Farfalle: parallel permutation-based cryptography

Guido Bertoni[1], Joan Daemen[1,2], Michaël Peeters[1], Gilles Van Assche[1], and Ronny Van Keer[1]

[1] STMicroelectronics
[2] Radboud University

Abstract. In this paper, we introduce *Farfalle*, a new mode for building a pseudorandom function (PRF) from a b-bit cryptographic permutation. The constructed PRF takes as input a b-bit key and a sequence of variable-length data strings, and it generates a variable-length output. It consists of a *compression layer* and an *expansion layer*, each of them involving the parallel application of the permutation. The construction aims for simplicity and efficiency, among others with the ability to compute it for incremental inputs and with its inherent parallelism. Thanks to its input-output characteristics, Farfalle is very 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 permutation. In particular, we instantiate it with one of the Keccak-p permutations, attach concrete security claims to it and call the result Kravatte. To offer protection against attacks that exploit the low algebraic degree of the round function of Keccak-p, we do domain separation with a particular rolling function that aims at preventing the construction of input sets that form affine spaces of large dimension.

Keywords: pseudorandom function, permutation-based crypto, Keccak

1 Introduction

In symmetric cryptography, a pseudorandom function (PRF) is a function that takes as input a key and a string (or a sequence of strings), and produces a variable-length output that looks like independent uniformly-drawn random bits to an adversary not knowing the key. It is regarded as an important component for its ability to build many other cryptographic functions, such as message authentication codes (MAC), stream ciphers, key derivation functions and even block ciphers.

In this paper, we introduce *Farfalle*, a new mode for building a PRF from a b-bit cryptographic permutation. The constructed PRF takes as input a b-bit key and a sequence of variable-length data strings, and it generates a variable-length output. It consists of a *compression layer* and a *expansion layer*, each of them involving the parallel application of the permutation. The compression layer, coined Far, applies the permutation to blocks built from the key, transformed by a *rolling function* taking the block index, and from data input, and it (bitwise) adds their outputs in a b-bit *accumulator*. The *expansion layer*, coined Falle, applies for each output block the permutation to the accumulator transformed by the rolling function taking the block index and adds the key to the result.

Depicted in Figure 1, the construction aims for simplicity and efficiency. It has some features in common with the sponge construction [4]. As in sponges, the inverse of the permutation is not used. Also, an interesting feature of sponges is the ability to compute it for incremental inputs, as performed in the duplex construction [6]. For Farfalle, this is achieved by reusing the accumulator already computed and adding to it the contribution of the appended input strings.

Farfalle can be seen as a parallelizable counterpart of the sponge for keyed applications. Its inherent parallelism can be exploited on many platforms, in particular on modern processors with single-instruction multiple-data (SIMD) units. Moreover, it can be

made very efficient as the number of rounds in the permutation can be taken much smaller than in sponge-based modes, thanks to the fact that in Farfalle an attacker never has access to both the input and the output of a permutation call.

Thanks to its input-output characteristics, Farfalle is very 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 one of the Keccak-p permutations, attach concrete security claims to it and call the result Kravatte. To offer protection against attacks that exploit the low algebraic degree of the round function of Keccak-p, we define the caracolle() rolling function that aims at preventing the construction of input sets that form affine spaces of large dimension. Reference and optimized code for Kravatte will be soon made available in KeccakTools and in the Keccak code package, respectively [8,12].

1.1 Overview of the paper

After introducing notation and conventions in Section 1.2, we specify the Farfalle construction, its immediate application as a PRF and its security claim in Section 2. Then, in Section 3, we define (authenticated) encryption modes: a synthetic initial value (SIV) mode for authenticated encryption (AE), a session-supporting AE mode and a wide block cipher. Section 4 gives a rationale for the basic construction and these modes. Section 5 is dedicated to caracolle(), the rolling function that we use in Kravatte and that can be used in conjunction with any permutation that has low algebraic degree. Finally, in Section 6 we specify Kravatte, a concrete instantiation of Farfalle making use of Keccak-p, the permutation underlying Keccak, make a security claim and provide some rationale.

1.2 Notation

The length of a string X is denoted $|X|$. The concatenation of strings X and Y is written as $X||Y$. The set of bit strings of length n is $\mathbb{Z}_2^n$ and the set of strings of any length is $\mathbb{Z}_2^*$. We denote the first l bits of a string X as $\lfloor X \rfloor_l$. The number of r-bit blocks of a string X is defined as

$$|X|_r = \max\left(\left\lfloor \frac{|X|}{r} \right\rfloor, 1\right).$$

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

When applied to bit strings, the $+$ operator is the bitwise modulo-2 addition (or exclusive-or). When adding two strings of different lengths, the length of the result is that of the shorter string. In other words, for $X = (x_0, \ldots, x_{|x|-1})$ and $Y = (y_0, \ldots, y_{|y|-1})$, the result $Z = X + Y$ satisfies $|Z| = \min(|X|, |Y|)$ and $z_i = x_i + y_i$ in $\mathbb{Z}_2$ with $0 \leq i < |Z|$.

We use uppercase characters for variable-length strings and lowercase characters for fixed-length strings (e.g. b-bit blocks).

2 Specification of the Farfalle construction, PRF and security claim

We define Farfalle as the composition of two auxiliary constructions, called Far and Falle. The three of them are based on:

- a b-bit permutation f;
- a rate r in bits;
- a rolling function roll(k, i).

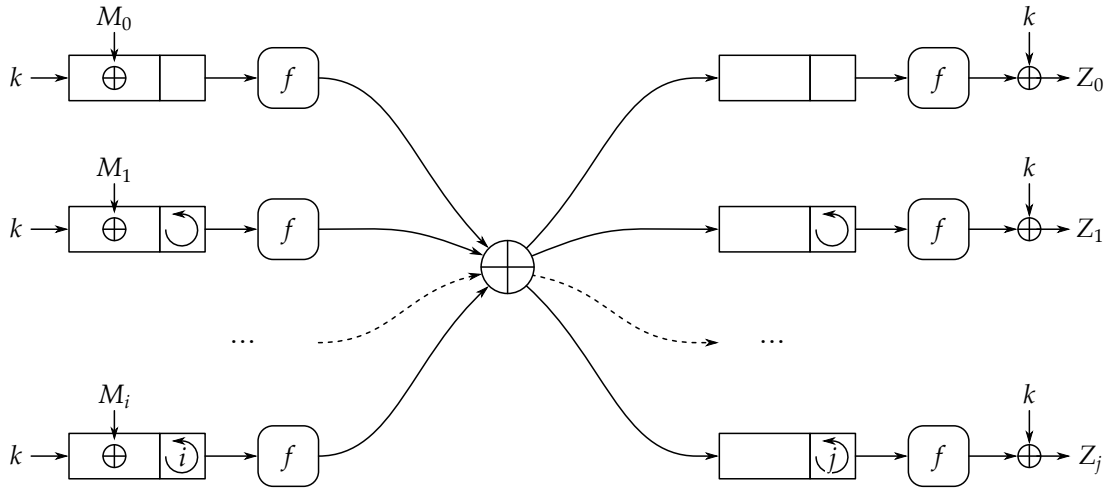

Fig. 1. The Farfalle construction

2.1 Rolling function

A *rolling function* maps $\mathbb{Z}_2^b \times \mathcal{R}_{\text{roll}}$ to $\mathbb{Z}_2^b$, with $\mathcal{R}_{\text{roll}} = \mathbb{Z}_{n_{\text{roll}}}$ the set of integers from zero to some specified limit n_{roll}. We require that for any $k \in \mathbb{Z}_2^b$, the mapping from $i \in \mathcal{R}_{\text{roll}}$ to $\text{roll}_{\text{end}}(k, i)$ is injective, where $\text{roll}_{\text{end}}(k, i)$ is equal to $\text{roll}(k, i)$ restricted to its last $b - r - 2$ bits.

The purpose of the rolling function is to yield a sequence of distinct state values, hence properly domain-separating the different blocks of input and of output. This prevents in particular an attack that would reorder blocks.

In practice, the rolling function can be a function that, when going from i to $i + 1$, leaves the first $b - e$ bits unchanged and applies some invertible transformation to the last e bits, with $e \leq b - r - 2$. The state values restricted to their last e bits form a sequence without repetition of length at least n_{roll}. This is what we are going to assume in the sequel.

In addition, we informally require that $\text{roll}(k, i)$ is as unpredictable as k, so that $\text{roll}(k, i)$ can effectively whiten the b-bit state before being input to the permutation f. Furthermore, it is desired that for any given $i \neq i'$, the difference $\Delta = \text{roll}(k, i) + \text{roll}(k, i')$ is hard to predict without knowing k (except of course that Δ is limited to the last e bits).

2.2 Far-Falle-Farfalle

Far builds a *keyed compression function* $y = H_k(M \dots)$ that takes as input a key $k \in \mathbb{Z}_2^b$ and a chain of at least one string, and that returns a b-bit string. The definition of Far is given in Algorithm 1. Its output is called the *accumulator*.

The Far construction enjoys the *incremental property*: When doing multiple Far computations with the same key and on chains with same prefix, the computation on that prefix can be factored out, i.e.,

$$H_k\left(M^{(m)} \circ M^{(m-1)} \circ \cdots \circ M^{(0)}\right) = H'_k\left(M^{(m)}, \alpha_m\right) + H_k\left(M^{(m-1)} \circ \cdots \circ M^{(0)}\right).$$

In addition, for any string that keeps its value and stays in the same position, the value of $H'_k(M, \alpha)$ can be reused.

Similarly, we define an auxiliary construction called Falle for building a keyed expansion function $Z = S_k(y, n)$ based on the same ingredients as Far. The input is a b-bit

string and the output is a string of some length n as requested by the caller. There are two flavours of Falle indicated by an option in the definition: one that truncates the outputs of f to r bits before concatenating them and one that just concatenates the b-bit outputs. In the former option, processing on input in Far and output in Falle have equal block-length and it is therefore called *Aligned*. In the latter option the permutation outputs are fully used and it is called *Full*. The construction Falle is defined in Algorithm 2. In our notation we omit the output length if it is clear from the context. In particular, if we write $X + S_k(y)$, we assume that the desired output length is $|X|$.

Farfalle is simply defined as applying Falle to the output of Far, both defined with the same parameters. We provide the definition in Algorithm 3 and an illustration in Figure 1.

2.3 Pseudorandom function with variable key length

We define a PRF with variable key length making use of Farfalle. It takes as input a secret key K of variable length, a chain of data strings, and it returns an output of desired length. It first derives a Farfalle key $k \in \mathbb{Z}_2^b$ from the key K, making use of Far, and then just applies Farfalle with k to the chain of data strings. We provide a formal specification in Algorithm 4. The PRF can be readily used for MAC computation, key derivation and keystream generation. It naturally inherits the incremental property of Farfalle.

2.4 Expressing security claims

We now give two security claims, one for instances of Farfalle and another for combined instances of Far and Farfalle. Both are parameterized by the value of the *claimed capacity* c_{claim} and address the case where the key is derived as $k = H_{0^b}(K)$.

The first claim is for Farfalle-PRF. It applies also to all modes defined in terms of it. This includes all modes defined in this document, with the exception of the wide block cipher mode.

Definition 1. *We say that a given Farfalle instance $F = Farfalle[f, r, \text{roll}, Full]$ is* Farfalle-secure *with claimed capacity c_{claim} if the following condition is satisfied. Let $\mathbf{K} = (K_0, \ldots, K_{u-1})$ be an array of u secret keys, each with a min-entropy of at least κ bits and with a probability of collision of at most $\binom{u}{2}2^{-\kappa}$, and let $k_i = H_{0^b}(K_i)$ for $0 \le i \le u-1$. Then, the advantage of distinguishing the array of functions $F_{k_i}(\cdot)$ with $i \in \mathbb{Z}_u$ from $\mathcal{RO}(i, \cdot)$, is at most*

$$\frac{uN + \binom{u}{2}}{2^\kappa} + \frac{MN}{2^{c_{claim}}} + \frac{M^2}{2^{c_{claim}+1}}. \tag{1}$$

Here,

- *N is the* computational complexity *expressed in the number of calls to f, and*
- *M is the* online *or* data complexity *expressed in the number of calls to f when computing the queries to F.*

We formulate an additional claim for Far for modes making use of both Farfalle and Far, such as the wide block cipher defined in Section 3.3.

Definition 2. *We say that a pair of Far and Farfalle instances $H = Far[f, r, \text{roll}]$ and $F = Farfalle[f, r, \text{roll}, Full]$ is* jointly Farfalle-secure *with claimed capacity c_{claim} if, in the same circumstances as in Definition 1, the probability of distinguishing $\big(\mathcal{RO}_1(\Delta + H_{k_i}(\cdot)), F_{k_i}(\cdot)\big)$ from $(\mathcal{RO}_2(i, \cdot, \Delta), \mathcal{RO}(i, \cdot))$, with $i \in \mathbb{Z}_u$ and $\Delta \in \mathbb{Z}_2^b$, is at most the expression given in Eq. (1).*

Algorithm 1 Definition of Far$[f, r, \text{roll}]$

Single-input: $H' : \mathbb{Z}_2^b \times \mathbb{Z}_2^* \times \mathbb{N} \to \mathbb{Z}_2^b$
Input: key $k \in \mathbb{Z}_2^b$, data string $M \in \mathbb{Z}_2^*$ and start index $\alpha \in \mathbb{N}$ ($\alpha = 0$ by default)
Split M in $n = |M|_r$ blocks of r bits M_0 to M_{n-1}, with $|M_i| = r$ for $i < n-1$ and $|M_{n-1}| \leq r$
Define $p_i = M_i||10$, for $0 \leq i < n-1$
Define $p_{n-1} = M_{n-1}||10^*1$, with $|p_{n-1}| = r+2$
return $H'_k(M, \alpha) = \sum_{0 \leq i < n} f\left(p_i + \text{roll}(k, \alpha + i)\right)$

Chaining on multiple input strings: $H : \mathbb{Z}_2^b \times (\mathbb{Z}_2^*)^+ \to \mathbb{Z}_2^b$
Input: key $k \in \mathbb{Z}_2^b$ and data strings $M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)} \in (\mathbb{Z}_2^*)^+$
return $H_k\left(M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)}\right) = \sum_{i=0}^{m-1} H'_k\left(M^{(i)}, \alpha_i\right)$, with $\alpha_i = \sum_{j<i} |M^{(j)}|_r$

Algorithm 2 Definition of Falle$[f, r, \text{roll}, \text{option}]$

$S : \mathbb{Z}_2^b \times \mathbb{Z}_2^b \times \mathbb{N} \to \mathbb{Z}_2^*$
Input: key $k \in \mathbb{Z}_2^b$, secret $y \in \mathbb{Z}_2^b$, and requested output length n
Initialize $Z \leftarrow$ empty string, and $i \leftarrow 0$
while $|Z| < n$ **do**
  $z = k + f\left(\text{roll}(y, i)\right)$
  if option = Aligned **then** $z \leftarrow \lfloor z \rfloor_r$
  $Z \leftarrow Z||z$
  $i \leftarrow i + 1$
end while
return $S_k(y, n) = \lfloor Z \rfloor_n$

Algorithm 3 Definition of Farfalle$[f, r, \text{roll}, \text{option}]$

Definitions
$H = \text{Far}[f, r, \text{roll}]$
$S = \text{Falle}[f, r, \text{roll}, \text{option}]$

$F : \mathbb{Z}_2^b \times (\mathbb{Z}_2^*)^+ \times \mathbb{N} \to \mathbb{Z}_2^*$
Input: key $K \in \mathbb{Z}_2^b$, data strings $M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)} \in (\mathbb{Z}_2^*)^+$ and requested output length n.
$y \leftarrow H_k(M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)})$
return $F_k(M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)}, n) = S_k(y, n)$

Algorithm 4 Definition of Farfalle-PRF$[f, r, \text{roll}, \text{option}]$

Definitions
$H = \text{Far}[f, r, \text{roll}]$
$F = \text{Farfalle}[f, r, \text{roll}, \text{option}]$

Key Schedule taking key $K \in \mathbb{Z}_2^*$ and returning $k \in \mathbb{Z}_2^b$
$k \leftarrow H_{0^b}(K)$

Data Path taking key $k \in \mathbb{Z}_2^b$ and sequence of input strings $M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)} \in (\mathbb{Z}_2^*)^+$, requested length n and returning $Z \in \mathbb{Z}_2^*$
return $F_k(M^{(m-1)} \circ \cdots \circ M^{(1)} \circ M^{(0)}, n)$

In (1), 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 should be as hard as recovering c_{claim} secret bits, when the adversary has access to M output values it can compare to. The third term expresses that the effort of finding a collision in the accumulator should be as hard as finding a collision for a random oracle with a c_{claim}-bit output.

Clearly, the second claim implies the first one. While Definition 1 is a simple PRF claim on Farfalle-PRF, Definition 2 claims in addition that Far is a PRF after filtering through a random oracle. To succeed in distinguishing $\mathcal{RO}_1(\Delta + H_{k_i}(M))$ from $\mathcal{RO}_2(i, M, \Delta)$, the adversary is limited to observing equality in the expression $\Delta + H_{k_i}(M)$ for chosen inputs (i, M, Δ). In other words, she is successful if she can find outputs with a predictable difference Δ, i.e., $H_{k_i}(M) + H_{k_j}(M') = \Delta$ for $(i, M) \neq (j, M')$.

3 Encryption modes

We now define three (authenticated) encryption modes that make use of Farfalle. Like in Farfalle-PRF, the key is derived as $k = H_{0^b}(K)$ and this step can naturally be pre-computed for several uses with the same key.

3.1 SIV authenticated encryption scheme

We define a so-called synthetic initial value (SIV) *authenticated encryption scheme* [30,24] making use of Farfalle. It takes as input a secret key K of variable length, plaintext P and metadata A, all variable-length, and returns a ciphertext C with the same length as the plaintext and a fixed-length tag T. It first derives a Farfalle key k from a user key K making use of Far, then computes the tag by applying Farfalle with k to $P \circ A$ and finally performs the encryption by adding to P the output of Farfalle with k to $T \circ A$. We provide a formal specification in Algorithm 5.

Algorithm 5 Definition of Farfalle-SIV$[f, r, \text{roll}, t]$ with t the tag length

Definitions
$H = \text{Far}[f, r, \text{roll}]$
$F = \text{Farfalle}[f, r, \text{roll}, \text{Aligned}]$

Key Schedule taking key $K \in \mathbb{Z}_2^*$ and returning $k \in \mathbb{Z}_2^b$
$k \leftarrow H_{0^b}(K)$

Wrap taking metadata $A \in \mathbb{Z}_2^*$ and plaintext $P \in \mathbb{Z}_2^*$, and returning ciphertext $C \in \mathbb{Z}_2^{|P|}$ 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 $A \in \mathbb{Z}_2^*$, ciphertext $C \in \mathbb{Z}_2^*$ and tag $T \in \mathbb{Z}_2^t$, and returning ciphertext $C \in \mathbb{Z}_2^{|P|}$ 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!
end if

Note that the compression of A in common to both calls to F_k must be done only once, thanks to the incremental property.

The security of this mode relies on Farfalle-PRF to be a PRF. In addition, it depends on the length t of the tag T. In general, if the value of A can repeat, then T must be long enough to make the probability of colliding tags negligible. A collision in T implies that the same keystream is used to encrypt two different plaintexts, resulting in a confidentiality break. If the value of A is guaranteed to be a nonce, then the keystream is repeated only if there is a collision in $H_k(T \circ A)$.

3.2 Session-supporting authenticated encryption scheme

We define a session-supporting *authenticated encryption scheme* based on Farfalle, similar to the Motorist mode [10]. The session keeps track of a *history* that is presented to an instance of Farfalle with the key k for generating tags and keystream for encryption/decryption. Starting a session initializes the history to a nonce N. From then on, messages consisting of metadata A and plaintext P are presented. The wrapping of a message consists of appending the metadata to the history and encrypting the plaintext by adding to it the output of Farfalle-PRF applied to the history. Then the ciphertext is appended to the history and the tag is the output of Farfalle-PRF applied to the history. Unwrapping is similar. We provide a formal specification in Algorithm 6.

Algorithm 6 Definition of Farfalle-SAE$[f, r, \text{roll}, t]$ with t the tag length

Definitions
$H = \text{Far}[f, r, \text{roll}]$
$F = \text{Farfalle}[f, r, \text{roll}, \text{Aligned}]$

Initialization taking key $K \in \mathbb{Z}_2^*$ and nonce $N \in \mathbb{Z}_2^*$, and return tag T
$k \leftarrow H_{0^b}(K)$
$T \leftarrow 0^t + F_k(N)$
history $\leftarrow N$
return T

Wrap taking metadata $A \in \mathbb{Z}_2^*$ and plaintext $P \in \mathbb{Z}_2^*$, and returning ciphertext $C \in \mathbb{Z}_2^{|P|}$ and tag $T \in \mathbb{Z}_2^t$
$C \leftarrow P + F_k(A \circ \text{history})$
$T \leftarrow 0^t + F_k(C \circ A \circ \text{history})$
history $\leftarrow C \circ A \circ$ history
return C, T

Unwrap taking metadata $A \in \mathbb{Z}_2^*$, ciphertext $C \in \mathbb{Z}_2^*$ and tag $T \in \mathbb{Z}_2^t$, and returning ciphertext $C \in \mathbb{Z}_2^{|P|}$ or an error
$T' \leftarrow 0^t + F_k(C \circ A \circ \text{history})$
if $T' = T$ **then**
  $P \leftarrow C + F_k(A \circ \text{history})$
  history $\leftarrow C \circ A \circ$ history
  return P
else
  return error!
end if

Naturally, thanks to the incremental property, an implementation does not have to maintain the history as a chain of string, but only as a b-bit accumulator. In every call to Farfalle-PRF only the recently appended string must be compressed.

Like in Motorist, the initialization returns a tag that can be sent along with the nonce (sender) or verified at the beginning of a session (receiver).

3.3 Wide block cipher

We define a *tweakable wide block cipher* based on Farfalle. The global construction is an instantiation of the HHFHFH mode as presented by Dan Bernstein at the Symmetric Cryptography Dagstuhl seminar in January 2016 [2], that is in turn based on work of Naor and Reingold [28], that is based on a paper by Stefan Lucks [26], that builds further on work of Luby and Rackoff [25].

It takes as input a secret key K a plaintext P and a tweak W, all variable-length, and returns a ciphertext C of same length as the plaintext. It first derives a Farfalle key k from the key K making use of Far, and then performs a 4-round Feistel network to the plaintext. The latter is split in a left and right part. The functions used in the first and last round apply Far to a part of the state and the two middle rounds apply Farfalle to part of the state and the tweak. We provide a formal specification in Algorithm 7.

The function split() splits the plaintext P into a left part L and a right part R in the following way. Let us denote the length of P in bits by n, the length of the left part by n_L and the length of the right part by n_R. split() operates as follows:

- If $n < 2r$, $n_R = \lfloor n/2 \rfloor$ and $n_L = n - n_R$.
- If $n \geq 2r$, $n_L = 2^x r$ with $x = \left\lfloor \log_2 \frac{n-r}{r} \right\rfloor$ the largest exponent such that $n - 2^x r \geq r$. So L gets the first 2^x r-bit blocks of P and R gets the remainder, containing at least one r-bit block.

Algorithm 7 Definition of encryption in Farfalle-WBC$[f, r, \text{roll}]$

Definitions
$H = \text{Far}[f, r, \text{roll}]$
$F = \text{Farfalle}[f, r, \text{roll}, \text{Aligned}]$

Key Schedule taking key $K \in \mathbb{Z}_2^*$ and returning $k \in \mathbb{Z}_2^b$
$k \leftarrow H_{0^b}(K)$

Encipher taking key $k \in \mathbb{Z}_2^*$, tweak $W \in \mathbb{Z}_2^*$ and plaintext $P \in \mathbb{Z}_2^*$ and returning plaintext $C \in \mathbb{Z}_2^{|P|}$
$(L, R) \leftarrow \text{split}(P, r)$
$R_0 \leftarrow R_0 + H_k(L \circ 0)$, with R_0 the first $\min(b, |R|)$ bits of R
$L \leftarrow L + F_k(R \circ W \circ 1)$
$R \leftarrow R + F_k(L \circ W \circ 0)$
$L_0 \leftarrow L_0 + H_k(R \circ 1)$, with L_0 the first $\min(b, |L|)$ bits of L
return $C \leftarrow L||R$

The values of $H_k(0)$ and $H_k(1)$ can be pre-computed together with the key schedule, at the cost of $2b$ extra bits of memory. Similarly, the contribution $H'_k(W, 1)$ of W to the accumulator can be shared among the computations of $F_k(R \circ W \circ 1)$ and of $F_k(L \circ W \circ 0)$.

4 Rationale and comparison with prior art

We give a rationale for the Farfalle construction in Section 4.1, for the pseudorandom function mode in Section 4.2, for the SIV authenticated encryption mode in Section 4.3, for the session-supporting authenticated encryption mode in Section 4.4 and finally for the wide block cipher in Section 4.5.

4.1 Farfalle

A Farfalle function loaded with a secret key k can be distinguished from a random function in several ways, among others:

- *accumulator collision*: finding two inputs leading to identical accumulators, or with values differing only in the last $b - r - 2$ bits, without guessing k;
- *accumulator retrieval*: reconstructing its value from the output, without guessing k;
- *key retrieval*: finding the value of k from input-output pairs.

Finding two inputs M and M' leading to the same value of the accumulator yields identical outputs. If the values y, y' of the accumulator differ only in the last $b - r - 2$ bits, then there may be some indexes i and i' such that $\text{roll}(y, i) = \text{roll}(y', i')$ and this equality results in output block i for input M colliding with output block i' for input M'. Clearly, the indexes i and i' depend on the particular difference in the accumulator and its absolute value and are hence not known by the attacker. It follows that the adversary can only observe the equality of output blocks if both i and i' are smaller than the requested output length for M and M' respectively.

The difficulty of finding collisions in the accumulator is mostly a security property of Far and the fact that the accumulator cannot be directly observed in Farfalle. Before being subject to f, each input blocks p_i is whitened with the key k. One may try to apply two inputs differing only in two blocks. Each of these blocks defines an input differential. The probability of a collision is then the probability that the output difference is of the form $0^{r+2}||\Delta$. This is directly linked to the differential propagation probabilities of the permutation f with output truncated to its first $r + 2$ bits. Note that this attack involves two differentials over f, one for each input block. If the input differences are δ and δ', then the probability of success is $\sum_\gamma \text{DP}(\delta, \gamma)\text{DP}(\delta', \gamma)$ with $\text{DP}(x, y)$ the differential probability of the differential (x, y) over f with output truncated to its first $r + 2$ bits. One can choose the input blocks M_i, M_j and M'_i, M'_j in such a way that they differ very little. In this way, one can make sure that the difference propagation is synchronous in both differentials. However, the unpredictability of the difference due to the rolling function makes this difficult to implement. Another approach would be the application of higher-order differentials. For this we refer to Section 5.1.

Retrieving the accumulator from the output could be done by comparing output blocks for different block indices. The attacker does not know the difference between $\text{roll}(y, i)$ and $\text{roll}(y, i')$, but knows that it is limited to the last $b - r - 2$ bits. A strategy could be to observe the output difference and to try to reconstruct the accumulator value. This strategy is also directly linked to the difference propagation properties of f, where the input difference is limited to the last $b - r - 2$ bits of the input. Determining the accumulator by other means is made difficult by the fact that the key k is added bitwise to the result of f before being output.

Finally, finding the value of k from input-output pairs can be seen as a variant to doing key retrieval in an Even-Mansour [19] cipher, where the permutation consists of two iterations of f. The main difference is that in Farfalle an attacker has the additional degree of freedom of exploiting $\text{roll}(y, i)$ to alter the last $b - r - 2$ bits of the accumulator that forms the intermediate state in the middle of the permutation.

Informally, the properties we require from f are good differential propagation properties, especially for input differences restricted to the last $b - r - 2$ bits of the state and some minimum resistance against higher-order differential attacks. Interestingly, between any input and output, there are two iterations of f. The attacker has some additional degrees of freedom provided by the rolling function in between, but a good choice of the rolling

function can greatly limit its exploitability. Furthermore, the use of a rolling function, as compared to simply bitwise adding the binary encoding of the block index, makes differential attacks harder since the differences at the input of Falle depend on the absolute value of the accumulator.

The Farfalle construction reminds of the keyed sponge construction, of which the most efficient version is the full-width keyed sponge [5,1,27]. It differs in that the keyed sponge is strictly serial while Farfalle consists of two phases 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 f being available to the adversary. This implies that for equal safety margins in Farfalle one can afford to take a permutation 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 much smaller permutations for the same target security strength. The rate is limited to $b - 2 - e$ with e the number of bits used by the rolling function. By limiting e, i.e., limiting the total number of blocks that can be processed in a single call to a Farfalle function, one can use a large part of b for the rate and applying Farfalle to permutations as small as 200 bits can still be quite efficient.

4.2 Pseudorandom function

Pseudorandom functions based on Farfalle-PRF 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 f for setting up the key and then one permutation call per input block and one per output block. We compare to some similar MAC modes (setting aside the fact that these modes do not support arbitrary-length outputs):

- Alred [16,18] was one of the first permutation-based modes proposed for MAC computation and is mostly known for the instance Pelican-MAC based on AES [17]. The main difference with Farfalle-PRF 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-PRF 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.
- PMAC [13] is essentially a block cipher mode for MAC computation. However, Far is similar to it as it performs the block cipher calls in parallel and (bitwise) adds their outputs. 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.
- MARVIN [22,21] is a mode for MAC computation that was inspired at the same time by Alred and by PMAC and uses a block cipher and a permutation. Far is similar to the compression phase 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 [23] 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 variable-length inputs):

- The counter mode of a block cipher: no distinction is made between the long-term *nonce* and the short term block index. Depending on the size of the nonce, an attacker can apply differences of large choice and observe corresponding outputs.
- The mode underlying Salsa and ChaCha [3]: this is very close to counter mode of a block cipher, where the block cipher is rather of type Even-Mansour. Again the attacker has more degrees of freedom than in Farfalle-PRF. By instantiating Farfalle-PRF with the Salsa or ChaCha permutation, it is likely that less rounds are needed for the same safety margin. Due to the fixed cost of two permutation calls (one permutation call for deriving k from K and one for absorbing the nonce), for short keystreams Farfalle-PRF would be less efficient. However, starting from a few blocks, this would be compensated for by the lower number of rounds.

Finally, we can compare to the pseudorandom function HS1 that is used in the CAESAR submission HS1-SIV [24]. This pseudorandom permutation 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-PRF has the advantages that it requires only a single permutation, it is in general much simpler and it takes sequences of input strings rather than a single one, simplifying modes built on top of it.

4.3 SIV authenticated encryption mode

Our mode Farfalle-SIV is a close variant of the SIV construction [30], mostly for the purpose of key wrapping and that was later adopted in the CAESAR submission HS1-SIV [24]. The main advantage of Farfalle-SIV in comparison to its two examples is the following. In the 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), which 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 Farfalle-SIV there is 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. This comes no extra cost thanks to the incremental property of Far.

4.4 Session-based authenticated encryption mode

For Farfalle-SAE a comparison with duplex-based session-supporting authenticated encryption seems appropriate. The most recent mode realizing that is the Motorist mode underlying Keyak [10]. 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 plaintext P or metadata A.

When considering protections against side channel attacks, Motorist is at the advantage. The 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 differential power (or electromagnetic) analysis (DPA/DEMA) attack can only be conducted to the session setup phase. In Farfalle-SAE the key k is applied for each call to f and hence the attack surface for DPA/DEMA is much larger.

4.5 Wide block cipher

The novelty of Farfalle-WBC with respect to HHFHFH [2] is that thanks to the incremental property of Far, the compression of the tweak must be done only once.

Another well-known tweakable wide block cipher construction is AEZ-Core proposed in the CAESAR submission AEZ [20]. It is not easy to compare Farfalle-WBC with AEZ-Core, as the former is based on permutations and the latter on tweakable block cipher. Although it is hard to measure simplicity, we feel that Farfalle-WBC is a significantly simpler construction than AEZ-Core.

5 The caracolle() rolling functions

A rolling function takes a b-bit state and transforms its last e bits according to an index i in a range $\mathcal{R}_{\text{roll}}$. In this section we propose caracolle(), a type of rolling function for use in Farfalle if instantiated with permutations of relatively low algebraic degree. It can be seen as variant of the cryptographic CRC introduced in [23], however, with slightly different objectives. We give some background on higher-order differential attacks in Section 5.1, present the basic mechanism in Section 5.2 and estimate its desired nonlinearity properties in Section 5.3.

5.1 Impact of the rolling function on higher-order differential attacks

For the permutation, we have in mind an iterated permutation with a round function consisting of a non-linear layer and a linear layer. We wish to limit the algebraic degree of the non-linear layer to 2 to minimize the overhead due to countermeasures against side channel attacks such as masking or threshold sharing. This has as side effect that an n_{r}-round permutation only has degree at most $2^{n_{\text{r}}}$. Permutations of low algebraic degree are vulnerable to cube and other attacks exploiting higher-order differentials. 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 a set of inputs $\langle a + v_i \rangle$, with $\langle v_i \rangle$ a vector space of dimension m, is a function of algebraic degree at most $d - m$. Often, one can even reduce this degree by choosing the vector space appropriately.

If there was no rolling function, the input to the permutation f could be chosen freely modulo the secret key k. The adversary could apply an input of $2^{2^{n_{\text{r}}}}$ blocks that forms a vector space. The resulting value of the accumulator would be a degree-0 function of the key k and hence a constant independent of it. Naturally, there must be some influence of the block index in the input of the permutation. We could have simply encoded the index i in binary form as part of the block p_i. In that case too, the adversary could easily form a vector space with $2^{2^{n_{\text{r}}}}$ input blocks. Similarly, if in Falle one could find an affine space of dimension $2^{n_{\text{r}}}$ in the encodings of the indices, summing the corresponding output blocks would result in a constant value allowing to mount a distinguishing or keystream prediction attack. Clearly, by taking sufficientlly many rounds n_{r} and by limiting the maximum

number of blocks in Farfalle these attacks can be prevented. On the other hand, techniques exist to peel off some rounds in the beginnning and the end, so it is appropriate to take some safety margin.

An interesting countermeasure against higher-order algebraic attacks is a rolling function that prevents forming affine spaces at the input of the permutations in Farfalle. The property that we wish to have is that $\{\text{roll}(k,i)|i \in \mathcal{R}_{\text{roll}}\}$ does not contain affine spaces that have dimension above some limit dimension d_{lim}. The achievable limit dimension depends on e, the number of bits that roll works on and on $n_{\text{roll}} = |\mathcal{R}_{\text{roll}}|$.

We consider the minimum length of a sequence of values $\{\text{roll}(k,j)|i \leq j < i+n\}$ starting from some index i that contains an affine space of dimension d and want this to be high for all i. If a rolling function is injective over $n = 2^e$, then the code comprises all possible codewords and for $n = 2^e$ it contains an affine space of dimension e. However, if we limit the range of the index to some value $n_{\text{roll}} < n$, it is reasonable to expect the minimum length of sequences to grow much faster than 2^d. We investigate how much faster in Section 5.3.

5.2 Definition of the caracolle() rolling functions

It turns out that the cryptographic CRC of [23], when applied to the last e bits of a state is a rolling function with low implementation cost and excellent nonlinearity properties. We call it caracolle(). For a given rolling function block length e, we limit the index range $\mathcal{R}_{\text{roll}}$ to $[0, n_{\text{roll}} - 1]$ with $n_{\text{roll}} = 2^{e-x}$ and x a small constant, e.g., $x = 4$.

Our rolling function simply applies exponentiation in GF(2^e): rolling by an index i corresponds with interpreting the last e bits of the state as a binary polynomial $k(x)$ and multiplying it with the polynomial x^i modulo a primitive polynomial $p(x)$ of degree e. Hence, the output of the rolling function is $\left(k(x) \times x^i\right) \bmod p(x)$. In order to avoid the case $k(x) = 0$, leading to a sequence of equal state values, we set the value of the constant term in $k(x)$ to 1. We provide a formal specification in Algorithm 8.

Algorithm 8 Definition of caracolle$[p(x)]$, with e the degree of $p(x)$

Input: state $k \in \mathbb{Z}_2^b$ and index $i \in \mathcal{R}_{\text{roll}}$
Output: state k'
Map last e bits of k to bitstring s: $\forall 0 \leq j < e : s_j = k_{b-e+j}$
Force the first bit of s to 1: $s_0 \leftarrow 1$
Convert s to a binary polynomial $s(x) = \sum_{0 \leq j < e} s_j x^j$
$s(x) \leftarrow s(x) \times x^i \bmod p(x)$
Convert binary polynomial $s(x)$ to bitstring s with s_j the coefficient of x^j in $s(x)$
Set $k' = \lfloor k \rfloor_{b-e} || s$
return $k' = \text{caracolle}(k, i)$

The exponentiation can be implemented by a simple Galois-type linear feedback shift register (LFSR), where the feedback polynomial $p(x)$ is the minimal polynomial of x. The cells of the feedback register contain the representation of the element with respect to the basis given by elements $1, x, x^2$ up to x^{e-1}. For Farfalle, the implementation typically needs to evaluate $\text{roll}(k,i)$ sequentially starting from $i = 0$. Computing $\text{caracolle}(k,0)$ comes down to setting a bit to 1, and generating $\text{caracolle}(k,i+1)$ given $\text{caracolle}(k,i)$ amounts to a single shift and a bitwise XOR instruction. In implementations that use masking to protect against side channel attacks, thanks to the linearity of this operation it can be performed on shares separately.

5.3 Nonlinearity properties of caracolle()

In this section we estimate the minimum length of sequences of values $\{\text{caracolle}(k, jf) | i \leq j < i + n\}$ that contain an affine space of dimension d. We start by proving the following lemma.

Lemma 1. *The probability that a random set of 2^d vectors of dimension e forms an affine space is*

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

Proof. The total number of possible vector spaces of dimension d of e-bit vectors is (see e.g. [14])

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

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 $\binom{2^e}{2^d}$.

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

□

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. These can however not be considered independent. Namely, the element at position a is x^a. The four elements in positions a, b, c, d, form an affine space if $x^a + x^b + x^c + x^d = 0$. From this it follows that the four elements in positions $a + i, b + i, c + i, d + i$ for any i also form an affine space as $x^{a+i} + x^{b+i} + x^{c+i} + x^{d+i} = x^i(x^a + x^b + x^c + x^d$. In general, if 2^d elements in positions $\{a_0, a_1, \ldots\}$ form an affine space, this is also the case for the elements in positions $\{a_0 + i, a_1 + i, \ldots\}$. This partitions the $\binom{n}{2^d}$ subsets of the length-n 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:

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

To get some intuition, we simplify this expression by making a number of approximations, namely $(n-1)_{(2^d-1)} \approx n^{2^d-1}$, $(2^e-1)_{(2^d-1)} \approx 2^{e(2^d-1)}$ and $\prod_{0\le i<d} 2^e - 2^i \approx 2^{ed}$. These approximations are justified as long as $2^d \ll 2^e$ and $2^d \ll n$. This yields

$$\frac{n^{2^d-1}}{2^{\left(2^d-1-d\right)e} \prod_{0\le i<d} 2^d - 2^i} \,. \tag{2}$$

We now define $L_{\min}(p(x), d)$ as the minimum length of a sequence $\{\text{caracolle}(k, i) | 0 \le i < n\}$ containing an affine space of dimension d. As explained above, despite this expression, $L_{\min}(p(x), d)$ only depends on $p(x)$ and d and is hence independent of both k and i.

A set of n random vectors is likely to contain an affine space of dimension d if the expected number as expressed in Equation (2) equals 1. We can now estimate $L_{\min}(p(x), d)$ for some dimensions d and e. We have

$$L_{\min}(p(x), d) \approx 2^{\left(1-\frac{d}{2^d-1}\right)e} \left(\prod_{0\le i<d} \left(2^d - 2^i\right) \right)^{\frac{1}{2^d-1}} \tag{3}$$

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

$$\nu \approx \left(1 - \frac{d}{2^d-1}\right) e + \epsilon(d) \,,$$

with $\epsilon(d)$ the binary logarithm of the rightmost term of Equation (3). 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. However, we have conducted some experiments and it turns out that when we take for $p(x)$ polynomials that are not too sparse, Equation (3) does a very good job in predicting the values of $L_{\min}(p(x), d)$. This can be seen in Table 1 that lists the coefficients for computing Equation (3) and compares some predicted values with experimental ones.

d	2	3	4	5	6	7
$\left(1-\frac{d}{2^d-1}\right)$	$\frac{1}{3}$	$\frac{4}{7}$	$\frac{11}{15}$	$\frac{26}{31}$	$\frac{57}{63}$	$\frac{120}{127}$
$\epsilon(d)$	0.86	1.06	0.95	0.75	0.54	0.37
$e = 13$, estimated $L_{\min}(p(x), d)$	$2^{5.2}$	$2^{8.5}$	$2^{10.5}$	$2^{11.7}$	$2^{12.3}$	$2^{12.7}$
$e = 13$, measured $L_{\min}(p(x), d)$	$2^{5.5}$	$2^{8.5}$	$2^{10.5}$	$2^{11.6}$	-	-
$e = 17$, estimated $L_{\min}(p(x), d)$	$2^{6.5}$	$2^{10.8}$	$2^{13.4}$	2^{15}	$2^{15.9}$	$2^{16.4}$
$e = 17$, measured $L_{\min}(p(x), d)$	$2^{6.9}$	$2^{10.4}$	$2^{13.3}$	-	-	-
$e = 29$, estimated $L_{\min}(p(x), d)$	$2^{10.5}$	$2^{17.6}$	$2^{22.2}$	2^{25}	$2^{26.8}$	$2^{27.8}$
$e = 29$, measured $L_{\min}(p(x), d)$	$2^{10.2}$	$2^{17.4}$	-	-	-	-
$e = 61$, estimated $L_{\min}(p(x), d)$	$2^{21.2}$	2^{36}	$2^{45.7}$	2^{52}	$2^{55.7}$	2^{58}
$e = 61$, measured $L_{\min}(p(x), d)$	$> 2^{19.5}$	-	-	-	-	-

Table 1. Coefficients for estimating $L_{\min}(p(x), d)$, predicted and estimated values. The lower bound "$> 2^{19.5}$" is where the scanning program was at the time of writing.

6 KRAVATTE: Farfalle based on KECCAK-p

In this section we present a Farfalle instantiation based on KECCAK-p, the permutation underlying KECCAK, KEYAK and KETJE and standardized in FIPS 202 [7,9,10,29].

KRAVATTE is defined as Farfalle$[f, r, \text{roll}]$ with the following parameters:

- $f =$ KECCAK-$p[1600, n_\text{r} = 6]$,
- $r = 1536$ bits,
- roll $=$ caracolle$[p(x)]$ with $e = 61$,
- $p(x) = x^{61} + x^{60} + x^{57} + x^{56} + x^{53} + x^{52} + x^{48} + x^{45} + x^{44} + x^{41} + x^{40} + x^{39} + x^{36} + x^{35} + x^{34} + x^{33} + x^{31} + x^{30} + x^{29} + x^{28} + x^{27} + x^{25} + x^{23} + x^{20} + x^{18} + x^{17} + x^{16} + x^{13} + x^{12} + x^{11} + x^{8} + x^{7} + x^{2} + x + 1$,
- $n_\text{roll} = 2^{56}$.

We define KRA and VATTE as the compression and expansion layers of KRAVATTE respectively. Functions and schemes based on KRAVATTE follow the same naming conventions as for Farfalle, e.g., KRAVATTE-PRF, KRAVATTE-WBC, etc. The option Full/Aligned is determined by the mode and otherwise can be chosen by the user.

We aim for at least 128-bit security, with the following claim:

Claim 1 *KRA and KRAVATTE are jointly Farfalle-secure with claimed capacity $c_{claim} = 256$ bits.*

The choice of $r = 1536$ aims at maximizing the block size and at allowing the alignment of memory addresses on 2^9 bits, while leaving enough bits for the rolling function.

Since $e \leq b - r - 2 = 62$ bits, we have preferred to take $e = 61$ because $2^{61} - 1$, the order of the multiplicative group of GF(2^{61}) and hence the length of LFSR sequences of primitive polynomials of degree 61, is a prime. We feel this minimizes the risk of regular structures in the LFSR sequence.

With $e = 61$, this concretely means that the rolling function caracolle() works on bits $z = 3$ (coefficient of 1) to $z = 63$ (coefficient of x^{60}) in the last lane (at $x = y = 4$) of the KECCAK-p state. In particular, when computing caracolle$(k, 0)$, the bit at $z = 3$ is set to 1. The bits at $z = 0$ and $z = 1$ in that lane get the padding bits, and the bit at $z = 2$ is left alone.

We have chosen the particular polynomial $p(x)$ as follows. We started with the irreducible binary pentanomial of degree 61 with the smallest degree terms: $q(x) = x^{61} + x^5 + x^2 + x + 1$ (there are no irreducible binary trinomials of degree 61). Then, for α a root of $q(x)$, we took for $p(x)$ the minimal polynomial of α^{61}.

By taking $n_\text{roll} = 2^{56}$, Table 1 suggests that the largest affine subspaces that can occur in the sequence are likely to have dimension 6. The value n_roll limits the size of the input in a call to KRAVATTE to 2^{56} blocks, 3×2^{65} bits or 12 exbibytes (=12×1024^6 bytes). We are not aware of a practical use case where this would be limiting. Note that this limit applies only to the input of a single call. The global limit on data complexity is much higher, bound by the security claim and captured by the value of M in Definition 1. With $c_\text{claim} = 256$ bits, the data complexity can go as high as 2^{128} blocks.

The choice of the number of rounds in the permutation $f =$ KECCAK-$p[1600, n_\text{r} = 6]$ is driven by the rationale of Section 4.1. Between any input and any output of KRAVATTE, there are always $2n_\text{r} = 12$ rounds, which should be enough to resist against known attacks and in particular against cube attacks. Regarding the exploitation of differential properties, note that we showed lower bounds on the weight of any trail in KECCAK-$p[1600, n_\text{r} = 6]$ [15]. With the unknown difference induced by the rolling function in the lane at $x = y = 4$, we expect its propagation to be very fast since it is outside of the so-called column parity kernel [7].

We do not have performance figures available yet. One can nevertheless look at the performance of KangarooTwelve for the cost of 12 rounds when exploiting parallelism on modern processors [11], and to extrapolate it to 6 rounds. Of course, this gives a crude estimation, as on top of the time needed to compute the 6 rounds of Kra (per input block) and the 6 rounds of Vatte (per output block), there is also the time needed to evaluate the rolling function. Also, the block size is different and at the advantage of Kravatte, i.e., $r = 1536$ vs $r = 1344$ for KangarooTwelve.

7 Conclusions

Farfalle is a versatile new construction for keyed functions in symmetric cryptography based on a permutation. It can be seen as an inherently parallelizable counterpart of sponge/duplex. 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 full-state keyed duplex for keyed applications in embedded platforms where dedicated hardware can be afforded or where DPA/DEMA-type side-channel attacks are a threat.

Acknowledgements: 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

1. E. Andreeva, J. Daemen, B. Mennink, and G. Van Assche, *Security of keyed sponge constructions using a modular proof approach*, Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers (Gregor Leander, ed.), Lecture Notes in Computer Science, vol. 9054, Springer, 2015, pp. 364–384.
2. D. Bernstein, *Some challenges in heavyweight cipher design*, 2016, Presented at Dagstuhl seminar on Symmetric Cryptography. Schloss Dagstuhl.
3. D. J. Bernstein, *The Salsa20 family of stream ciphers*, 2007, Document ID: 31364286077dcdff8e4509f9ff3139ad, `http://cr.yp.to/papers.html#salsafamily`.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *On the indifferentiability of the sponge construction*, Advances in Cryptology – Eurocrypt 2008 (N. P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, `http://sponge.noekeon.org/`, pp. 181–197.
5. _____, *Cryptographic sponge functions*, January 2011, `http://sponge.noekeon.org/`.
6. _____, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Selected Areas in Cryptography (SAC), 2011.
7. _____, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
8. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, KeccakTools *software*, September 2015, `https://github.com/gvanas/KeccakTools`.
9. _____, *CAESAR submission:* Ketje *v2*, September 2016, `http://ketje.noekeon.org/`.
10. _____, *CAESAR submission:* Keyak *v2, document version 2.2*, September 2016, `http://keyak.noekeon.org/`.
11. _____, KangarooTwelve*: fast hashing based on* Keccak*-p*, Cryptology ePrint Archive, Report 2016/770, 2016, `http://eprint.iacr.org/2016/770`.
12. _____, Keccak *code package*, June 2016, `https://github.com/gvanas/KeccakCodePackage`.
13. J. Black and P. Rogaway, *A block-cipher mode of operation for parallelizable message authentication*, Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings (L. R. Knudsen, ed.), Lecture Notes in Computer Science, vol. 2332, Springer, 2002, pp. 384–397.
14. C. Bouillaguet, *Etudes d'hypothèses algorithmiques et attaques de primitives cryptographiques, thèse de doctorat*, Université Paris Diderot, 2011.
15. J. Daemen and G. Van Assche, *Differential propagation analysis of* Keccak, Fast Software Encryption 2012, 2012.
16. J. Daemen and V. Rijmen, *A new MAC construction ALRED and a specific instance ALPHA-MAC*, Fast Software Encryption (H. Gilbert and H. Handschuh, eds.), Lecture Notes in Computer Science, vol. 3557, Springer, 2005, pp. 1–17.

17. _____, *The Pelican MAC function*, IACR Cryptology ePrint Archive **2005** (2005), 8.
18. _____, *Refinements of the ALRED construction and MAC security claims*, IET information security **4** (2010), 149–157.
19. S. Even and Y. Mansour, *A construction of a cipher from a single pseudorandom permutation*, J. Cryptology **10** (1997), no. 3, 151–162.
20. Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway, *Robust authenticated-encryption AEZ and the problem that it solves*, Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I (Elisabeth Oswald and Marc Fischlin, eds.), Lecture Notes in Computer Science, vol. 9056, Springer, 2015, pp. 15–44.
21. Marcos A. Simplício Jr. and Paulo S. L. M. Barreto, *Revisiting the security of the ALRED design and two of its variants: Marvin and lettersoup*, IEEE Trans. Information Theory **58** (2012), no. 9, 6223–6238.
22. Marcos A. Simplício Jr., Pedro d'Aquino F. F. S. Barbuda, Paulo S. L. M. Barreto, Tereza Cristina M. B. Carvalho, and Cintia B. Margi, *The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme*, Security and Communication Networks **2** (2009), no. 2, 165–180.
23. H. Krawczyk, *LFSR-based hashing and authentication*, Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings (Y. Desmedt, ed.), Lecture Notes in Computer Science, vol. 839, Springer, 1994, pp. 129–139.
24. Ted Krovetz, *HS1-SIV (v2)*, 2015, Submission to CAESAR competition.
25. Michael Luby and Charles Rackoff, *How to construct pseudorandom permutations from pseudorandom functions*, SIAM J. Comput. **17** (1988), no. 2, 373–386.
26. Stefan Lucks, *Faster Luby-Rackoff ciphers*, Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings (Dieter Gollmann, ed.), Lecture Notes in Computer Science, vol. 1039, Springer, 1996, pp. 189–203.
27. B. Mennink, R. Reyhanitabar, and D. Vizár, *Security of full-state keyed sponge and duplex: Applications to authenticated encryption*, Advances in Cryptology - ASIACRYPT 2015, New Zealand, 2015 (T. Iwata and J. H. Cheon, eds.), LNCS, vol. 9453, Springer, 2015, pp. 465–489.
28. M. Naor and O. Reingold, *On the construction of pseudorandom permutations: Luby-Rackoff revisited*, J. Cryptology **12** (1999), no. 1, 29–66.
29. NIST, *Federal information processing standard 202, SHA-3 standard: Permutation-based hash and extendable-output functions*, August 2015, `http://dx.doi.org/10.6028/NIST.FIPS.202`.
30. Phillip Rogaway and Thomas Shrimpton, *Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem*, IACR Cryptology ePrint Archive **2006** (2006), 221.