(2)

This presents a clear criterion for $p_c$.

In the initial rounds an adversary may try to guide the difference propagation by making assumptions on the masks $\text{roll}^i(k)$ and $\text{roll}^j(k)$ and choose the corresponding message blocks $m_i$ and $m_j$ to satisfy certain conditions in the propagation. If the number of bits to be guessed is smaller than the number of conditions to be satisfied, this may result in some gain. This hence presents a criterion for the rolling function.

5.2 Properties of the mask derivation

To derive the mask $k$, we apply the $p_b$ permutation to a variable-size key $K$ after padding it in a reversible way. This ensures that no collision can happen between different keys, even with different sizes.

We would like to highlight two properties that we expect from the mask derivation.

– As explained in Section 5.1, the adversary should have no effective way to predict the value of $k + \text{roll}_c^\delta(k)$ for reasonable values of $\delta$. Otherwise, she can easily produce accumulator collisions, e.g., by swapping input blocks.

– Regarding differential attacks, the input to $p_c$, namely, $m_i + \text{roll}_c^i(k)$, should look sufficiently random to an adversary, so as to prevent her from choosing input values that significantly decreases the workload to propagate a difference according to a differential trail.

To satisfy these properties, we express the requirement on $p_b$ that for every matrix $M$ of rank $r$, the min-entropy of $M \times k$ is close to the minimum of $r$ and the min-entropy of the key $K$. This encompasses the first property when $\text{roll}_c$ is a linear function, as in the case of $\text{kra} \text{va} \text{t} \text{te}$, by choosing $M = I + \text{roll}_c^\delta$, with $I$ the identity matrix. For the second property, when $p_c$ employs a degree-two round function, the conditions on the absolute values for the propagation of differences are linear. Hence, by choosing $M$ accordingly, this requirement ensures that the difference cannot be propagated with a higher probability than expected.

We may need to restrict on the allowed distribution of keys $K$ to exclude (artificial) key distributions that would pose a problem for particular rolling functions. As in the case of $\text{kra} \text{va} \text{t} \text{te}$, a rolling function can operate on $e$ bits of the mask, its support, and leave the remaining $b - e$ bits invariant. One could craft a distribution of $K$ such that $k$ always has a particular key-independent value in the support of $\text{roll}_c$. This would allow switching blocks as the difference between the masks of two different blocks, $k + \text{roll}_c^\delta(k)$, is known for any $\delta$. Despite the fact that this distribution would be fairly artificial and involve the definition of $p_b^{-1}$, a solution could be to require that the keys $K$ are drawn uniformly.

5.3 Distinguishing the output from a random string (single input)

In this section, we assume that $\text{roll}_i$ is the identity. Attacks discussed in this section tend to become more complex if $\text{roll}_i$ is a true rolling function.
Clearly, one may have a collision in the sequence of \(\text{roll}_e^i(y)\) values if there is a short cycle or if different accumulator values \(x\) and \(x'\) lead to rolling state values \(y = p_d(x)\) and \(y' = p_d(x')\) such that \(\text{roll}_e^i(y') = y\) for some reasonably small value of \(\delta\). As a consequence, the rolling function \(\text{roll}_e^i\) must be chosen such that the states are in very long cycles.

Every output block depends on an output mask and a rolling state. Hence, it requires at least two output blocks to determine the value of the rolling state and/or the output mask. When performing an algebraic attack using two output blocks, the adversary must solve a system of equations with unknown variables spread over two full instances of the permutation \(p_e\). Using more than two output blocks leads to an overdetermined system of equations, with even more variables. Due to the application of the rolling function to the rolling state, every additional output block will introduce many additional variables. The main criterion for \(p_e\) due to these attacks is the difficulty of solving a system of non-linear equations involving (at least) two instances of \(p_e\).

One may try to distinguish the output by observing biases in its bits, or combination of bits. If \(p_e\) has sufficiently low algebraic degree, one may try to find affine spaces of dimension at least equal to that degree in the sequence of rolling states. If such an affine space can be found, the corresponding output blocks sum to zero allowing to distinguish it from a random sequence. This presents a criterion for \(\text{roll}_e\).

Another method to distinguish the output is to find biases in linear combinations (parities) of output bits using the correlation properties of the permutation. Consider the case where these bits are spread across two blocks. The output block with index \(i + j\) is completely determined by the output block with index \(j\) and the output mask: \(p_e^{-1}(z_{i+j} + k) = \text{roll}_e^i(p_e^{-1}(z_j + k))\), or equivalently

\[
z_{i+j} = k + \left(p_e \circ \text{roll}_e^i \circ p_e^{-1}\right)(z_j + k).
\]

The sign of the bias in any parity of output bits \(v^T z_j + u^T z_{i+j}\) depends on \(k\), but its amplitude is independent of \(k\) and equal to the absolute value of

\[
\sum_w \text{Corr}_{p_e}(u, \text{roll}_e^i(w)) \text{Corr}_{p_e}(v, w).
\]

This comes down to a joint criterion for \(p_e\) and \(\text{roll}_e\).

### 5.4 Distinguishing the output from a random string (multiple inputs)

We now describe a distinguishing attack that requires \(2^d\) chosen input strings with \(d - 1\) the algebraic degree of \(p_e \circ p_d\). For a given key \(K\), we compute Farfalle for \(2^d\) input strings, each consisting of \(d\) input blocks and one block of padding. Each string has the form \(m(\lambda) = m_0^\lambda || m_1^\lambda || \ldots || m_{d-1}^\lambda\), where \(\lambda \in \mathbb{Z}_2^d\) and \(m_0^\lambda \neq m_1^\lambda\) for all \(i\).

If we denote \(r_i = p_e(m_0^i + \text{roll}^i(k))\) and \(r_i' = p_e(m_1^i + \text{roll}^i(k))\) and \(r_i' = r_i + r_i' \neq 0\), then the value of the accumulator for the input string with label \(\lambda\) is \(x = \sum_i r_i + \sum_i \lambda_i r_i'\).

Over the space of input strings, this is an affine space. So summing the Farfalle outputs for these input strings corresponds to taking the \(d\)-th derivative of the function applied to the accumulator. If this function has degree less than \(d\), this sum becomes zero. Hence, we can use this to distinguish the Farfalle output from a random string. Clearly, this imposes a criterion on the algebraic degree of \(p_e \circ p_d\). Note that this works independently from the algebraic degree of \(p_e\) and of the concrete values of the blocks \(m_0^i\) and \(m_1^i\).
5.5 Finding the value of $k$ from input-output pairs

Finally, finding the value of $k$ from input-output pairs can be seen as a variant to doing key retrieval in an Even-Mansour [27] cipher, where the permutation is $p_e \circ p_d \circ p_c$. Here the complete spectrum of classical block cipher attacks can be applied. The main differences are that in Farfalle an adversary has the additional degree of freedom of exploiting $\text{roll}_i(y)$ to alter the rolling state that forms the intermediate state after $p_d \circ p_c$ rounds, and that in most Even-Mansour use cases an adversary can query the inverse cipher, which is not the case for Farfalle.

A key point for Farfalle is that the marginal cost for processing an input block is executing $p_c$, for generating an output block is executing $p_e$ but that an adversary performing an attack using input-output pairs must span $p_e \circ p_d \circ p_c$.

6 Comparison with prior art

We describe the prior art for Farfalle in Section 6.1, for the session-supporting authenticated encryption mode in Section 6.2, for the SIV authenticated encryption mode in Section 6.3 and finally for the wide block cipher in Section 6.4.

6.1 Farfalle

The Farfalle construction reminds of the keyed sponge construction, with most efficient version the full-width keyed sponge [7,13,21]. It differs in that the keyed sponge is strictly serial while Farfalle consists of two main layers that are by themselves parallel. The keyed sponge can be duplexed, i.e., incremental inputs can be processed, with consequence that the partial input and output to an underlying permutation $f$ are available to the adversary. Duplexing works in Farfalle too but in a slightly different manner and never leads to the input and output of a single call to $p_c$, $p_d$ or $p_e$ being available to the adversary. This implies that for equal safety margins in Farfalle one can afford to take permutations with less rounds than in the keyed sponge. Moreover, in the sponge construction the (squeezing) rate is limited to $b - c$ with $c$ the capacity. As the capacity determines the security strength, the sponge construction tends to become less efficient for small permutation widths. In Farfalle, one can plug in much smaller permutations for the same target security strength. The size is basically limited by the birthday bound on having collisions in the input to $p_c$ or $p_e$.

Farfalle can readily be used for MAC computation, keystream generation, key derivation and as building blocks in more elaborate schemes. Its computational cost is a single permutation $p_h$ for setting up the key, the single call to $p_d$ and then one execution of $p_c$ per input block and one execution of $p_e$ per output block. We compare to some similar MAC modes (setting aside the fact that these modes do not support arbitrary-length outputs):

Protected counter sum This is a method proposed in [3] to build an unpredictable function with arbitrary-length input from an unpredictable function with fixed-length input. The Farfalle mode, with its output restricted to a single block, can be seen as an instantiation of this method. In Farfalle, the unpredictable function with fixed-length input is implemented with a permutation, where the rolling function (implicitly) codes the block index. The permutations $p_d$ and $p_e$ and the whitening of the output with a mask take the place of the final call of the unpredictable function in the protected
counter sum. Farfalle has several key ideas in common with protected counter sums, but there are also some differences. The most important differences are that Farfalle has an arbitrarily extendable output and that Farfalle has the ambition to be unpredictable (in the terminology of [3]) even if the underlying building blocks are not unpredictable as such.

**PMAC** This is a block cipher mode for MAC computation proposed in [17] and a variant of protected counter sum. The random function is instantiated by a block cipher and it has a specific coding for the counter block in the inputs. The compression layer of Farfalle is similar to PMAC as it performs the block cipher calls in parallel and (bitwise) adds their outputs. In PMAC the input blocks are $b$ bits and before applying the block cipher, their value is offset by a rolled version (with rolling function based on a Gray code) of a $b$-bit secret $k$ derived from the user key $K$ with a block cipher call. The tag is obtained by offsetting the accumulator with a rolled version of $k$ and applying the block cipher to the result.

**Alred** This was one of the first permutation-based modes proposed for MAC computation in [23,25] and is mostly known for the instance Pelican-MAC based on AES [24]. The main difference with Farfalle is that it is strictly serial and can therefore not take advantage of available parallelism (e.g., pipelining in the AES-NI instructions) in resources. On the other hand, it shares with Farfalle that never an input and output to a permutation $f$ is available and uses that to reduce the number of rounds in $f$ drastically compared to more generic modes such as CBC-MAC or C-MAC. Moreover, the output length of Alred is limited to the width of the underlying block cipher and it does not support multiple input strings.

**MARVIN** This is a mode for MAC computation proposed in [34,33] that was inspired at the same time by Alred and by PMAC and uses both a block cipher and a permutation. The compression layer of Farfalle is similar to that of MARVIN that applies a permutation to the input blocks in parallel and (bitwise) adds their outputs in an accumulator. The input blocks to the permutation calls are formed in a way similar to PMAC, but here a variant of Hugo Krawczyk’s cryptographic CRC [35] is used for rolling. In MARVIN, all input blocks pass through the permutation before being added into the accumulator. Moreover, the tag is obtained by applying the block cipher to the accumulator offset with the secret $k$ and some constants coding message and tag lengths.

We can also compare to some stream cipher modes (setting aside the fact that these modes do not support arbitrary-length inputs):

**Counter mode of a block cipher** In counter mode, a priori no distinction is made between the long-term nonce and the short-term block index. Depending on the size of the nonce, an adversary can apply differences of large choice and observe corresponding outputs.

**Mode underlying Salsa and ChaCha** These stream ciphers proposed in [4] can be considered as a mode on top of a permutation. This is very close to counter mode of a block cipher, where the block cipher is rather of type Even-Mansour. Again, the adversary has more degrees of freedom than in Farfalle.

Finally, we can compare to the pseudorandom function HS1 that is used in the CAESAR submission HS1-SIV [36]. This pseudorandom function uses two strongly different functions for compression and for expansion: a differentially uniform hash function for compression and ChaCha (in a non-standard mode) for expansion. The expansion part
is purely parallel but the compression part has only limited parallelizability. Farfalle has
the advantages that it can be constructed with a single permutation only, it is in general
much simpler and it takes sequences of input strings rather than a single one, simplifying
the modes built on top of it.

6.2 Session-based authenticated encryption mode

For Farfalle-SAE a comparison with duplex-based session-supporting authenticated encryp-
tion seems appropriate. The most recent mode realizing this is the Motorist mode
underlying KEVAK [13]. Functionally, Farfalle-SAE and Motorist are almost the same. The
difference is in the way this is realized. Motorist supports parallelism but it is limited and
must be determined or negotiated at session setup. Farfalle-SAE on the other hand is fully
parallel for each metadata \( A \) or plaintext \( P \).

When considering protection against differential power analysis (DPA) or differential
electromagnetic analysis (DEMA), Motorist offers a high level of leakage resilience. Its
key is only applied at session setup. From that point on, the state evolves and its value
depends on all input received. If nonce uniqueness is respected, state values for different
sessions will be completely decorrelated and a DPA/DEMA attack can only be applied to
the session setup phase. The consequence is that DPA countermeasures such as masking
must only be applied during that phase. Achieving something similar for Farfalle-SAE
requires adopting appropriate rolling functions. First, roll\(_C\) and roll\(_F\) must affect all bits
of the mask. A suitable choice would be a single round of a cryptographic permutation
operating on the full \( b\)-bit string. The rolling state can be protected in the same way by
adopting a similar rolling function for roll\(_E\). For dealing with multiple Farfalle calls with
the same key, it would be necessary to tweak Farfalle slightly and allow taking in as ini-
tial mask the final mask of the previous call to Farfalle. Notice that the introduction of
leakage resilience goes at the expense of some parallelism as the rolling function becomes
heavier.

6.3 SIV authenticated encryption mode

Our mode Farfalle-SIV is a close variant of the SIV construction [49]. The SIV construction
was proposed mostly for the purpose of key wrapping and that was later adopted in the
CAESAR submission HS1-SIV [36] for authenticated encryption. The main advantage of
Farfalle-SIV in comparison to these two examples is the following. In the original SIV
construction, the input consists of only metadata \( A \) and plaintext \( P \). These are subject to
a first keyed PRF that results in a tag (called IV) that serves as input to a second keyed
PRF for generating the keystream. In case the two inputs have colliding tags, the same
keystream is used to encipher two different plaintexts. In HS1-SIV this is addressed by
having an additional nonce that is input to both PRFs and having a tag collision is only
problematic if also the same nonce is used for both messages.

In contrast, Farfalle-SIV has no dedicated nonce, but the metadata \( A \) are input in both
PRFs. So now there is only a problem if there is a tag collision and if the two message
have the same metadata \( A \). Thanks to the incremental property of Farfalle, the only cost
this incurs is the caching of the contribution of the metadata to the accumulator.
6.4 Wide block cipher

The novelty of Farfalle-WBC with respect to HHFHFH [3] is that thanks to the incremental property of Farfalle, the compression of the tweak must be done only once.

Another tweakable wide block cipher construction is AEZ-Core proposed in the CAESAR submission AEZ [22]. It is not easy to compare Farfalle-WBC with AEZ-Core, as the former is based on permutations and the latter on tweakable block ciphers. Although it is hard to measure simplicity, we feel that Farfalle-WBC is a significantly simpler construction than AEZ-Core.

7 Kravatte: Farfalle based on Keccak-p

In this section, we present a Farfalle instance based on Keccak-p, the permutation underlying Keccak, Keyak and Ketje and standardized in FIPS 202 [10][12][13][47].

Definition 1. Kravatte is Farfalle[pb, pc, pd, pe, rollc, ro lle, rollf] with the following parameters:
- pb = pc = Keccak-p[1600, nr = 6],
- pd = pe = Keccak-p[1600, nr = 4],
- rollc as specified below,
- rollc = rollf,
- rollf is the identity.

The rolling function rollc applies a linear transformation to the five lanes of the plane y = 4 of the Keccak-p state and leaves the other 20 lanes unchanged:

\[
\begin{align*}
    a[x][4] &\leftarrow a[x + 1][4] & \forall x \neq 4 \\
    a[4][4][z] &\leftarrow a[0][4][z - 7 \text{ mod } 64] + a[1][4][z] & \forall z > 60 \\
    a[4][4][z] &\leftarrow a[0][4][z - 7 \text{ mod } 64] + a[1][4][z] + a[1][4][z + 3] & \forall z \leq 60
\end{align*}
\]

7.1 Security claim

We make the following security claim on Kravatte.

Claim 1 Let K = (K_0, \ldots, K_{u-1}) be an array of u secret keys, each uniformly and independently chosen from \( \mathbb{Z}_2^k \) with \( k \leq 320 \). Then, the advantage of distinguishing the array of functions Kravatte_{K_i}(\cdot) with \( i \in \mathbb{Z}_u \) from an array of random oracles RO(\cdot, \cdot), is at most

\[
\frac{uN + \binom{u}{2}}{2^k} + \frac{N}{2^{256}} + \frac{M}{2^{137}} + \frac{\sqrt{2^{uN'}}}{2^{x/2-1}} + \frac{N'}{2^{127}}. 
\]

Here,

- \( N \) is the computational complexity expressed in the (computationally equivalent) number of executions of Keccak-p[1600, nr = 4],
- \( N' \) is the quantum computational complexity expressed in the (equivalent) number of quantum oracle accesses to Keccak-p[1600, nr = 4], and
– $M$ is the online or data complexity expressed in the total number of input and output blocks processed by $\text{Kravatte}_K(\cdot)$.

In (3), the first term accounts for the effort to find one of the $u$ secret keys by exhaustive search, and for the probability that two keys are equal. The second term expresses that the complexity of recovering the accumulator or any rolling state inside $\text{Kravatte}$ must be as hard as recovering 256 secret bits. The third term expresses the effort to find a collision in the accumulator. The number 137 in the denominator follows the lower bound on differential trails inside Keccak-p as detailed in Section 7.4.

The fourth and fifth terms only apply if the adversary has access to a quantum computer. The fourth term accounts for a quantum search (or quantum amplification algorithm) to find a $k$-bit key among $u$ possibilities $[2^k, 19]$. The probability of success after $N'$ iterations is $\sin^2 \left( \left( 2N' + 1 \right) \theta \right)$ with $\theta = \arcsin \sqrt{u/2^k}$. We upper bound this as $2N' \sqrt{u/2^k}$. The fifth term similarly accounts for a quantum search over 256 secret bits.

Note that we assume that $\text{Kravatte}$ is implemented on a classical computer. In other words, we do not make claims w.r.t. adversaries who would make quantum superpositions of queries to the device implementing $\text{Kravatte}$ and holding its secret key(s).

We limit ourselves to fixed-length keys that are chosen uniformly and independently to keep our claim simple. We expect the construction to be secure for nonuniformly chosen and variable-length keys too, but it is possible to come up with pathological cases that would not offer good security and that hence should be excluded in the security claim (see Section 5.2).

In the multi-user setting, we require the keys to be independently drawn. If an adversary can manipulate $K$, such as in so-called unique keys that consist of a long-term key with a counter appended, we recommend hashing the key and the counter with a proper hash function such as KangarooTwelve [15].

7.2 $\text{Kravatte}-\text{SIV}$ and -SAE

Functions and schemes based on $\text{Kravatte}$ follow the same naming conventions as for Farfalle and adopt specific values for the parameters. In particular:

– $\text{Kravatte}-\text{SIV}$ has $t = 256$,

– $\text{Kravatte}-\text{SAE}$ has $t = 128$ and $\ell = 8$.

7.3 $\text{Kravatte}-\text{WBC}$ and -WBC-AE

The wide block cipher and the authenticated encryption scheme based on it make use of two Farfalle instances: one for $H$ and one for $G$. In $\text{Kravatte}-\text{WBC}$, the latter is instantiated with $\text{Kravatte}$ and the former is a variant of $\text{Kravatte}$ that we call $\text{Short-Kravatte}$. $\text{Short-Kravatte}$ has the same parameters as $\text{Kravatte}$, with the sole exception of $p_d$ that is the identity function instead of Keccak-p[1600, $n_r = 4$]. In addition, we set the following parameters:

– $\text{Kravatte}-\text{WBC}$ has $\ell = 8$,

– $\text{Kravatte}-\text{WBC-AE}$ has $t = 128$ and $\ell = 8$. 
Making joint use of Kravatte and Short-Kravatte is not something we support in general, as it is specific to the definition of Kravatte-WBC. We make a dedicated security claim on Kravatte-WBC.

Claim 2 Let \( K = (K_0, \ldots, K_{u-1}) \) be an array of \( u \) secret keys, each uniformly and independently chosen from \( \mathbb{Z}_2^k \) with \( k \leq 320 \) and let \( P_{K_i}(\cdot) \) with \( i \in \mathbb{Z}_u \) be instances of Kravatte-WBC. Each of these instances support two interfaces:

Encipherment denoted as \( C = P_{K_i}(W, P) \) taking as input a tweak and a plaintext and returning a ciphertext;

Decipherment denoted as \( P = P_{K_i}^{-1}(W, C) \) taking as input a tweak and a ciphertext and returning a plaintext.

We express as \( \text{Adv}^{\text{sprp}} \) the probability of distinguishing \( P_{K_i}(W, \cdot) \) from an array of uniformly and independently drawn random permutations \( \pi_{i, W, n} \) indexed by the key index \( i \), the value of \( W \) and the length \( n = |P| = |C| \), where the adversary can query the inverse permutations.

Let \( n_{\text{min}} \) be the minimum length \( n \) among all the queries. When \( n_{\text{min}} \geq 512 \) bits, \( \text{Adv}^{\text{sprp}} \) is claimed to be upper bounded by

\[
\text{(3)} + \frac{M^2}{2^{n_{\text{min}}/2 - 4}}. \tag{4}
\]

Here, \( N, N' \) and \( M \) are as in Claim 4 except that \( M \) also counts the number of input and output blocks processed by Short-Kravatte.

The terms in (3) are those of Claim 4 and an additional term. This additional term covers the case of an adversary attempting a collision in one of the branches of the Feistel network. For small values of \( n_{\text{min}} \), we believe there are no better attacks than generic ones. In particular, when \( n < 3184 \), the size of either branch is at least \( \frac{n - 8}{2} \) bits, thanks to the definition of split\([b, \ell\] function, and this explains the additional term. We do not explicitly care about the case \( n_{\text{min}} \geq 3184 \), as this additional term becomes negligible compared to the term in \( M \) in (3).

### 7.4 Rationale for the design choices

**The number of rounds in \( p_c \)** The choice of taking 6 rounds for \( p_c \) is motivated by the difficulty of generating collisions in the accumulator. Our investigations of differential propagation in Keccak-p provide evidence that differentials with high DP over a small number of rounds are dominated by a single trail \([10, 12]\). For our reasoning in this paragraph, we explicitly assume that this is the case for all differentials with relatively high DP over 5 and 6 rounds of Keccak-p. Hence, for any such differential \( (\Delta, \gamma) \) there exists one trail \( Q \) from \( \Delta \) to \( \gamma \) with \( \text{DP}(\Delta, \gamma) \approx \text{DP}(Q) = 2^{-w(Q)}. \) We will denote such a dominant trail by \( Q(\Delta, \gamma). \)

Using our assumption, we can substitute the differentials in Equation (2) in Section 5.4 by trails, yielding:

\[
\Pr(\text{collision}) \approx \sum_{\gamma} \text{DP}^2(Q(\Delta, \gamma)) = \sum_{\gamma} 2^{-2w(Q(\Delta, \gamma))}.
\]

In iterated primitives with a round function of degree 2 such as Keccak-p, the DP of a round differential (or equivalently, its weight) only depends in the difference pattern at the input of a round. Let us apply this to our 6-round trails \( Q(\Delta, \gamma) \):

\[
Q = \Delta \xrightarrow{\chi^{\text{out}}} q_1 \xrightarrow{\chi^{\text{out}}} q_2 \xrightarrow{\chi^{\text{out}}} q_3 \xrightarrow{\chi^{\text{out}}} q_4 \xrightarrow{\chi^{\text{out}}} q_5 \xrightarrow{\chi^{\text{out}}} \gamma.
\]
The weight of its last-round differential \( (q_5, \gamma) \) is fully determined by \( q_5 \) and we can express it as \( w(q_5) \). The output difference \( \gamma \) is a value that is compatible with \( q_5 \) through the round function. There are exactly \( 2^{w(q_5)} \) output differences that can be reached from \( q_5 \), with exactly the same DP. Hence, for each trail \( Q(\Delta, \gamma) \), the \( 2^{w(q_5)} - 1 \) other trails with the same first five difference patterns have the same weight. The set of \( 2^{w(q_5)} \) six-round trails that have the same sequence of differences as \( Q(\Delta, \gamma) \), except the last one, have the same weight. In \( \Box \), such sets of trails are called \textit{trail prefixes}. For the weight of a trail prefix, we take the weight of any trail in it. Clearly, \( Q(\Delta, \gamma) \) is in trail prefix \( Q'(\Delta, q_5) \). For our approximation of the collision probability above, we can now write:

\[
\Pr(\text{collision}) \approx \sum_{\gamma \text{ compatible with } q_5} 2^{-2w(Q(\Delta, \gamma))} = 2^{w(q_5)} 2^{-2w(Q'(\Delta, q_5))} = 2^{-w(Q'(\Delta, q_5)) + w(q_5)} = 2^{-w(Q'(\Delta, q_5)) - w(Q'(\Delta, q_4))}.
\]

Summarizing, Equation (5) gives the probability of a collision if we apply two messages that differ in two blocks, with the same difference \( \Delta \), making the following assumptions:

- There is a single difference \( q_5 \) such that DP\( (\Delta, q_5) \gg \gg \text{DP}(\Delta, a) \) for any \( a \not= q_5 \),
- The differential \( \text{DP}(\Delta, q_5) \) is dominated by a single trail.

If we apply \( n \) message pairs to Kravatte with the given difference, the probability to have an accumulator collision within at least one such pair is close to \( n \) times the expression of Equation (5). However, there can also be collisions between members of different pairs. Due to the symmetry in Keccak-p, we can increase the total collision probability significantly by choosing the values of the pairs in a careful way.

For difference propagation, the round function of Keccak-p is invariant with respect to translation in the direction of the \( z \)-axis. If the round differential \( (a, b) \) has some weight \( w(a) \), then any round differential \( (a \ll \tau, b \ll \tau) \) has the same weight, where \( (a \ll \tau, b \ll \tau) \) denotes \( (a, b) \) circularly shifted (or rotated) along \( z \) by some offset \( \tau \). This carries over to trails and hence if \( Q'(\Delta, q_5) \) has some weight, \( Q'(\Delta \ll \tau, q_5 \ll \tau) \) has the same weight.

We can use this to boost the collision probability in the following way. As a starter, we apply four messages \( M, M', M'' \) and \( M''' \) such that \( M_i + M'_i = M''_i = M'''_i = \Delta \) and \( M_i + M''_i = M'_i + M'''_i = \Delta \ll 1 \) and the same for blocks with index \( j \). This set has two pairs with difference \( \Delta \), two with difference \( \Delta \ll 1 \) and two with difference \( (\Delta \ll 1) + \Delta \). The former four pairs each have a collision probability given by Equation (5). The latter two pairs have an input difference \( (\Delta \ll 1) + \Delta \) that typically does not have a low-weight 5-round trail prefix starting from it. Hence, this structure of 4 inputs has a collision probability four times that of Equation (5). The two differences \( \Delta \) and \( \Delta \ll 1 \) can be seen as basis vectors of a two-dimensional vector space. We can generalize this by adding \( \Delta \ll 2, \Delta \ll 3 \), etc. as additional basis vectors. Each additional basis vector adds another difference and hence in a hypercube of dimension \( d \) there are in total \( d2^{d-1} \) pairs with collision probability of Equation (5). This technique reaches its limit when we have exhausted all shift offsets, i.e., when \( d = 64 \). A 64-dimensional vector space has \( 2^{63}64 = 2^{69} \) pairs with each a collision probability given by Equation (5).
Applying $n$ inputs with $n = n'2^{64}$ for an integer $n'$ and with the inputs structured in vector spaces described as above, gives rise to the following collision probability:

$$\Pr(\text{collision}) \approx n'2^{69}2^{-w(Q'(\Delta q_3))} = n2^{-35}2^{-w(Q'(\Delta q_3))} + w(Q'(\Delta q_4)).$$

It is easy to see that $w(Q'(\Delta q_3)) + w(Q'(\Delta q_4))$ is lower bounded by the sum of the lower bound for 5-round trail weights plus that for 6-round trails. As such, the lower bounds of 50 for the weight of trails in Keccak-p[1600, $n_r = 5$] that of 92 in Keccak-p[1600, $n_r = 6$] as proven in [20] are directly applicable. Taking these bounds would result in an estimation of the collision probability of $n2^{−92−50+5} = n2^{−137}$. Note that no trails have been found of the given weights and that it is likely that the best trails have much higher weight, leading to an even lower collision probability.

The rolling functions For understanding this section, we recommend first reading Appendix A. The rolling function roll restricted to the last plane is a linear transformation of maximum-order. As a consequence, each mask value with non-zero last plane will be in a cycle of length $2^{320}−1$. Mask values with a last plane equal to zero form fixed points for our rolling function. We think the probability that a user key $K$ maps to such a mask value is negligible.

We found our rolling function using the method proposed in [28], that goes as follows. We try candidate rolling functions with an efficient implementation until we have found one that is maximum-order. Many candidates can be generated by varying some parameters of a simple linear mapping. For each candidate, we determine the minimal polynomial of the sequence formed by one bit of the state using the Berlekamp-Massey algorithm [41]. We check whether this is a primitive polynomial of degree $e$ and if so, we have found a maximum-order rolling function. In the notation adopted in [28], the rolling function roll looks like this:

$$(x_0, x_1, x_2, x_3, x_4) \rightarrow (x_1, x_2, x_3, x_4, (x_0 \ll 7) + x_1 + (x_1 \gg 3)),$$

where $\ll$ denotes a cyclic shift (or rotation) to the left and $\gg$ a shift to the right. We looked for a rolling function where the new value of $x_4$ only depends on $x_0$ and $x_1$. This allows the parallel computation of four subsequent iterations.

The subspace properties of the rolling function are determined by its minimal polynomial:

$$1 + x^{58} + x^{74} + x^{86} + x^{102} + x^{116} + x^{118} + x^{122} + x^{129} + x^{134} + x^{137} + x^{138} + x^{144}$$

$$+ x^{148} + x^{186} + x^{187} + x^{189} + x^{195} + x^{197} + x^{203} + x^{206} + x^{211} + x^{214} + x^{215} + x^{218} + x^{221} + x^{223} + x^{229} + x^{230} + x^{231} + x^{232} + x^{239} + x^{244} + x^{246} + x^{250} + x^{251} + x^{253} + x^{256} + x^{257} + x^{259} + x^{260} + x^{261} + x^{262} + x^{265} + x^{267} + x^{272} + x^{273} + x^{275} + x^{276} + x^{279} + x^{281} + x^{282} + x^{285} + x^{287} + x^{292} + x^{293} + x^{295} + x^{296} + x^{303} + x^{305} + x^{313} + x^{320}.$$
blocks. To achieve this, the rolling function \( \text{roll}_c \) operates on a full plane and the difference between any two inputs to \( p_c \) is spread over the full plane. The application of \( \theta, \rho \) and \( \pi \) spreads this difference over all bits of the state at the input of the nonlinear step \( \chi \) of the first round. Hence, the conditions imposed by the first round differentials translate to conditions on different bit parities of the \( p_c \) input.

**Short-Kravatte** The requirement for Short-Kravatte is that for any differential \((\Delta, \sigma)\), its mean (or expected) DP over all keys \( K \), denoted as \( \text{EDP}(\Delta, \sigma) \), is below some limit \( \epsilon \). Clearly, if the output is only \( n \) bits, then \( \epsilon \) is at best \( 2^{-n} \). We conjecture that for Short-Kravatte, with output 1600 – 8 bits, \( \epsilon \) is below \( 2^{-137} \). We base this conjecture on the fact that Short-Kravatte has 10 rounds between its input and output and it is likely that the best attack is to try constructing a collision in the accumulator. For block length of the block cipher below 3184 bits the output of Short-Kravatte is truncated to roughly half this block length and for block cipher width below 274 bits \( \epsilon \) will inevitably be larger than \( 2^{-137} \).

The number of rounds in \( p_d \) and \( p_e \) For \( p_e \) we estimate that taking 4 rounds is sufficient. The difference between any pair of rolling state values is unknown and can cover a complete plane. Moreover, this difference is guaranteed to be outside the column parity kernel and hence it will quickly propagate to high-weight differences \([10]\). As for state recovery attacks based on the output only, they require an attack covering 8 rounds of Keccak-\( p \). The number of rounds in \( p_d \) is 4, so that \( p_e \circ p_d \) has algebraic degree 256. The result is that between any input and any output of Kravatte, there are always 14 rounds. We believe this to be sufficient to resist known types of cryptanalysis.

7.5 Implementations

Reference and optimized code for Kravatte will be soon made available in KeccakTools and in the Keccak code package, respectively \([11,16]\).

Following a similar work we did for the KangarooTwelve extendable output function \([15]\), we report on the speed of our current optimized implementation on the Intel® Core™ i5-6500 (Skylake). This processor supports the AVX2 instruction set with bitwise operations on 256-bit registers. Using these instructions, we can exploit the parallelism present in Kravatte and efficiently evaluate 4 instances of the Keccak-\( p \) permutation at once.

We list the result in Table \([1]\), not only of Kravatte itself, but also of Kravatte-SAE, Kravatte-SIV and Kravatte-WBC. It reports on number of cycles for in- and outputs that are short (below 200 bytes) and of intermediate size 4096 bytes and the number of cycles per byte for long in- and outputs. In general, one can observe a discontinuity as the input/output size crosses a multiple of 200 bytes. In Table \([1]\) this is apparent for Kravatte-WBC: the first bump occurs just below 400 bytes, the point where the right part of the plaintext becomes two blocks instead of one (the left part remains 1 block).

8 Conclusions

Farfalle is a versatile new construction for keyed functions in permutation-based symmetric cryptography. It can be seen as an inherently parallelizable counterpart of sponge/duplex.
<table>
<thead>
<tr>
<th><strong>Kratavte</strong></th>
<th>mask derivation</th>
<th>4/5 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>input and output less than 200 bytes</td>
<td>1045 cycles</td>
</tr>
<tr>
<td>MAC computation use case:</td>
<td>4096-byte input, output less than 200 bytes</td>
<td>3410 cycles</td>
</tr>
<tr>
<td></td>
<td>long inputs</td>
<td>0.59 cycles/byte</td>
</tr>
<tr>
<td>Stream encryption use case:</td>
<td>input less than 200 bytes, 4096-byte output</td>
<td>2880 cycles</td>
</tr>
<tr>
<td></td>
<td>long outputs</td>
<td>0.41 cycles/byte</td>
</tr>
<tr>
<td><strong>Kratavte-SAE</strong></td>
<td>processing of metadata:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>metadata less than 200 bytes, no plaintext</td>
<td>1160 cycles</td>
</tr>
<tr>
<td></td>
<td>metadata of 4096 bytes, no plaintext</td>
<td>3590 cycles</td>
</tr>
<tr>
<td></td>
<td>long metadata</td>
<td>0.62 cycles/byte</td>
</tr>
<tr>
<td>Processing of plaintext:</td>
<td>plaintext less than 200 bytes, no metadata</td>
<td>1195 cycles</td>
</tr>
<tr>
<td></td>
<td>plaintext of 4096 bytes, no metadata</td>
<td>6695 cycles</td>
</tr>
<tr>
<td></td>
<td>long plaintexts</td>
<td>1.20 cycles/byte</td>
</tr>
<tr>
<td><strong>Kratavte-SIV</strong></td>
<td>plaintext and metadata both less than 200 bytes</td>
<td>2945 cycles</td>
</tr>
<tr>
<td></td>
<td>plaintext of 4096 bytes, metadata less than 200 bytes</td>
<td>7490 cycles</td>
</tr>
<tr>
<td></td>
<td>long plaintexts</td>
<td>1.22 cycles/byte</td>
</tr>
<tr>
<td><strong>Kratavte-WBC</strong></td>
<td>long block lengths</td>
<td>1.98 cycles/byte</td>
</tr>
</tbody>
</table>

| ≤ 398 bytes          | 5200 cycles          |
| ≤ 598 bytes          | 6550 cycles          |
| ≤ 798 bytes          | 7750 cycles          |
| ≤ 998 bytes          | 8450 cycles          |
| 2048 bytes           | 11650 cycles         |
| 4096 bytes           | 14100 cycles         |
| 8192 bytes           | 21100 cycles         |
| 16384 bytes          | 36300 cycles         |

Table 1: Performance measured on Skylake.
It can better exploit resources available on high-end CPUs such as SIMD instructions. Yet, the sponge remains the best choice for unkeyed hashing and for full-state keyed duplex in keyed applications on embedded platforms where dedicated hardware can be afforded.

**Acknowledgments:** We would like to thank Monika Seidlová for her investigations on higher-order differential attacks, Kay Lukas for his investigations of rolling functions and Joost Renes for his help on finite fields.

**References**


48. The Tor Project, *Tor project: Anonymity online*, https://www.torproject.org/.
A Nonlinearity properties of linear rolling functions

In this section, we give some background on how a linear rolling function can contribute to the resistance against higher-order differential attacks. We explain higher-order differential attacks in the context of Farfalle in Section A.1. Section A.2 explains how we can investigate the resistance a rolling function can offer against such attacks and Section A.3 gives a formula to estimate these characteristics. Finally, in Section A.3 we report on some experiments on toy-size rolling functions that confirm our estimations.

A.1 Higher-order differential attacks

For building $p_b$, $p_c$, $p_d$ and $p_e$ we have in mind iterating a round function consisting of a non-linear layer and a linear layer. We think non-linear layers of algebraic degree 2 are an excellent choice for several reasons. Among others, this brings the study of difference propagation and correlation to the realm of linear algebra. The downside is that an $n_r$-round permutation only has algebraic degree at most $2^n$.

Permutations of low algebraic degree are vulnerable to attacks exploiting higher-order differentials [27]. We denote a vector space (over GF(2)) by $\langle v_i \rangle$, with $\{ v_i \}$ a set of linearly independent vectors that forms its basis. Its dimension is the number of basis vectors. An affine space is a vector space translated over an offset (vector) that is not in the basis and we denote it as $a + \langle v_i \rangle$. Its dimension is that of the vector space $\langle v_i \rangle$. In a higher-order differential attack, one exploits the fact that the (bitwise) sum of the output of a function of algebraic degree $d$ over an affine space of dimension $m$ is a function of algebraic degree at most $d - m$ of the bits of $a$. Often, one can even reduce this degree by choosing the vector space $\langle v_i \rangle$ appropriately. One could attack Farfalle by attempting to construct an input string such that the corresponding sequence of inputs to $p_c$ form an affine space of dimension higher than $d$. The contribution of such a string to the accumulator, i.e., the sum of the images of these inputs through $p_c$ is zero. Similarly, if an adversary could identify in the sequence of rolling state values $roll^i_j(y)$ a subset that forms an affine space of dimension higher than $d$, the corresponding output blocks of $p_e$ would to zero yielding a distinguisher.

Clearly, these attacks can be prevented by taking sufficiently many rounds $n_r$ and by limiting the maximum number of blocks in Farfalle. However, for computational efficiency and latency, we wish to limit the number of rounds. Therefore, adopting a rolling function that prevents forming affine spaces at the input of $p_c$ and at the input of $p_e$ is a more interesting countermeasure against higher-order algebraic attacks.

Let us take a look at the problem of generating four input blocks $m_i$ to Farfalle for index values $a$, $b$, $c$ and $d$ such that the input blocks to $p_c$ form an affine space of dimension 2 for an adversary that does not know the mask $k$. Four binary vectors $A, B, C, D$ from an affine space iff they sum to zero, i.e., $A + B + C + D = 0$. This affine space can be expressed as $A + \langle v_0, v_1 \rangle$ with $v_0 = A + B$ and $v_1 = A + C$, yielding $A = A + 0$, $B = A + v_0$, $C = A + v_1$ and $D = A + v_0 + v_1$. So the adversary must generate the input blocks $m_i$ such that $m_a + roll^a(k) + m_b + roll^b(k) + m_c + roll^c(k) + m_d + roll^d(k) = 0$ or equivalently

$$m_a + m_b + m_c + m_d = roll^a(k) + roll^b(k) + roll^c(k) + roll^d(k). \quad (6)$$

The adversary only has to guess the sum in the righthand side of Equation (6) and any set of input blocks summing to that value will do. It follows that for this to be infeasible,
the sum at the righthand side must be hard to predict for unknown k. For a linear rolling function, the sum in the righthand side of Equation (7) is a linear function of k that can be written as $M \times k$ with M a matrix that only depends on indices $a, b, c, d$. If this matrix has rank r, knowing the righthand side of Equation (7) requires correctly guessing r bits. So we must choose our rolling function such that there are no indices $a, b, c, d$ in range $[0, n]$ for some reasonable value of n that give a matrix M of low rank.

Generating sets of input blocks that form an affine space of higher dimension is even harder because multiple such equations must be satisfied. This is because any affine space has several subspaces of smaller dimensions. For example, a set of input blocks with indices $a, b, c, d, e, f, g, h$ form an affine 3-dimensional space if four equations like Equation (8) are satisfied with following index sets: \{a, b, c, d\}, \{e, f, g, h\}, \{a, b, e, f\}, \{a, c, e, g\}. Note that the indices can be grouped differently. In general, a set of $2^d$ input blocks forms an affine $d$-dimensional space if $2^d - d$ equations like Equation (9) are satisfied.

### A.2 Subspace properties of linear rolling mask sequences

Any linear permutation acting on $e$ bits can be expressed as a multiplication of the $e$-bit vector $k$ with a non-singular binary matrix $R$, so $\text{roll}(k) = R \times k$. Iterating the rolling function corresponds to exponentiating the matrix: $\text{roll}^i(k) = R^i \times k$. So now we consider the affine spaces in sets $\{R^i \times k | i \leq j < i + n\}$. A priori, this leaves many cases to investigate as there is a huge number of possible rolling functions $R$, namely $\prod_{0 \leq d \leq e} (2^e - 2^d) \approx 2^{2^e - 1}$. Moreover, even for a single rolling function there are three parameters: $k, i$ and $n$. Fortunately, this can be greatly simplified by considering equivalence.

We can transform $R$ into $F = P \times R \times P^{-1}$ for $P$ any non-singular matrix and prove following lemma.

**Lemma 1.** Let $F = P \times R \times P^{-1}$ and $I$ an index set of cardinality $2^d$ for some $d$. Then, $\{R^j \times k | j \in I\}$ is an affine space iff $\{F^j \times (P \times k) | j \in I\}$ is an affine space.

**Proof.** Clearly $F^j = P \times R^j \times P^{-1}$. Then we can write $\{F^j \times (P \times k) | j \in I\}$ as $\{P \times R^j \times k | j \in I\}$. This is simply equal to $\{R^j \times k | j \in I\}$ where $P$ is applied to all its elements. As an affine space remains an affine space after applying a linear mapping to its elements, this is an affine space. The proof for the other direction is similar. \hfill $\square$

By choosing $P$ carefully, we can obtain a matrix $F$ with a particularly simple structure, called the Frobenius normal form of $R$ [38]. It follows that we can focus our attention to such matrices. For the rolling functions we are interested in, we can simplify even more. In particular, we aim for rolling functions with the maximum-order property: the iterated application to a non-zero mask $k$ results in a single cycle of length $2^e - 1$.

The Frobenius normal form of a maximum-order mapping simply corresponds to the update function of a linear feedback shift register (LFSR) with the minimal polynomial of the Frobenius normal form as feedback polynomial [38]. It follows that we can limit our analysis to state sequences of LFSRs with primitive feedback polynomials. The state of such an LFSR at time $t$ is simply $x^t k(x) \mod p(x)$ with $p(x)$ its feedback polynomial and $k(x)$ its initial state. So, the sequences we investigate are now $\{x^t k(x) \mod p(x) | 0 \leq t < n\}$.

Let us now return to our problem at the end of Section A.1 in particular, the righthand side of Equation (9) now becomes $k(x)(x^d + x^b + x^c + x^d) \mod p(x)$: it is the multiplication of
$k(x)$ with a sum of four monomials. As these are elements in a field, their sum is simply another field element. Let $x^a + x^b + x^c + x^d \mod p(x) = d(x)$. There are now two cases: Either $d(x)$ is zero or it is non-zero. If zero, the lefthand side of Equation (1) is also zero and the adversary has an easy job in forming an affine space: she just has to choose $m_i$ values that sum to zero. If $d(x) \neq 0$, guessing the lefthand side of Equation (1) implies guessing $k$ fully as $k(x)$ can be computed from it by multiplying with the multiplicative inverse of $d(x)$. It follows that for the latter case the success probability of forming an affine space of dimension 2 is $2^{-e}$.

It follows that it should be hard to find a subset of a sequence of mask values that forms an affine space. In general, the property that we wish to have is that for any sequence $f_{roll}^j(k)$, $j < j + n$ to contain an affine space of high dimension, $n$ must be very large.

Multiplication by $k(x)$ modulo $p(x)$ is an invertible linear mapping that can be factored out and hence we can limit our analysis to sequences $\{x^t \mod p(x)\}|0 \leq t < n$. In particular, we will study the so-called affine span profile of a primitive polynomial.

**Definition 2.** The affine span for dimension $d$ of a primitive polynomial $p(x)$ is the minimum length of a sequence $\{x^i \mod p(x)\}|0 \leq i < n$ containing an affine space of dimension $d$. We denote it as $L_{min}(p(x), d)$.

We call the sequence of values $L_{min}(p(x), d)$ for increasing $d$ the affine span profile of $p(x)$.

For the case $d = 2$ there is an interesting alternative description. As four vectors $t, u, v$ and $w$ form an affine space iff $t + u + v + w = 0$, $L_{min}(p(x), 2)$ is the smallest value of $n$ such that there exist values $n_1, n_2$ smaller than $n$ such that $1 + x^{n_1} + x^{n_2} + x^n = 0 \mod p(x))$. In other words, $n$ is the degree of the polynomial with smallest degree and Hamming weight 4 that is a multiple of $p(x)$. For higher dimensions ($d > 2$), there must be multiple such polynomials.

### A.3 Estimating the affine span profile

In this section, we try to estimate the affine span profile of primitive polynomials using combinatorics and making randomness assumptions. We can verify the quality of our estimations by actually computing affine span profiles for primitive polynomials of relatively low degree.

**Lemma 2.** The probability that a random set of $2^d$ vectors of dimension $e$ forms an affine space is

$$\frac{\prod_{0 \leq i < d} 2^e - 2^i}{(2^{d-1}) \prod_{0 \leq i < d} 2^d - 2^i}.$$  

**Proof.** The total number of possible vector spaces of dimension $d$ of $e$-bit vectors is (see e.g. [18])

$$\frac{\prod_{0 \leq i < d} 2^e - 2^i}{\prod_{0 \leq i < d} 2^d - 2^i}.$$

An affine space is a vector space shifted over an offset. If we select the offset from the space orthogonal to the vector space, each choice will give another affine space. So, we
choose the offset from a space with dimension \( e - d \) and hence the total number of affine spaces of dimension \( d \) of \( e \)-bit vectors is:

\[
\frac{2^e \prod_{0 \leq i < d} 2^e - 2^i}{2^d \prod_{0 \leq i < d} 2^d - 2^i}
\]

The total probability is this expression divided by the total number of different sets of \( e \)-bit vectors of cardinality \( 2^d \), namely \( \binom{n}{2^d} \).

\[
\frac{2^e \prod_{0 \leq i < d} 2^e - 2^i}{\binom{n}{2^d} 2^d \prod_{0 \leq i < d} 2^d - 2^i} = \frac{\prod_{0 \leq i < d} 2^e - 2^i}{\binom{n}{2^d-1} \prod_{0 \leq i < d} 2^d - 2^i}.
\]

\( \square \)

For the sake of our estimation we assume a sequence of successive LFSR states behaves like a sequence of different random independent values. A subsequence of length \( n \) of the LFSR states has \( \binom{n}{2^d} \) subsets. As discussed in Section A.1, these can however not be considered independent. Namely, if \( 2^d \) elements with indices in some index set \( I \) form an affine space, this is also the case for the elements in positions \( I + j \). This partitions the \( \binom{n}{2^d} \) subsets of the length-\( n \) sequence in classes and in each class all subsets are affine spaces or none are. Each class has exactly one member with smallest index equal to 0 and hence we can fix the smallest index to 0. The total number of classes is hence \( \binom{n-1}{2^d-1} \).

If we assume that modulo this symmetry property, a sequence of LFSR states behaves like a sequence of different random independent values, the expected number of affine spaces of dimension \( d \) in a subsequence of length \( n \) of an \( e \)-bit LFSR is:

\[
\frac{(n-1)_{(2^d-1)} \prod_{0 \leq i < d} 2^e - 2^i}{\binom{n}{2^d-1} \prod_{0 \leq i < d} 2^d - 2^i} = \frac{(n-1)_{(2^d-1)} \prod_{0 \leq i < d} 2^e - 2^i}{(2^e - 1)_{(2^d-1)} \prod_{0 \leq i < d} 2^d - 2^i}.
\]

In order to manipulate this expression so that it can be used for our estimations, we simplify it by making a number of approximations, namely \( (n-1)_{(2^d-1)} \approx n^{2^d-1}, (2^e - 1)_{(2^d-1)} \approx 2^{(2^d-1)} \) and \( \prod_{0 \leq i < d} 2^e - 2^i \approx 2^{2d} \). These approximations are justified as long as \( 2^d \ll 2^e \) and \( 2^d \ll n \). This yields

\[
\frac{n^{2^d-1}}{2^{(2^d-1-d)} e \prod_{0 \leq i < d} 2^d - 2^i}.
\]

We can now estimate \( L_{\min}(p(x), d) \) for some dimensions \( d \) and \( e \). A set of \( n \) random vectors is likely to contain an affine space of dimension \( d \) if the expected number as expressed in Equation (7) equals 1. Setting it equal to 1 and solving for \( n \) yields

\[
L_{\min}(p(x), d) \approx 2^{1 - \frac{d}{2^d - 1}} \left( \prod_{0 \leq i < d} (2^d - 2^i) \right)^{\frac{1}{2^d - 1}}
\]

(8)

If we express \( L_{\min}(p(x), d) \) by its logarithm with base 2: \( L_{\min}(p(x), d) = 2^v \) we obtain a simple expression:

\[
v \approx \left( 1 - \frac{d}{2^d - 1} \right) e + e(d),
\]

with \( e(d) \) the binary logarithm of the rightmost term of Equation (8). Table 2 lists the coefficients for computing Equation (8) for small dimensions.

31
<table>
<thead>
<tr>
<th>$d$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 - \frac{d}{2^d - 1}$</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{7}$</td>
<td>$\frac{11}{15}$</td>
<td>$\frac{26}{27}$</td>
<td>$\frac{57}{57}$</td>
<td>$\frac{120}{127}$</td>
</tr>
<tr>
<td>$\epsilon(d)$</td>
<td>0.86</td>
<td>1.06</td>
<td>0.95</td>
<td>0.75</td>
<td>0.54</td>
<td>0.37</td>
</tr>
</tbody>
</table>

Table 2: Coefficients for estimating the affine span values.

Fig. 3: Measured affine span of some primitive pentanomials (marks) vs. estimates (lines).

A.4 Experimental verification

Clearly, we agree with John von Neumann that assuming a sequence of successive LFSR states behaves like a sequence of different random independent values puts us in a state of sin. To verify the validity of Equation (8) we have done some experimental verification.

We started with primitive trinomials and found that their affine span for dimension 2 does not obey Equation (8). Namely primitive trinomials systematically exhibit a low value $L_{\min}(p(x), 2)$. The reason is that low-degree multiples of trinomials exist with Hamming weight 4. For trinomial $1 + x^d + x^e$ we have $(1 + x^d + x^e)(1 + x^d) = 1 + x^{2d} + x^e + x^{d+e}$. It follows that $L_{\min}(1 + x^d + x^e, 2) = a + e$. This does not generalize to primitive polynomials of higher Hamming weight. Naturally, this does not mean that no similar effects would exist for other types of polynomials, but in our experiments, we did not observe any.

We further computed the affine span values for small dimension $d$ (2, 3 and 4) for the pentanomials listed in [51] of degrees $e$ from 8 to 48. We report the results in Figure 3. Although we see large deviations (up to a factor 5) between the measured and estimated values, the estimates clearly give the trend. The largest deviations occur for dimension 2 and as the dimension grows, the deviations get smaller.
B  On this version of the paper

We here list the changes we made in this paper since its preliminary version, posted on
the IACR ePrint archive on December 29, 2016.

We have changed the Farfalle construction in the following way:

– There is now a single algorithm for Farfalle, which integrates also the mask derivation,
as in the former Farfalle-PRF. The split between Far and Falle has been removed.
– Message blocks are \( b \) bits rather than \( r < b \) bits. In the preliminary version, a message
block and the encoding of the block index were added to the secret mask. Now the
message block is added to the mask, that itself undergoes the rolling function. This
changes the role of the rolling function. In the preliminary version, input blocks to the
permutation could not collide within a single message. Now collisions are possible.
We count on the unpredictability of the rolling masks for this to be hard. This modifi-
cation allowed us to simplify Farfalle by removing the parameter rate \( r \) and the need
for the aligned mode.
– Different strings in a sequence of strings are now separated using a blank index instead
of frame bits.
– An additional permutation step has been introduced between the accumulator and
the rolling state. This was needed to thwart higher-order differential attacks that are
generic in \( p_c \). Moreover, we specify four permutations rather than one, allowing us to
fine-tune the number of rounds for the different layers of the Farfalle computation.
– We allow a rolling function in the mask addition just before output and allow the
rolling functions at input and in the rolling state to be different.
– A specific notation allows to easily express that a chosen segment of the output stream
is requested.
– We removed the generic security claim template, and made a specific one for \textsc{Kravatte}
instead.

These changes in Farfalle of course also affect \textsc{Kravatte}. In addition, we made the follow-
ing changes to \textsc{Kravatte}:

– \textsc{Kravatte} is defined with 6, 6, 4 and 4 rounds respectively for permutations \( p_b, p_c, p_d \)
and \( p_e \) instead of 6 and 6 rounds for the now-removed Kra and Vatte.
– \textsc{Kravatte-WBC} and \textsc{Kravatte-WBC-AE} are now defined in terms of \textsc{Kravatte} and
\textsc{Short-Kravatte} in light of the removal of the separate Kra and Vatte.
– The rolling function specified for \textsc{Kravatte} now operates on a support of 320 bits as
opposed to 61 bits, and the \( n_{\text{roll}} \) limit on a single input size has been removed.
– There are two dedicated security claims, one for \textsc{Kravatte} and another one for \textsc{Kravat-
tte-WBC}. Compared to the preliminary version, we take into account adversaries
equipped with a quantum computer, and we restrict keys to be uniformly distributed.
– We expanded the design rationale specific to \textsc{Kravatte}.

The encryption modes were also improved:

– Farfalle-SAE has been changed to more efficiently handle authentication-only and
encryption-only scenarios by using frame bits for domain separation.
– Farfalle-WBC has undergone the following changes:
• For improved space efficiency, Farfalle-WBC now domain separates the \( L \) and \( R \) parts by appending frame bits instead of using standalone strings.

• Farfalle-WBC is now defined entirely in terms of two Farfalle functions \( H \) and \( F \), instead of Far and Farfalle.

• The split function has been made more elaborate in the sense that alignment and parallelism are optimized in accordance with an alignment unit length.

• A new mode Farfalle-WBC-AE has been added to demonstrate authenticated encryption implemented on top of Farfalle-WBC, with support for early rejection of invalid cryptograms.

We moved the discussion on the properties of the linear rolling functions to Appendix A to improve the flow, and we extended our numerical experiments. As we now adopt rolling functions constructed with the methods proposed in [28], we do not use the name “cara-colle” any more.

We have also added a reference, Dan Bernstein’s paper on protected counter sums [3]. This paper was brought to our attention by Bart Mennink and we were not aware of it before. It already contained several of the key ideas underlying Farfalle.

And last but not least, Seth Hoffert has joined the team of co-authors of Farfalle and Kravatte.