From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications

Lorenzo Grassi[1], Morten Øygarden[2], Markus Schofnegger[3], and Roman Walch[4,5,6]

[1] Radboud University Nijmegen (Nijmegen, The Netherlands)
[2] Simula UiB (Bergen, Norway)
[3] Horizen Labs (Austin, United States)
[4] Graz University of Technology (Graz, Austria)
[5] Know-Center GmbH (Graz, Austria)
[6] TACEO GmbH (Graz, Austria)
`lgrassi@science.ru.nl`, `morten.oygarden@simula.no`, `mschofnegger@horizenlabs.io`, `roman.walch@iaik.tugraz.at`

Abstract. The area of multi-party computation (MPC) has recently increased in popularity and number of use cases. At the current state of the art, Ciminion, a Farfalle-like cryptographic function, achieves the best performance in MPC applications involving symmetric primitives. However, it has a critical weakness. Its security highly relies on the independence of its subkeys, which is achieved by using an expensive key schedule. Many MPC use cases involving symmetric pseudo-random functions (PRFs) rely on secretly shared symmetric keys, and hence the expensive key schedule must also be computed in MPC. As a result, Ciminion's performance is significantly reduced in these use cases.

In this paper we solve this problem. Following the approach introduced by Ciminion's designers, we present a novel primitive in symmetric cryptography called Megafono. Megafono is a keyed extendable PRF, expanding a fixed-length input to an arbitrary-length output. Similar to Farfalle, an initial keyed permutation is applied to the input, followed by an expansion layer, involving the parallel application of keyed ciphers. The main novelty regards the expansion of the intermediate/internal state for "free", by appending the sum of the internal states of the first permutation to its output. The combination of this and other modifications, together with the impossibility for the attacker to have access to the input state of the expansion layer, make Megafono very efficient in the target application.

As a concrete example, we present the PRF Hydra, an instance of Megafono based on the Hades strategy and on generalized versions of the Lai–Massey scheme. Based on an extensive security analysis, we implement Hydra in an MPC framework. The results show that it outperforms all MPC-friendly schemes currently published in the literature.

Keywords: Megafono – Hydra – Farfalle – Ciminion – MPC Applications

1 Introduction

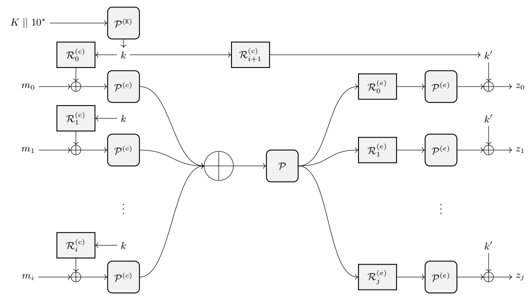
Secure multi-party computation (MPC) allows several parties to jointly and securely compute a function on their combined private inputs. The correct output is computed and given to all parties (or a subset) while hiding the private inputs from other parties. In this work we focus on secret-sharing based MPC schemes, such as the popular SPDZ protocol [24,23] or protocols based on Shamir's secret sharing [47]. In these protocols private data is shared among all parties, such that each party receives a share which by itself does not contain any information about the initial data. When combined, however, the parties are able to reproduce the shared value. Further, the parties can use these shares to compute complex functions on the data which in turn produce shares of the output.

MPC has been applied to many use cases, including privacy-preserving machine learning [46], private set intersection [40], truthful auctions [13], and revocation in credential systems [38]. In the literature describing these use cases, data is often directly entered from and delivered to the respective parties. However, in practice this data often has to be transferred securely from/to third parties before it can be used in the MPC protocol. Moreover, in some applications, intermediate results of an MPC computation may need to be stored securely in a database. As described in [35], one can use MPC-friendly pseudo-random functions (PRFs), i.e., PRFs designed to be efficient in MPC, to efficiently realize this secure data storage and data transfer by directly encrypting the data using a secret-shared symmetric key.

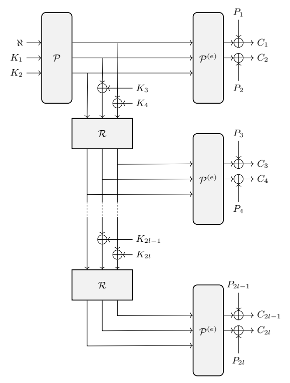
Besides being used to securely transmit data in given MPC computations, these MPC-friendly PRFs can also be used as a building block to speed up many MPC applications, such as secure database join via an MPC evaluation of a PRF [44], distributed data storage [35], virtual hardware security modules[7], MPC-in-the-head based zero-knowledge proofs [39] and signatures [16], oblivious TLS [1], and many more. *In all these use cases, the symmetric encryption key is shared among all participating parties. Consequently, if one has to apply a key schedule for a given PRF, one has to compute this key schedule at least once in MPC for every fresh symmetric key.*

To be MPC-friendly, a PRF should minimize the number of multiplications in the native field of the MPC protocol. At the current state of the art, Ciminion [26] is one of the most competitive schemes for PRF applications. Proposed at Eurocrypt'21, it is based on the Farfalle mode of operation [10]. However, as we are going to discuss in detail, Ciminion has a serious drawback: *Its security heavily relies on the assumption that the subkeys are independent.* For this requirement, the subkeys are generated via a sponge hash function [11] instantiated via an expensive permutation. As a result, in all (common) cases where the key is shared among the parties, the key schedule cannot be computed locally and needs to be evaluated in MPC. This leads to a significant increase in the multiplicative complexity of Ciminion. In this paper, we approach this problem in two steps. First, we propose Megafono, a new mode of operation inspired by

[7] `https://www.fintechfutures.com/files/2020/09/vHSM-Whitepaper-v3.pdf`



(a) The Farfalle construction.



(b) The Ciminion PRF.

Fig. 1: Farfalle and Ciminion (notation adapted to the one used in this paper).

Farfalle and Ciminion.[8] It is designed to be competitive in all MPC applications. Secondly, we show how to instantiate it in an efficient way. The obtained PRF Hydra is currently the most competitive MPC-friendly PRF in the literature.

1.1 Related Works: Ciminion and the MPC Protocols

Traditional PRFs (e.g., AES, Keccak) are not efficient in MPC settings. First, MPC applications usually work over a prime field $\mathbb{F}_p$ for a large p (e.g., $p \approx 2^{128}$), while traditional cryptographic schemes are usually bit-/byte-oriented. Hence, a conversion between $\mathbb{F}_{2^n}$ and $\mathbb{F}_p$ must take place, which can impact the performance. Secondly, traditional schemes are designed to minimize their plain implementation cost, and therefore no particular focus is laid on minimizing specifically the number of nonlinear operations (e.g., `AND` gates).

For these reasons, several MPC-friendly schemes over $\mathbb{F}_q^t$ for $q = p^s$ and $t \geq 1$ have been proposed in the literature, including LowMC [4], MiMC [3], GMiMC [2], HadesMiMC [32], and *Rescue* [5]. All those schemes are block ciphers – hence, invertible – and they are often used in counter (CTR) mode. However, invertibility is not required in MPC applications, and a lower multiplicative complexity may be achieved by working with non-invertible functions, as recently shown by in [26]. In the following, we briefly discuss the Farfalle construction and the MPC-friendly primitive Ciminion based on it.

Farfalle. Farfalle [10] is an efficiently parallelizable permutation-based construction of arbitrary input and output length, taking as input a key. As shown in Fig. 1a and recalled in Section 3, the Farfalle construction consists of a *compression layer* followed by an *expansion layer*. The compression layer produces

[8] "Megafono" is the Italian word for "megaphone", a cone-shaped horn used to amplify a sound and direct it towards a given direction. Our strategy resembles this goal.

a single accumulator value from the input data. A permutation is potentially applied to the obtained state. Then, the expansion layer transforms it into a tuple of (truncated) output blocks. Both the compression and expansion layers involve the secret key, and they are instantiated via a set of permutations (namely, $\mathcal{P}^{(c)}, \mathcal{P}, \mathcal{P}^{(e)}$) and rolling functions ($\mathcal{R}_i^{(c)}, \mathcal{R}_i^{(e)}$).

Ciminion. As shown with Ciminion in [26], a modified version of Farfalle based on a Feistel scheme can be competitive for MPC protocols, an application which Farfalle's designers did not consider. Following Fig. 1b and Section 3,

(1) compared to Farfalle, the compression phase is missing, a final truncation is applied, and the key addition is performed before $\mathcal{P}^{(e)}$ is applied, and
(2) in contrast to MPC-friendly block ciphers, Ciminion is a *non-invertible* PRF. For encryption it is used as a stream cipher, where the input is defined as the concatenation of the secret key and a nonce.

The main reason why Ciminion is currently the most competitive scheme in MPC protocols is related to one crucial feature of Farfalle, namely the possibility to instantiate its internal permutations with a smaller number of rounds compared to other design strategies. This is possible since the attacker does not have access to the internal states of the Farfalle construction. Hence, while the permutation $\mathcal{P}^{(c)}$ is designed in order to behave like a pseudo-random permutation (PRP), the number of rounds of the permutation $\mathcal{P}^{(e)}$ can be minimized and kept significantly lower for both security and good performance.

Besides minimizing the number of nonlinear operations, Ciminion's designers paid particular attention to the number of linear operations. Indeed, even though the main cost in MPC applications depends on the number of multiplications, other factors (e.g., the number of linear operations) affect efficiency as well.

1.2 The Megafono Design Strategy

The main drawback of Ciminion is the expensive key schedule to generate subkeys that can be considered independent. This implies that Ciminion only excels in MPC applications where the key schedule can be precomputed for a given shared key, or in the (non-common) scenarios where the key is not shared among the parties. However, in the latter case, the party knowing the key can also compute Ciminion's keystream directly in plain (i.e., without MPC) if the nonce and `IV` are public in a given use case (which is also true for any stream cipher).

Clearly, the easiest solution is the removal of the nonlinear key schedule. However, by e.g. defining the subkey as an affine function of the master key, the security analysis of Ciminion does not hold anymore. As we discuss in detail in Section 4, this is a direct consequence of the Farfalle construction itself. Even if the attacker does not have any information about the internal states of Farfalle, they can exploit the fact that its outputs are generated from the **same** unknown input (namely, the output of $\mathcal{P}^{(c)}$ and/or $\mathcal{P}$). Given these outputs and by exploiting the relations of the corresponding unknown inputs (which are related to the definition of the rolling function), the attacker can potentially find the key and break the scheme. For example, this strategy is exploited in the attacks on the Farfalle schemes Kravatte and Xoofff [15,19]. In Ciminion, this problem is solved by including additions with independent secret subkeys in the application of the rolling function. In this way, the mentioned relation is unknown due to the presence of the key, and $\mathcal{P}^{(e)}$ can be instantiated via an efficient permutation.

We make the following three crucial changes in the Farfalle design strategy.

1. First, we replace the permutation $\mathcal{P}^{(e)}$ with a keyed permutation $\mathcal{C}_k$.
2. Secondly, we expand the input of this keyed permutation. The second change aims to frustrate algebraic attacks, whose cost is related to both the degree and the number of variables of the nonlinear equation system representing the attacked scheme. In order to create new independent variables for "free" (i.e., without increasing the overall multiplicative complexity), we reuse the computations needed to evaluate $\mathcal{P}$. That is, we define the new variable as the sum of all the internal state of $\mathcal{P}$, and we conjecture that it is sufficiently independent of its output (details are provided in the following).
3. Finally, we replace the truncation in Ciminion with a feed-forward operation, for avoiding to discard any randomness without any impact on the security.

Our result is a new design strategy which we call Megafono.

1.3 The PRF Hydra

Given the mode of operation, we instantiate it with two distinct permutations, one for the initial phase and one for the expansion phase. As in Ciminion, assuming the first keyed permutation behaves like a PRP and since the attacker does not know the internal states of Megafono, we choose a second permutation that is cheaper to evaluate in the MPC setting. In particular, while the first permutation is evaluated only once, the number of calls to the second permutation (and so the overall cost) is proportional to the output size.

For minimizing the multiplicative complexity, we instantiate the round functions of the keyed permutations $\mathcal{C}_k$ in the expansion part with quadratic functions. However, since no quadratic function is invertible over $\mathbb{F}_p$, we use them in a mode of operation that guarantees invertibility. We opted for the generalized Lai–Massey constructions similar to the ones recently proposed in [33]. Moreover, we show that the approach of using of high-degree power maps with low-degree inverses proposed in *Rescue* does not have any benefits in this scenario.

We instantiate the first permutation $\mathcal{P}$ via the Hades strategy [32], which mixes rounds with full S-box layers and rounds with partial S-box layers. Similar to Neptune [33], we use two different round functions, one for the internal part and one for the external one. We decided to instantiate the internal rounds with a Lai–Massey scheme, and the external ones with invertible power maps.

The obtained PRF scheme called Hydra is presented in Section 5 and Section 6, and its security analysis is proposed in Section 7.

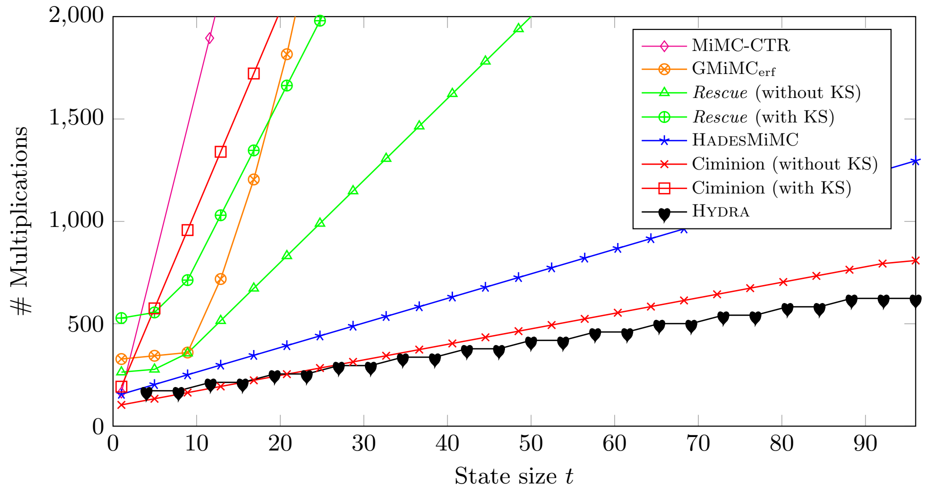


Fig. 2: Number of MPC multiplications of several designs over $\mathbb{F}_p^t$, with $p \approx 2^{128}$ and $t \geq 2$ (security level of 128 bits).

1.4 MPC Performance and Comparison

The performance of any MPC calculation scales with the number of nonlinear operations. In Figure 2 we compare the number of multiplications required to evaluate different PRFs for various plaintext sizes t using secret shared keys. One can observe that Hydra requires the smallest number of multiplications, with the difference growing further for larger sizes. The only PRF that is competitive to Hydra is Ciminion, but only *if the key schedule does not have to be computed*, which happens if shared round keys can be reused from a previous computation. However, this implies that the key schedule was already computed once in MPC, requiring a significant amount of multiplications. Hydra, on the other hand, does not require the computation of an expensive key schedule and also requires fewer multiplications than Ciminion without a key schedule for larger state sizes.

In Section 8, we implement and compare the different PRFs in the MP-SPDZ [41] library and confirm the results expected from Figure 2. Indeed,

(1) taking key schedules into account, Hydra is five times faster than Ciminion for $t = 8$, which grows to a factor of 21 for $t = 128$,
(2) without key schedules, Ciminion is only slightly faster than Hydra for smaller t, until it gets surpassed by Hydra for $t > 16$, showing that Hydra is also competitive, even if the round keys are already present.

Compared to all other benchmarked PRFs, Hydra is significantly faster for any state size t. Furthermore, Hydra requires the least amount of communication between the parties due to its small number of multiplications, giving it an advantage in low-bandwidth networks. As a result, we suggest to replace each of the benchmarked PRFs with Hydra in all their use cases, especially if a large number of words need to be encrypted.

1.5 Notation

Throughout the paper, we work over a finite field $\mathbb{F}_q$, where $q = p^s$ for an odd prime number p and an integer $s \geq 1$ (when needed, we will also assume a fixed vector space isomorphism $\mathbb{F}_{p^s} \cong \mathbb{F}_p^s$). We use $\mathbb{F}_q^n$, for $n \geq 1$, to denote the n-dimensional vector space over $\mathbb{F}_q$, and we use the notation $\mathbb{F}_q^\star$ to denote $\mathbb{F}_q$ strings of arbitrary length. The $\cdot \mid\mid \cdot$ operator denotes the concatenation of two elements. An element $x \in \mathbb{F}_q^t$ is represented as $x = (x_0, x_1, \ldots, x_{t-1})$, where x_i denotes its i-th entry. Given a matrix $M \in \mathbb{F}_q^{t \times s}$, we denote its entry in row r and column c either as $M_{r,c}$ or $M[r,c]$. We use the Fraktur font notation to denote a subspace of $\mathbb{F}_q^r$, while we sometimes use the calligraphic notation to emphasize functions. Given integers $a \geq b \geq 1$, we define the truncation function $\mathcal{T}_{a,b} : \mathbb{F}_q^a \to \mathbb{F}_q^b$ as $\mathcal{T}_{a,b}(x_0, \ldots, x_{a-1}) = (x_0, \ldots, x_{b-1})$. Finally, for MPC, we describe that the value $x \in \mathbb{F}_p$ is secret shared among all parties by $[x]$.

2 Symmetric Primitives for MPC Applications

Here we elaborate on why expensive key schedules are not desirable in many MPC use cases, and we discuss the cost metric in MPC protocols in more details.

2.1 MPC Use Cases and Key Schedules

To highlight that expensive key schedules are not suitable for many scenarios, we describe the use cases discussed in [26] and [35] in greater detail. Concretely, we discuss the data transfer into and out of the MPC protocol, as well as using symmetric PRFs to securely store intermediate results during an MPC evaluation. In the latter case, the setting is the following: The parties want to suspend the MPC evaluation and continue at a later point. As discussed in [35], the trivial solution for this problem is that each party encrypts its share of the data with a symmetric key and stores the encrypted share, e.g., at a cloud provider. The total storage overhead of this approach is a factor n for n MPC parties, since each party stores its encrypted shares of the data. Additionally, each party needs to memorize its symmetric key. The solution to reduce the storage overhead is to use a secret shared symmetric key (i.e., each party knows only a share of the key and the real symmetric key remains hidden), which can directly be sampled as part of the MPC protocol, and encrypt the data using MPC. The resulting ciphertexts cannot be decrypted by any party since no one knows the symmetric key, but can be used inside the MPC protocols at a later point to again create the shares of the data. This approach avoids the storage overhead of the data, and each party only has to memorize its share of the symmetric key which has the same size as the symmetric key itself. However, if the used PRF involves a key schedule, one also has to compute it in MPC for this use case. Other solutions either involve precomputing the round keys, or directly sampling random round keys in MPC instead of sampling a random symmetric key. These approaches require no storage overhead for the encrypted data, but each party needs to memorize its shares of the round keys. In Ciminion, the size of the round keys is equivalent to the size of the encrypted data (when using the same nonce for encrypting the full dataset), hence the whole protocol would be more efficient if each party would just memorize its shares of the dataset instead. Providing fewer round keys and using multiple nonces instead requires the recomputation of Ciminion's initial permutation in MPC, decreasing its performance.

Similar considerations also apply if the MPC parties are different from the actual data providers or if the output of the computation needs to be securely transferred to an external party. The solutions to both problems involve storing the dataset encrypted at some public place (e.g., in a cloud) alongside a public-key encryption of the shares of the symmetric key, such that only the intended recipient can get the shares. If the parties want to avoid expensive key schedules in MPC, they either have to provide shares of the round keys (which have the same size as the encrypted data in Ciminion), or provide fewer round keys alongside multiple nonces, decreasing the performance in MPC.

Remark 1. In this paper, we focus on comparing MPC-friendly PRFs which are optimized for similar use cases as the ones discussed in this section, i.e., use cases which require fast MPC en-/decryption of large amounts of data. Hence, we do not focus on PRFs not defined over $\mathbb{F}_p$ which are optimized for, e.g., Picnic-style signatures, such as LowMC [4], Rain [27], or weakPRF [25].

2.2 Cost Metric for MPC Applications

Modern MPC protocols such as SPDZ [24,23] are usually split into a data-independent *offline phase* and a data-dependent *online phase*. In the offline phase, a bundle of shared correlated randomness is generated, most notably Beaver triples [9] of the form $([a], [b], [a \cdot b])$. This bundle is then used in the online phase to perform the actual computation on the private data.

Roughly speaking, the performance scales with the number of nonlinear operations necessary to evaluate the symmetric primitives in the MPC protocol (sometimes we use the term "multiplication" to refer to the nonlinear operation). This is motivated by the fact that each multiplication requires one Beaver triple, which is computed in the offline phase, as well as one round of communication during the online phase (see [34, Appendix D]. In contrast, linear operations do not require any offline computations and can directly be applied to the shares without communication. Consequently, the number of multiplications is a decent first estimation of the cost metric in MPC, and MPC-friendly PRFs usually try to minimize this number. Whereas each multiplication requires communication between the parties, the depth directly defines the required number of communication rounds, since parallel multiplications can be processed in the same round. Thus, the depth should be low in high-latency networks. To summarize,

– the cost of the offline phase of the MPC protocols directly scales with the number of required Beaver triples (i.e., multiplications), and
– the cost of the online phase scales with both the number of multiplications and the multiplicative depth.

As a concrete example, in many MPC-friendly PRFs, such as HadesMiMC, MiMC, GMiMC, and *Rescue*, the nonlinear layer is instantiated with a power map $R(x) = x^d$ for $d \geq 2$ over $\mathbb{F}_q$. Then, the cost per evaluation is

$$\#\texttt{triples} = \texttt{cost}_d := \mathrm{hw}(d) + \lfloor \log_2(d) \rfloor - 1\,, \qquad \texttt{depth}_{\text{online}} = \texttt{cost}_d\,. \quad (1)$$

Several algorithms to reduce the number of multiplications and communication rounds were developed in the literature. Here we discuss those relevant for our goals. They require random pairs $[r], [r^2]$, and $[r], [r^{-1}]$, which can be generated from Beaver triples in the offline phase (see [34, Appendix D]).

Decreasing the Number of Online Communication Rounds. In the preferred case of $d = 3$, the cost is two Beaver triples and a depth of two. However, in [35] the authors propose a method to reduce the multiplicative depth by delegating the cubing operation to a random value in the offline phase. Hence, all cubings can be performed in parallel reducing the depth. This algorithm (see [34, Appendix D]) requires two triples, but only one online communication round.

Special Case: $R(x) = x^{1/d}$. Optimizations can also be applied for the case $R(x) = x^d$ with very large d. In [5], the authors propose two different algorithms to evaluate R (see [34, Appendix D]), in which the cost of evaluating $R(x) = x^d$ can be reduced to the cost of evaluating $R(x) = x^{1/d}$ (plus an additional multiplication in the online phase) which requires significantly fewer multiplications if $1/d$ is smaller than d. This is, for example, relevant when evaluating *Rescue* with its high-degree power maps in MPC. The algorithm works by delegating the $1/d$ power map evaluation to the offline phase, and evaluating the costly d power map on a random value in plain. Furthermore, since the main MPC work (i.e., $1/d$) is evaluated in the input-independent offline phase, all communication rounds can be parallelized, significantly reducing the multiplicative depth. Using these algorithms and $\texttt{cost}_d$ from Eq. (1), the cost of evaluating x^d in MPC is modified to the following, with a significantly smaller multiplicative depth and a smaller number of multiplications for large d:

$$\#\texttt{triples} = 2 + \min\left\{\texttt{cost}_d, \texttt{cost}_{1/d}\right\}\,, \qquad \texttt{depth}_{\text{online}} = 2.$$

3 Starting Points of Megafono: Farfalle and Ciminion

Here we recall Farfalle and Ciminion, which are starting points for Megafono.

Farfalle and $^{1/2\times}$Farfalle. Farfalle is a keyed PRF proposed in [10] with inputs and outputs of arbitrary length. As shown in Fig. 1a, it has a compression layer and an expansion layer, each involving the parallel application of a permutation. For our goal, we focus only on the expansion phase, and introduce the term $^{1/2\times}$Farfalle for a modified version of Farfalle that lacks the initial compression phase and only accepts input messages of a fixed size n.

Let $\mathtt{K} \in \mathbb{F}_q^\kappa$ be the secret key for $\kappa \geq 1$. $^{1/2\times}$Farfalle uses a key schedule $\mathcal{K} : \mathbb{F}_q^\kappa \to (\mathbb{F}_q^n)^\star$ for the subkeys used in the expansion phase, two unkeyed permutations $\mathcal{P}, \mathcal{P}^{(e)} : \mathbb{F}_q^n \to \mathbb{F}_q^n$, and a rolling function $\mathcal{R} : \mathbb{F}_q^n \to \mathbb{F}_q^n$.[9] We define $\mathcal{R}_i$ as $\mathcal{R}_i(y) = \rho_i + \mathcal{R} \circ \mathcal{R}_{i-1}(y)$ for each $i \geq 1$ and $\rho_i \in \mathbb{F}_q^n$, where we assume $\mathcal{R}_0$ to be the identity function, i.e., $\mathcal{R}_0(y) = y$. Given an input $x \in \mathbb{F}_q$, $^{1/2\times}$Farfalle : $\mathbb{F}_q^n \to (\mathbb{F}_q^n)^\star$ operates as $^{1/2\times}$Farfalle$(x) = y_0 \,||\, y_1 \,||\, y_2 \,||\, \cdots \,||\, y_j \,||\, \cdots$, where $\forall i \geq 0 : y_i := k_{i+1} + \mathcal{P}^{(e)}\left(\mathcal{R}_i\left(\mathcal{P}(x + k_0)\right)\right)$..

From $^{1/2\times}$Farfalle to Ciminion. Ciminion [26] is based on a modified version of $^{1/2\times}$Farfalle over $\mathbb{F}_q^n$ for a certain $n \geq 2$. As shown in Fig. 1b, the main difference with respect to $^{1/2\times}$Farfalle is the definition of the function $k + \mathcal{P}^{(e)}$. In Farfalle/$^{1/2\times}$Farfalle, the key addition is the last operation. In Ciminion, $k + \mathcal{P}^{(e)}(x)$ is replaced by $\mathcal{F}^{(e)}(x+k)$ for a *non-invertible* function $\mathcal{F}^{(e)}$ instantiated via a truncated permutation, i.e., $\mathcal{F}^{(e)}(x+k) \coloneqq \mathcal{T}_{n,n'} \circ \mathcal{P}^{(e)}(x+k)$ for a certain $1 \leq n' < n$. Moving the key inside the scheme prevents its cancellation when using the difference of two outputs.

In Ciminion, the key schedule $\mathcal{K} : \mathbb{F}_q^\kappa \to (\mathbb{F}_q^n)^\star$ uses a sponge function [11] instantiated via the permutation $\mathcal{P}$. We refer to [26, Section 2] for more details.

4 The Megafono Strategy for Hydra

Generating the subkeys of Ciminion via a sponge function and a strong permutation is expensive in terms of multiplications. This makes it inefficient in cases where the secret keys are shared among the parties, as discussed in Section 2.1. Another weakness of Ciminion is the final truncation. While it prevents an attacker from computing the inverse of the final permutations $\mathcal{P}^{(e)}$, it is wasteful as it lowers the output of each iteration. To fix these issues, here we propose the Megafono strategy, based on the design strategy of Ciminion (and $^{1/2\times}$Farfalle), but with some crucial modifications.

Definition of Megafono. Let $n \geq 1$ be an integer and let $\mathbb{F}_q$ be a field, where $q = p^s$ for a prime integer $p \geq 2$ and a positive integer $s \geq 1$. Let $\mathtt{K} \in \mathbb{F}_q^\kappa$ be the secret key for $n \geq \kappa \geq 1$. The ingredients of Megafono are

[9] We mention that in [10], authors use the terms "masks" and "(compressing) rolling function" instead of "subkeys" and "key schedule". In Farfalle, the same subkey is used in the expansion phase, that is, $k_1 = k_2 = \cdots = k_i$. Here, we consider the most generic case in which the subkeys are not assumed to be equal.

(1) a key schedule $\mathcal{K} : \mathbb{F}_q^{\kappa} \to (\mathbb{F}_q^n)^{\star}$ for generating the subkeys, that is, $\mathcal{K}(\mathtt{K}) = (k_0, k_1, \ldots, k_i, \ldots)$ where $k_i \in \mathbb{F}_q^n$ for each $i \geq 0$,

(2) an iterated unkeyed permutation $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ defined as

$$\mathcal{P}(x) = \mathcal{P}_{r-1} \circ \ldots \circ \mathcal{P}_1 \circ \mathcal{P}_0(x) \tag{2}$$

for round permutations $\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_{r-1}$ over $\mathbb{F}_q^n$,

(3) a (sum) function $\mathcal{S} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ defined as

$$\mathcal{S}(x) := \sum_{i=0}^{r-1} \mathcal{P}_i \circ \ldots \circ \mathcal{P}_1 \circ \mathcal{P}_0(x), \tag{3}$$

(4) a function $\mathcal{F}_k : \mathbb{F}_q^{2n} \to \mathbb{F}_q^{2n}$ defined as

$$\mathcal{F}_k(x) := \mathcal{C}_k(x) + x,$$

where $\mathcal{C}_k : \mathbb{F}_q^{2n} \to \mathbb{F}_q^{2n}$ is a block cipher for a secret key $k \in \mathbb{F}_q^{2n}$, and

(5) a rolling function $\mathcal{R} : \mathbb{F}_q^{2n} \to \mathbb{F}_q^{2n}$. For $y, z \in \mathbb{F}^n$, we further define

$$\mathcal{R}_i(y, z) := \varphi_i + \mathcal{R} \circ \mathcal{R}_{i-1}(y, z)$$

for $i \geq 1$, where $\varphi_i \in \mathbb{F}_q^{2n}$ and $\mathcal{R}_0(y, z) = (y, z)$.

$\textsc{Megafono}_{\mathtt{K}} : \mathbb{F}_q^n \to (\mathbb{F}_q^n)^{\star}$ is a PRF that takes as input an element of $\mathbb{F}_q^n$ and returns an output of a desired length, defined as

$$\textsc{Megafono}_{\mathtt{K}}(x) := \mathcal{F}_{k_2}(y, z) \mid\mid \mathcal{F}_{k_3}(\mathcal{R}_1(y, z)) \mid\mid \cdots \mid\mid \mathcal{F}_{k_{i+2}}(\mathcal{R}_i(y, z)) \mid\mid \cdots$$

for $i \in \mathbb{N}$, where $y, z \in \mathbb{F}_q^n$ are defined as

$$y := k_1 + \mathcal{P}(x + k_0) \qquad \text{and} \qquad z := \mathcal{S}(x + k_0)\,.$$

Remark 2. The main goal of Megafono is a secure variant of Ciminion without a heavy key schedule and without relying on independent subkeys $(k_0, k_1, \ldots)$. For this reason, we only consider the case $k = n$ and $\mathcal{K}(\mathtt{K}) = (\mathtt{K}, \ldots, \mathtt{K}, \ldots)$ in the following. Nevertheless, there may be applications in which a key schedule is acceptable, and hence we propose Megafono in its more general form.

Remark 3. The function $\mathcal{F}_k$ is meant to play the role of $\mathcal{P}^{(e)}$ (in the notation we have used to describe Farfalle and Ciminion). We use this notation to emphasize that the function is keyed and that we no longer require it to be a permutation.

4.1 Rationale of Megafono

Following its structure, Megafono shares several characteristics with Ciminion and $^{1/2\times}$Farfalle. Indeed, many attacks on Farfalle (and Ciminion, $^{1/2\times}$Farfalle) discussed in [10, Sect. 5] also apply to Megafono. Here we focus on the differences, by explaining and motivating the criteria for designing Megafono.

Expansion Phase. We emphasize the following point which is crucial for understanding the design rationale of MEGAFONO. As in $^{1/2\times}$Farfalle and Ciminion, the attacker has access to outputs $w_i = \mathcal{F}_k\left(\mathcal{R}_i(y,z)\right)$ for $i \geq 0$ that depend on a *single common* unknown input (y,z) (in addition to the key). By exploiting the relation among several inputs of $\mathcal{F}_k$ and the knowledge of the corresponding outputs, the attacker can break the entire scheme. Examples of such attacks can be found in [15,19]. In this scenario, one attack consists of solving the system of equations $\{w_i - \mathcal{F}_k\left(\mathcal{R}_i(y,z)\right) = 0\}_{i\geq 0}$ with Gröbner bases. We provide details in Section 7.4 and point out that the cost depends on several factors, including (i) the number of variables, (ii) the number of equations, (iii) the degree of the equations, and (iv) the considered representative of the system of equations.

Even–Mansour Construction. In Ciminion, the keyed permutation $\mathcal{P}$ is chosen in order to resemble a PRP. Indeed, since $\mathcal{P}$ is computed only once, it has little impact on the overall cost. Further, if $\mathcal{P}$ resembles a PRP, it is unlikely that an attacker can create texts with a special structure at the input of $\mathcal{P}^{(e)}$. This allows for a simplified security analysis of the expansion phase, as it rules out attacks that require control of the inputs of $\mathcal{P}^{(e)}$.

By performing a key addition before the expansion phase, the first part of the scheme becomes an Even–Mansour construction [29] of the form $x \mapsto \mathtt{K} + \mathcal{P}(x + \mathtt{K})$. As proven in [20,28], an Even–Mansour scheme is indistinguishable from a random permutation up to $q^{n/2}$ queries for $\mathtt{K} \in \mathbb{F}_q^n$, assuming both the facts that (i) the unkeyed permutation $\mathcal{P}$ behaves as a pseudo-random public permutation, and that (ii) the attacker knows both the inputs and outputs of the construction. Since $n/2 \cdot \log_2(q)$ is higher than our security level, this allows us to make a security claim on a subcomponent of the entire scheme, and so to further simplify the overall security analysis.

Keyed Permutation in the Expansion Phase. In Farfalle, the final key addition is crucial against attacks inverting the final permutation $\mathcal{P}^{(e)}$. However, an attacker can cancel the influence of the key by using the differences of two outputs if the key schedule is linear. For example, assume that the key schedule for the expansion phase is the identity map (as in Farfalle), and let x be the input of the expansion phase. Let $y_j = \mathtt{K} + \mathcal{P}^{(e)}(\mathcal{R}_j(x))$ and $y_h = \mathtt{K} + \mathcal{P}^{(e)}(\mathcal{R}_h(x))$ be two outputs of the expansion phase. Any difference of the form

$$y_j - y_h = \mathcal{P}^{(e)}(\mathcal{R}_j(x)) - \mathcal{P}^{(e)}(\mathcal{R}_h(x)) \tag{4}$$

results in a system of equations that is *independent of the key* or, equivalently, that depends only on the intermediate unknown state. This is an advantage when trying to solve the associated polynomial system with Gröbner bases.

The key in Ciminion has been moved from the end of $\mathcal{P}^{(e)}$ to the beginning, with the goal of preventing its cancellation by considering differences of the outputs. Inverting $\mathcal{P}^{(e)}$ is instead prevented by introducing a final truncation, which has the side effect of reducing the output size and thus the throughput.

Recently, in [8] the authors showed that moving the key inside of $\mathcal{P}^{(e)}$ is actually *not* sufficient by itself for preventing the construction of a system of equations – similar to (Eq. (4)) – which is independent of the secret key. For this reason, instead of working with a permutation-based non-invertible function, we propose to instantiate the last permutation with a block cipher $\mathcal{C}_k$, defined as an iterated permutation with a key addition in each round. In this way, we achieve the advantages of both $^{1/2\times}$Farfalle and Ciminion. First, the output size of $\mathcal{C}_k$ is equal to the input size and it is not possible to invert $\mathcal{C}_k$ without guessing the key (as in $^{1/2\times}$Farfalle). Secondly, a carefully chosen $\mathcal{C}_k$ will prevent the possibility to set up a system of equations for the expansion part that is independent of the key by considering differences of outputs (as in Ciminion).

Feed-Forward in Expansion Phase and Nonlinear Rolling Function. The proposed changes in Megafono may allow new potential problems. Let $v_j = \mathcal{C}_k\left(\mathcal{R}_j(y,z)\right)$ and $v_l = \mathcal{C}_k\left(\mathcal{R}_l(y,z)\right)$ be two outputs of the expansion phase for a shared input (y,z) and let $\mathcal{R}'_{j-l}$ denote the function satisfying $\mathcal{R}'_{j-l} \circ \mathcal{R}_l(\cdot) = \mathcal{R}_j(\cdot)$ for $j > l$. Since $\mathcal{C}_k(\cdot)$ is invertible for each fixed k, we have that

$$\forall j > l : \qquad \mathcal{C}_k \circ \mathcal{R}'_{j-l} \circ \mathcal{C}_k^{-1}(v_l) = v_j \implies \mathcal{R}'_{j-l} \circ \mathcal{C}_k^{-1}(v_l) = \mathcal{C}_k^{-1}(v_j)\,.$$

That is, it is possible to set up a system of equations that depend on the keys only (equivalently, that do not depend on the internal unknown state (y,z)). We therefore apply the feed-forward technique on the expansion phase, i.e., we work with $(y,z) \mapsto \mathcal{F}_k(y,z) := \mathcal{C}_k(y,z) + (y,z)$, which prevents this problem.

Assume moreover that the functions $\mathcal{R}_i$, $i \geq 1$ are linear. Given two outputs $w_j = \mathcal{F}_k\left(\mathcal{R}_j(y,z)\right)$ and $w_l = \mathcal{F}_k\left(\mathcal{R}_l(y,z)\right)$,

$$\begin{aligned}
\mathcal{R}'_{j-l}(w_l) - w_j &= \mathcal{R}'_{j-l}\left(\mathcal{F}_k\left(\mathcal{R}_l(y,z)\right)\right) - \mathcal{F}_k\left(\mathcal{R}_j(y,z)\right)\\
&= \mathcal{R}'_{j-l}\left(\mathcal{R}_l(y,z) + \mathcal{C}_k\left(\mathcal{R}_l(y,z)\right)\right) - \mathcal{R}_j(y,z) - \mathcal{C}_k\left(\mathcal{R}_j(y,z)\right)\\
&= \mathcal{R}'_{j-l}\left(\mathcal{R}_l(y,z)\right) + \mathcal{R}'_{j-l}\left(\mathcal{C}_k\left(\mathcal{R}_l(y,z)\right)\right) - \mathcal{R}_j(y,z) - \mathcal{C}_k\left(\mathcal{R}_j(y,z)\right)\\
&= \mathcal{R}'_{j-l}\left(\mathcal{C}_k\left(\mathcal{R}_l(y,z)\right)\right) - \mathcal{C}_k\left(\mathcal{R}_j(y,z)\right)
\end{aligned}$$

for each j, l with $j > l$. Similar equations can be derived for affine $\mathcal{R}_i$. Even if we are not aware of any attack that exploits such an equality, we suggest to work with a nonlinear rolling function. We point out that using a nonlinear function is also suggested by Farfalle's designers in order to frustrate meet-in-the-middle attacks in the expansion phase (see [10, Sect. 5] for more details).

Creating New Variables to Replace a Heavy Key Schedule. Due to the structure of $^{1/2\times}$Farfalle and Ciminion, and under the assumption that $\mathcal{P}$ behaves like a PRP, an attacker cannot control the inputs and outputs of the expansion phase. However, (meet-in-the-middle) attacks that require only the knowledge of the outputs of such an expansion phase are possible, because multiple outputs are created via a single *common* (unknown) input. The cost of such an attack depends on the number of involved variables and on the degree of the equations. We start by examining how Farfalle and Ciminion prevent such an attack.

Farfalle has been proposed for achieving the best performances in software and/or hardware implementations. For this reason, the field considered in applications is typically $\mathbb{F}_2^n$, where n is large (at least equal to the security level k). This implies that a large number of variables is needed to model the scheme as a polynomial system, which prevents the attack previously described, even when working with a low-degree permutation $\mathcal{P}^{(e)}$. Depending on the details of the permutation, the number of variables could be minimized by working over an equivalent field $\mathbb{F}_{2^l}^m$ where $n = m \cdot l$, without crucially affecting the overall degree of the equations that describe the scheme. For instance, 16 variables, as opposed to 128, are sufficient for describing AES, since all its internal operations (namely, the S-box, ShiftRows, and MixColumns) are naturally defined over $\mathbb{F}_{2^8}^{16}$. This is not the case for SHA-3/Keccak, for which only the nonlinear layer (defined as the concatenation of χ functions) admits a natural description over $\mathbb{F}_{2^5}^{5 \cdot l}$. In general, this scenario can easily be prevented when working with weak-arranged SPN schemes [17] and/or unaligned SPN schemes [14], for which this equivalent representation that minimizes the number of variables comes at the price of huge/prohibitive degrees of the corresponding functions.

Ciminion has, on the other hand, been proposed for minimizing the multiplicative complexity in the natural representation of the scheme over $\mathbb{F}_q^n$ for *large/huge* q and *small* n, namely, the opposite of Farfalle. Hence, in order to work with low-degree permutations $\mathcal{P}^{(e)}$, it is necessary to "artificially" increase the number of variables to prevent attacks. By using a heavy key schedule, one can guarantee that the algebraic relation between the keys $k_0, k_1, k_2, \ldots$ is non-trivial, i.e., described by dense algebraic functions of high degree. Such a complex relation could not be exploited in an algebraic attack, and the attacker is then forced to treat the subkeys as independent variables. To summarize,

- in Farfalle, the (MitM) attack on the expansion phase is prevented by working over a field $\mathbb{F}_p^n$ for a small prime p and a large integer n, and
- in Ciminon, it is prevented by "artificially" increasing the number of variables, working with a heavy key schedule.

None of the two approaches is suitable for our goal, since we mainly target applications over a field $\mathbb{F}_p^n$ for a huge prime p in which a heavy key schedule cannot be computed efficiently. For this reason, we propose to increase the number of variables "for free" by reusing the computation needed to evaluate $\mathcal{P}$. Since $\mathcal{P}$ is instantiated as an iterated permutation in practical use cases, we can fabricate a new $\mathbb{F}_q^n$ element by considering the sum of all internal states of $\mathcal{P}$. This corresponds to the definition of the function $\mathcal{S}$ in Eq. (3). In this way, we can double the size of the internal state (and so, the number of variables) for free.

In more detail, for a given input $x \in \mathbb{F}_q^n$, let $y \in \mathbb{F}_q^n$ be the output $\mathtt{K} + \mathcal{P}(x + \mathtt{K})$, and let $z \in \mathbb{F}_q^n$ be the output $\mathcal{S}(x + \mathtt{K})$. Then y and z are not independent, since $z = \mathcal{S}(\mathcal{P}^{-1}(y - \mathtt{K}))$.[10] However, for proper choices of $\mathcal{P}$ and $\mathcal{S}$, the relation between the two variables is too complex to be exploited in practice, exactly as

[10] Note that it is not possible to define y as a function of z, since there is no way to uniquely recover x given z.

in the case of the keys $k_0, k_1, k_2, \dots$ in Ciminion. As a result, the attacker is forced to consider both y and z as two independent variables, which is exactly our goal.

Similar Techniques in the Literature. For completeness, we mention that the idea of reusing internal states of an iterated function is not new in the literature. E.g., let $E_k^{(r)}$ be an iterated cipher of $r \geq 1$ rounds. In [45], the authors set up a PRF F as the sum of the output of the iterated cipher after r rounds and the output after s rounds for $s \neq r$, that is, $F(x) = E_k^{(r)}(x) + E_k^{(s)}(x)$. Later on, a similar approach has been exploited in the Fork design strategy [6], which is an expanding invertible function defined as $x \mapsto E_{\hat{k}}^{(r_0)}(E_k^{(s)}(x)) \| E_{\tilde{k}}^{(r_1)}(E_k^{(s)}(x))$.

4.2 Modes of Use of Megafono

As in the case of Farfalle and Ciminion, Megafono can be used for key derivation and key-stream generation. It allows amortizing the computation of the key among different computations with the same initial master key $\mathtt{K}$. Besides that, other possible use cases of Megafono are a wide block cipher, in which Megafono is used to instantiate the round function of a contracting Feistel scheme, and a (session-supporting) authenticated encryption scheme. Since these applications were also proposed for Farfalle, we do not describe them here, but refer to [10, Sect. 4] for further details.

We conclude by pointing out the following. Megafono is designed to be competitive for applications that require a natural description over $\mathbb{F}_q^n$, where q is a large prime of order at least 2^{64}. However, this does not mean that Megafono cannot be efficiently used in other applications, e.g., for designing schemes that aim to be competitive in software or hardware. From this point of view, the main difference with respect to Farfalle and Ciminion is the fact that Megafono requires two permutations with different domains, namely, $\mathbb{F}_q^n$ and $\mathbb{F}_q^{2n}$. However, this is not a problem when e.g. considering the family of the SHA-3/Keccak permutations [12], defined over $\mathbb{F}_2^n$ for $n = 25 \cdot 2^l$ for $l \in \{0, 1, \dots, 6\}$. In this case it is possible to instantiate $\mathcal{P}$ and $\mathcal{C}_k$ with two unkeyed/keyed permutations defined over domains whose size differs by a factor of two. The resulting PRF based on Megafono would be similar to the PRF Kravatte based on Farfalle proposed in [10, Sect. 7]. (Proposing concrete round numbers for this version is beyond the scope of this paper. Rather, we leave the open problem to evaluate and compare the performances of the two PRFs for future work.)

5 Specification of Hydra

5.1 The PRF Hydra

Let $p > 2^{63}$ (i.e., $\lceil \log_2(p) \rceil \geq 64$) and let $t \geq 4$ be the size of the output. The security level is denoted by κ, where $2^{80} \leq 2^{\kappa} \leq \min\{p^2, 2^{256}\}$, and $\mathtt{K} \in \mathbb{F}_p^4$ is

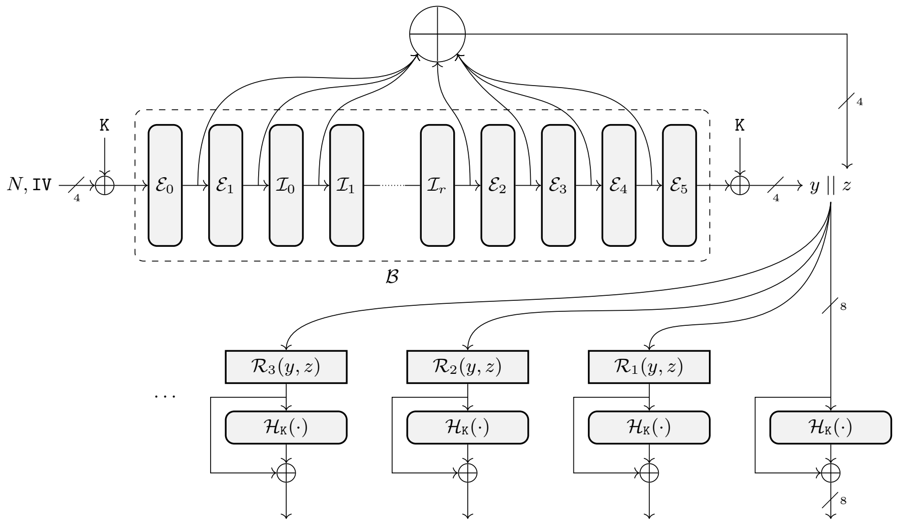


Fig. 3: The Hydra PRF (where $r := R_{\mathcal{I}} - 1$ for aesthetic reasons).

the master key. We assume that the data available to an attacker is limited to $2^{40} \leq 2^{\kappa/2} \leq \min\{p, 2^{128}\}$. For a plaintext $P \in \mathbb{F}_p^t$, the ciphertext is defined by

$$C = \textsc{Hydra}([N \mid\mid \texttt{IV}]) + P,$$

where $\textsc{Hydra} : \mathbb{F}_p^4 \to \mathbb{F}_p^t$ is the Hydra PRF, $\texttt{IV} \in \mathbb{F}_p^3$ is a fixed initial value and $N \in \mathbb{F}_p$ is a nonce (e.g., a counter).

Hydra. An overview of Hydra[11] is given in Fig. 3, where

(1) $y := \texttt{K} + \mathcal{B}([N \mid\mid \texttt{IV}] + \texttt{K}) \in \mathbb{F}_p^4$ for a certain permutation $\mathcal{B} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ defined in the following,

(2) $z \in \mathbb{F}_p^4$ defined as $z =: \mathcal{S}_{\texttt{K}}([N \| \texttt{IV}])$ for the non-invertible function $\mathcal{S}_{\texttt{K}} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ which corresponds to the sum of the internal states of $\texttt{K} + \mathcal{B}([N \| \texttt{IV}] + \texttt{K})$,

(3) $\mathcal{H}_{\texttt{K}} : \mathbb{F}_p^8 \to \mathbb{F}_p^8$ is a keyed permutation defined in Section 5.4, and

(4) the functions $\mathcal{R}_i : \left(\mathbb{F}_p^4\right)^2 \to \mathbb{F}_p^8$ are defined as

$$\forall i \geq 1 : \qquad \mathcal{R}_i(y,z) := \varphi_i + \mathcal{R} \circ \mathcal{R}_{i-1}(y,z)\,, \tag{5}$$

where $\mathcal{R}_0(y,z) = (y,z)$, and where $\mathcal{R} : \left(\mathbb{F}_p^4\right)^2 \to \mathbb{F}_p^8$ is the rolling function defined in Section 5.3, and $\varphi_i \in \mathbb{F}_p^8$ are random constants.

We give an algorithmic description of Hydra in [34, Appendix E].

[11] The (Lernaean) Hydra is a mythological serpentine water monster with many heads. In our case, we can see $\mathcal{B}$ as the body of the Hydra, and the multiple parallel permutations $\mathcal{H}_{\texttt{K}}$ as its multiple heads.

5.2 The Body of the Hydra: The Permutation $\mathcal{B}$

The permutation $\mathcal{B} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ is defined as

$$\mathcal{B}(x) = \underbrace{\mathcal{E}_5 \circ \cdots \circ \mathcal{E}_2}_{4\text{ times}} \circ \underbrace{\mathcal{I}_{R_\mathcal{I}-1} \circ \cdots \circ \mathcal{I}_0}_{R_\mathcal{I}\text{ times}} \circ \underbrace{\mathcal{E}_1 \circ \mathcal{E}_0}_{2\text{ times}}(M_\mathcal{E} \times x), \tag{6}$$

where the external and internal rounds $\mathcal{E}_i, \mathcal{I}_j : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ are defined as

$$\mathcal{E}_i(\cdot) = \varphi^{(\mathcal{E},i)} + M_\mathcal{E} \times S_\mathcal{E}(\cdot), \qquad \mathcal{I}_j(\cdot) = \varphi^{(\mathcal{I},j)} + M_\mathcal{I} \times S_\mathcal{I}(\cdot)$$

for $i \in \{0, 1, \ldots, 5\}$ and each $j \in \{0, 1, \ldots, R_\mathcal{I} - 1\}$, where $\varphi^{(\mathcal{E},i)}, \varphi^{(\mathcal{I},j)} \in \mathbb{F}_p^4$ are randomly chosen round constants (we refer to [34, Appendix E] for details on how we generate the pseudo-random constants).

The Round Function $\mathcal{E}$. Let $d \geq 3$ be the *smallest* odd integer such that $\gcd(d, p-1) = 1$. The nonlinear layer $S_\mathcal{E} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ is defined as

$$S_\mathcal{E}(x_0, x_1, x_2, x_3) = (x_0^d, x_1^d, x_2^d, x_3^d)\,.$$

We require $M_\mathcal{E} \in \mathbb{F}_p^{4\times 4}$ to be an MDS matrix and recommend an AES-like matrix such as $\mathrm{circ}(2, 3, 1, 1)$ or $\mathrm{circ}(3, 2, 1, 1)$.

The Round Function $\mathcal{I}$. The nonlinear layer $S_\mathcal{I} : \mathbb{F}_p^4 \to \mathbb{F}_p^4$ is defined as $S_\mathcal{I}(x_0, x_1, x_2, x_3) = (y_0, y_1, y_2, y_3)$ where

$$y_l = x_l + \left(\left(\sum_{j=0}^{3} (-1)^j \cdot x_j\right)^2 + \left(\sum_{j=0}^{3} (-1)^{\lfloor j/2 \rfloor} \cdot x_j\right)\right)^2 \quad \text{for } 0 \leq l \leq 3\,. \tag{7}$$

Note that the two vectors $\lambda^{(0)} := (1, -1, 1, -1), \lambda^{(1)} := (1, 1, -1, -1) \in \mathbb{F}_p^4$, that define the coefficients in the sums of (7), are linearly independent and their entries sum to zero. This latter condition is needed to guarantee invertibility by Proposition 1. $M_\mathcal{I} \in \mathbb{F}_p^{4\times 4}$ is an invertible matrix that satisfies the following conditions (which are justified in [34, Appendix G.2]):

(a) for each $i \in \{0, 1\}$: $\sum_{j=0}^{3} \lambda_j^{(i)} \cdot \left(\sum_{l=0}^{3} M_\mathcal{I}[j, l]\right) \neq 0$,

(b) for each $i \in \{0, 1\}$ and each $j \in \{0, 1, \ldots, 3\}$: $\sum_{l=0}^{3} \lambda_l^{(i)} \cdot M_\mathcal{I}[l, j] \neq 0$, and

(c) its *minimal polynomial* is of maximum degree and irreducible (for preventing infinitely long subspace trails – see [34, Appendix H] for details).

In particular, we suggest using an invertible matrix of the form

$$M_\mathcal{I} = \begin{pmatrix} \mu_{0,0}^{(\mathcal{I})} & 1 & 1 & 1 \\ \mu_{1,0}^{(\mathcal{I})} & \mu_{1,1}^{(\mathcal{I})} & 1 & 1 \\ \mu_{2,0}^{(\mathcal{I})} & 1 & \mu_{2,2}^{(\mathcal{I})} & 1 \\ \mu_{3,0}^{(\mathcal{I})} & 1 & 1 & \mu_{3,3}^{(\mathcal{I})} \end{pmatrix}, \tag{8}$$

for which the conditions *(a)*, *(b)*, and *(c)* are satisfied (we suggest to use the tool given in [34, Appendix H.1] in order to check that the condition *(c)* is satisfied).

5.3 The Rolling Function

The rolling function $\mathcal{R} : \left(\mathbb{F}_p^4\right)^2 \to \mathbb{F}_p^8$ is defined as $\mathcal{R}(y,z) = M_{\mathcal{R}} \times S_{\mathcal{R}}(y,z)$, where a round constant is included in the definition of $\mathcal{R}_i$ (Eq. (5)) and the nonlinear layer $S_{\mathcal{R}}$ is defined as

$$S_{\mathcal{R}}(y_0, y_1, y_2, y_3, z_0, z_1, z_2, z_3) = (y_0 + v, \ldots, y_3 + v, z_0 + w, \ldots, z_3 + w),$$

with $v, w \in \mathbb{F}_p$ defined as

$$v = \left(\sum_{i=0}^{3} (-1)^i \cdot y_i\right) \cdot \left(\sum_{i=0}^{3} (-1)^{\lfloor \frac{i}{2} \rfloor} \cdot z_i\right), \; w = \left(\sum_{i=0}^{3} (-1)^i \cdot z_i\right) \cdot \left(\sum_{i=0}^{3} (-1)^{\lfloor \frac{i}{2} \rfloor} \cdot y_i\right), \tag{9}$$

and the linear layer $M_{\mathcal{R}} \in \mathbb{F}_p^{8\times 8}$ is defined as

$$M_{\mathcal{R}} = \mathrm{diag}(M_{\mathcal{I}}, M_{\mathcal{I}}) = \begin{pmatrix} M_{\mathcal{I}} & 0^{4\times 4} \\ 0^{4\times 4} & M_{\mathcal{I}} \end{pmatrix},$$

where $M_{\mathcal{I}} \in \mathbb{F}_p^{4\times 4}$ is the matrix just defined for the body's internal rounds.

5.4 The Heads of the Hydra: The Permutation $\mathcal{H}_{\mathtt{K}}$

The keyed permutation $\mathcal{H}_{\mathtt{K}} : \mathbb{F}_p^8 \to \mathbb{F}_p^8$ is defined as

$$\mathcal{H}_{\mathtt{K}}(y,z) = \underbrace{\mathtt{K}' + \mathcal{J}_{R_{\mathcal{H}}-1} \circ (\mathtt{K}' + \mathcal{J}_{R_{\mathcal{H}}-2}) \circ \ldots \circ (\mathtt{K}' + \mathcal{J}_1) \circ (\mathtt{K}' + \mathcal{J}_0)}_{R_{\mathcal{H}} \text{ times}}(y,z),$$

where $\mathtt{K}' = \mathtt{K} \,||\, (M_{\mathcal{E}} \times \mathtt{K}) \in \mathbb{F}_p^8$, and $\mathcal{J}_j : \mathbb{F}_p^8 \to \mathbb{F}_p^8$ is defined as

$$\mathcal{J}_i(\cdot) = \varphi_i + M_{\mathcal{J}} \times S_{\mathcal{J}}(\cdot),$$

where $\varphi_i \in \mathbb{F}_p^8$ are random round constants for each $i \in \{0, 1, \ldots, R_{\mathcal{H}} - 1\}$. The nonlinear layer $S_{\mathcal{J}}(x_0, x_1, \ldots, x_7) = (y_0, \ldots, y_7)$ is defined by

$$y_l = x_l + \left(\sum_{h=0}^{7} (-1)^{\lfloor \frac{h}{4} \rfloor} \cdot x_h\right)^2 \quad \text{for } 0 \le l \le 7\,.$$

As in (7), we note that the coefficients in the sum, $(1, 1, 1, 1, -1, -1, -1, -1)$, sums to zero. $M_{\mathcal{J}} \in \mathbb{F}_p^{8\times 8}$ is an invertible matrix that fulfills similar conditions to *(a)*, *(b)*, and *(c)* described in Section 5.2, i.e., *(a)* $\sum_{h=0}^{7} (-1)^h \cdot \left(\sum_{l=0}^{7} M_{\mathcal{J}}[h,l]\right) \neq 0$, *(b)* $\sum_{l=0}^{7} (-1)^l \cdot M_{\mathcal{J}}[l,h] \neq 0$, for $h \in \{0, \ldots, 7\}$, and *(c)* the minimal polynomial of $M_{\mathcal{J}}$ is of maximum degree and irreducible (as detailed in [34, Appendix H]). We recommend that $M_{\mathcal{J}}$ has a similar form to the matrix in Eq. (8) for eight rows and columns.

5.5 Number of Rounds

In order to provide κ bits of security and assuming a data limit of $2^{\kappa/2}$, the number of rounds for the functions $\mathcal{B}$ and $\mathcal{H}_{\mathrm{K}}$ must be at least

$$R_{\mathcal{I}} = \left\lceil 1.125 \cdot \left\lceil \max\left\{ \frac{\kappa}{4} - \log_2(d) + 6, \widehat{R_{\mathcal{I}}} \right\} \right\rceil \right\rceil , \quad R_{\mathcal{H}} = \lceil 1.25 \cdot \max\{24, 2 + R_{\mathcal{H}}^*\} \rceil ,$$

where $\widehat{R_{\mathcal{I}}}$ and $R_{\mathcal{H}}^*$ are the minimum positive integers that satisfy [34, Appendix G.2] and Eq. (12), respectively. Note that we have added a security margin of 12.5% for $\mathcal{B}$ and 25% for $\mathcal{H}_{\mathrm{K}}$. In [34, Appendix A], we provide a script that returns the number of rounds $R_{\mathcal{I}}$ and $R_{\mathcal{H}}$ for given p and κ. For instance, with $\kappa = 128$, we get $R_{\mathcal{I}} = 42$ and $R_{\mathcal{H}} = 39$. A concrete instantiation of Hydra's matrices for $p = 2^{127} + 45$ is given in in [34, Appendix C].

About Related-Key Attacks. We do *not* claim security against related-key attacks, since the keys are randomly sampled in each computation, without any input or influence of a potential attacker. Thus, an attacker cannot know or choose any occurring relations between different keys. Indeed, since we focus on MPC protocols in a malicious setting with either honest or dishonest majority (e.g., SPDZ [24,23]), any difference added to one shared key would be immediately detected by the other parties in the protocol. We also emphasize that the same assumption has been made in previous related works [32,26].

6 Design Rationale of $\mathcal{B}$, $\mathcal{R}_i$ and $\mathcal{H}_{\mathrm{K}}$

6.1 The Body $\mathcal{B}$

The Hades Design Strategy. For $\mathcal{B}$, we aim to retain the advantages of Hades [32], in particular the security arguments against statistical attacks and the efficiency of the partial middle rounds. The Hades strategy is a way to design SPN schemes over $\mathbb{F}_q^t$ in which rounds with *full S-box layers* are mixed with rounds with *partial S-box layers*. The external rounds with full S-box layers (t S-boxes in each nonlinear layer) at the beginning and at the end of the construction provide security against statistical attacks. The rounds with partial S-box layers ($t' < t$ S-boxes and $t - t'$ identity functions) in the middle of the construction are more efficient in settings such as MPC and help to prevent algebraic attacks. In all rounds, the linear layer is defined via the multiplication of an MDS matrix.

This strategy has recently been pushed to its limit in Neptune [33], a modified version of the sponge hash function Poseidon [31]. In such a case, instead of using the same matrix and the same S-box both for the external and the internal rounds, Neptune's designers propose to use two different S-boxes and two different matrices for the external and internal rounds.

The External Rounds of $\mathcal{B}$. As in HADES, POSEIDON, and NEPTUNE, we use the external rounds to provide security against statistical attacks. In the case of HADES and POSEIDON, this is achieved by instantiating the external full rounds with power maps $x \mapsto x^d$ for each of the t words. We recall that this nonlinear layer requires $t \cdot (\mathrm{hw}(d) + \lfloor \log_2(d) \rfloor - 1)$ multiplications (see e.g. [33] for details).

We adopt this approach for $\mathcal{B}$, using 2 external rounds at the beginning and $2 + 2 = 4$ external rounds at the end, where 2 rounds are included as a security margin against statistical attacks (see [34, Appendix G.1] for more details). With respect to HADES and POSEIDON, we do not impose that the number of external rounds at the beginning is equal to the number of external rounds at the end (even if we try to have a balance between them). Instead, we choose the number of external rounds to be even at each side in order to maximize the minimum number of active S-boxes from the wide-trail design strategy [22] (the minimum number of active S-boxes over two consecutive rounds is related to the branch number of the matrix that defines the linear layer).

The Internal Rounds of $\mathcal{B}$. To minimize our primary cost metric (the number of multiplications over $\mathbb{F}_p$), we opt for using maps with degree $2^l \geq 2$ which cost $l \geq 1$ multiplications in the internal rounds. Indeed, let us compare the cost in terms of $\mathbb{F}_p$ multiplications in order to reach a certain degree Δ when using a round instantiated with the quadratic map $x \mapsto x^2$, with one instantiated via an invertible power map $x \mapsto x^d$ with $d \geq 3$, for odd d. Comparing the overall number of $\mathbb{F}_p$ multiplications, the first option is the most competitive, since

$$\underbrace{\lceil \log_2(\Delta) \rceil = \lceil \log_d(\Delta) \cdot \log_2(d) \rceil}_{\text{using } x \mapsto x^2} \leq \underbrace{\lceil \log_d(\Delta) \rceil \cdot (\lfloor \log_2(d) \rfloor + \mathrm{hw}(d) - 1)}_{\text{using } x \mapsto x^d},$$

where $\lceil \log_d(\Delta) \cdot \log_2(d) \rceil \leq \lceil \log_d(\Delta) \rceil \cdot \lceil \log_2(d) \rceil$ and $\lfloor \log_2(d) \rfloor + \mathrm{hw}(d) - 1 \geq \lfloor \log_2(d) \rfloor + 1 = \lceil \log_2(d) \rceil$. For example, consider $d = 3, \Delta = 2^{128}$. With quadratic maps we need 128 $\mathbb{F}_p$ multiplications to reach degree Δ. In the second case, 162 $\mathbb{F}_p$ multiplications are needed, requiring 27% more multiplications in total.

Nonlinear Layer. However, $x \mapsto x^2$ is not invertible, which may affect the security. Therefore, we use the quadratic map in a mode that preserves the invertibility, as in a Feistel or Lai–Massey construction [43]. The latter over $\mathbb{F}_q^2$ is defined as $(x, y) \mapsto (x + F(x - y), y + F(x - y))$, where $F : \mathbb{F}_q \to \mathbb{F}_q$. Generalizations over $\mathbb{F}_p^n$ have recently been proposed [33], including one defined as $(x_0, x_1, \ldots, x_{n-1}) \mapsto (y_0, y_1, \ldots, y_{n-1})$, where $y_i = x_i + F\left(\sum_{j=0}^{n-1} (-1)^j \cdot x_j\right)$ for $i \in \{0, 1, \ldots, n-1\}$ and even $n \geq 3$. This can be further generalized as follows.

Proposition 1. *Let $q = p^s$, where $p \geq 3$ is a prime and s is a positive integer, and let $n \geq 2$. Given $1 \leq l \leq n - 1$, let $\lambda_0^{(i)}, \lambda_1^{(i)}, \ldots, \lambda_{n-1}^{(i)} \in \mathbb{F}_q$ be such that $\sum_{j=0}^{n-1} \lambda_j^{(i)} = 0$ for $i \in \{0, 1, \ldots, l-1\}$. Let $F : \mathbb{F}_q^l \to \mathbb{F}_q$. The Lai-Massey function*

$\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ defined as $\mathcal{F}(x_0, \dots, x_{n-1}) = (y_0, \dots, y_{n-1})$ is invertible when

$$y_h = x_h + F\left(\sum_{j=0}^{n-1} \lambda_j^{(0)} \cdot x_j, \sum_{j=0}^{n-1} \lambda_j^{(1)} \cdot x_j, \dots, \sum_{j=0}^{n-1} \lambda_j^{(l-1)} \cdot x_j\right), \text{ for } 0 \le h \le n-1 .$$

We provide the proof in [34, Appendix F.1]. No conditions are imposed on F. Even if not strictly necessary, we choose $\{\lambda_j^{(0)}\}_{j=0}^{n-1}, \dots, \{\lambda_j^{(l-1)}\}_{j=0}^{n-1}$ such that they are linearly independent. Since we require $\sum_{j=0}^{n-1} \lambda_j^{(i)} = 0$ for $i \in \{0, \dots, l-1\}$, there can be at most $l = n-1$ linearly independent $\{\lambda_j^{(i)}\}$-vectors.

To reduce the number of rounds and matrix multiplications, we chose a generalized Lai–Massey construction instantiated with a nonlinear function of degree 4 that can be computed with 2 multiplications only.

Linear Layer. The Lai–Massey construction allows for invariant subspaces [48]. Hence, it is crucial to choose the matrix $M_{\mathcal{I}}$ in order to break them. For this goal, in [34, Appendix H], we show how to adapt the analysis/tool proposed in [36,37] for breaking arbitrarily long subspace trails for P-SPN schemes to the case of the generalized Lai–Massey constructions. In particular, based on [36, Proposition 13], we show that this result can be always achieved by choosing a matrix for which the minimal polynomial is of maximum degree and irreducible.

Moreover, the interpolation polynomial must be dense. Therefore, we require

(a) for $i \in \{0, 1, \dots, l-1\}$: $\sum_{j=0}^{n-1} \lambda_j^{(i)} \cdot \left(\sum_{k=0}^{n-1} M_{\mathcal{I}}[j,k]\right) \neq 0$,

(b) for $i \in \{0, 1, \dots, l-1\}$ and $j \in \{0, 1, \dots, n-1\}$: $\sum_{k=0}^{n-1} \lambda_k^{(i)} \cdot M_{\mathcal{I}}[k,j] \neq 0$.

We give further details on these two conditions in [34, Appendix G.2].

6.2 The Heads $\mathcal{H}_{\mathtt{K}}$

As in Farfalle and Ciminion, the attacker knows the outputs of the expansion phase of Megafono, but cannot choose them (to e.g. set up a chosen-ciphertext attack). By designing $\mathcal{B}$ in order to resemble a PRP, the attacker cannot know or choose the inputs of $\mathcal{H}_{\mathtt{K}}$ (i.e., the output of $\mathcal{B}$). Further, it is not possible to choose inputs of $\mathcal{B}$ which result in specific statistical/algebraic properties at the inputs of $\mathcal{H}_{\mathtt{K}}$. This severely limits the range of attacks that may work at the expansion phase of Megafono, and so of Hydra.

As a result, we find that the possible attacks are largely algebraic in nature, such as using Gröbner bases. The idea of this attack is to construct a system of equations that links the inputs and the outputs of $\mathcal{H}_{\mathtt{K}}$ in order to find the intermediate variables and the key. In our case, this corresponds to 12 variables: eight to represent the input and four variables related to the key. With this number of variables over such a large field (relative to the security level), we will see in Section 7.4 that it will not be necessary for $\mathcal{H}_{\mathtt{K}}$ to reach its maximal degree. Since $\mathcal{H}_{\mathtt{K}}$ is an iterated permutation, it is also possible to introduce new variables at the outputs of each round $\mathcal{J}_i$ in order to reduce the overall cost of the Gröbner basis attack. In such a case, the cost of the attack depends on $\min\{\deg(\mathcal{J}^{-1}), \deg(\mathcal{J})\}$. Indeed, since we can work at round level, each round function $y = \mathcal{J}(x)$ can be rewritten as $\mathcal{J}^{-1}(y) = x$, and the cost of the attack depends on the minimum degree among these equivalent representations.

Therefore, we instantiate the round function of $\mathcal{H}_{\text{K}}$ with a low-degree function, in particular a generalized Lai–Massey construction of degree 2 (where the matrix that defines the linear layer satisfies analogous condition to the ones given for $M_{\mathcal{I}}$). An alternative approach (used e.g. in Rescue) applies both high-degree and low-degree nonlinear power maps (recalled in Section 2.2). It is efficient in the MPC setting, and would prompt $\mathcal{H}_{\text{K}}$ to quickly reach its maximal degree. However, since reaching the maximal degree will not be a primary concern of ours (due to the high number of variables), we opt for the former choice of round functions, which allows Hydra to be fast in the plain setting as well.

6.3 The Rolling Functions $\mathcal{R}_i$

Finally, we consider a nonlinear rolling function, as already done in Xoofff [21] and Ciminion. This has multiple advantages, such as frustrating the meet-in-the-middle attacks on the expansion phase described in [15,19] and previously recalled in Section 4.1, and destroying possible relation between consecutive outputs due to the feed-forward operation (see Section 4.1 for details).

We work with a rolling function that is different from what is used in the heads, in order to break symmetry. The following (generalized) result ensures the invertibility of the chosen rolling function.

Proposition 2. *Let $n = 2 \cdot n' \geq 4$, with $n' \geq 2$, and $\{\lambda_i, \lambda'_i, \varphi_i, \varphi'_i\}_{0 \leq i \leq n'-1}$ be a set of constants in $\mathbb{F}_p \setminus \{0\}$ satisfying $\sum_{i=0}^{n'-1} \lambda_i = \sum_{i=0}^{n'-1} \lambda'_i = \sum_{i=0}^{n'-1} \varphi'_i = 0$. Let furthermore $G, H : \mathbb{F}_p \to \mathbb{F}_p$ be any $\mathbb{F}_p$ functions. Then the function $\mathcal{F}$ over $\mathbb{F}_p^n$ defined as $\mathcal{F}(x_0, \dots, x_{n-1}) = (y_0, \dots, y_{n-1})$ is invertible for*

$$y_i := \begin{cases} x_i + \left(\sum_{j=n'}^{n-1} \varphi_{j-n'} \cdot x_j\right) \cdot G\left(\sum_{j=0}^{n'-1} \lambda_j \cdot x_j\right) & \text{if } i \in \{0, \dots, n'-1\}, \\ x_i + \left(\sum_{j=0}^{n'-1} \varphi'_j \cdot x_j\right) \cdot H\left(\sum_{j=n'}^{n-1} \lambda'_{j-n'} \cdot x_j\right) & \text{if } i \in \{n', \dots, n-1\}. \end{cases}$$

The proof is given in [34, Appendix F.2]. We impose that $(\lambda_0, \dots, \lambda_{n'-1})$, $(\varphi'_0, \dots, \varphi'_{n'-1}) \in \mathbb{F}_p^{n'}$ and $(\varphi_0, \dots, \varphi_{n'-1})$, $(\lambda'_0, \dots, \lambda'_{n'-1}) \in \mathbb{F}_p^{n'}$ are pairwise linearly independent, in order to guarantee that the variables v and w in Eq. (9) are independent (i.e., there is no $\omega \in \mathbb{F}_p$ such that $v = \omega \cdot w$) with high probability.

As before, the matrix $M_{\mathcal{R}}$ is chosen in order to break infinitely long invariant subspace trails. Since the constants that defined the (generalized) Lai-Massey functions (namely, $(1, -1, 1, -1)$ and $(1, 1, -1, -1) \in \mathbb{F}_p^4$) are the same for the rolling function and for the body's internal rounds, we defined $M_{\mathcal{R}}$ via $M_{\mathcal{I}}$.

7 Security Analysis

Inspired by Ciminion, we choose the number of rounds such that $x \mapsto \mathtt{K}+\mathcal{B}(x+\mathtt{K})$ behaves like a PRP (where an attacker is free to choose its inputs and outputs) and no attack works on the expansion phase of Hydra. In the following, we motivate this choice and justify the number of rounds given in Section 5.5.

7.1 Overview

Attacks on the Body. Attacks taking into account the relations between the inputs and the outputs of Hydra are in general harder than the attacks taking into account the relations between the inputs and the outputs of $\mathcal{B}$. Hence, if an attacker is not able to break $x \mapsto \mathtt{K} + \mathcal{B}(x + \mathtt{K})$ if they have full control over the inputs and outputs, they cannot break Hydra by exploiting the relation of its inputs and outputs. Based on this fact, the chosen number of rounds guarantees that $x \mapsto \mathtt{K} + \mathcal{B}(x + \mathtt{K})$ resembles a PRP against attacks with a computational complexity of at most 2^{κ} and with a data complexity of at most $2^{\kappa/2}$.

We point out that this approach results in a very conservative choice for the number of rounds of $\mathcal{B}$. Indeed, in a realistic attack scenario the outputs of $x \mapsto \mathtt{K} + \mathcal{B}(x + \mathtt{K})$ are hidden by $\mathcal{H}_{\mathtt{K}}$, and the overall design will still be secure if $\mathcal{B}$ is instantiated with a smaller number of rounds. However, $\mathcal{B}$ is computed only once, and the overall cost grows linearly with the number of computed heads $\mathcal{H}_{\mathtt{K}}$. Hence, we find that the benefits of allowing us to simplify the security analysis of the heads outweighs this modest increase in computational cost.

Attacks on the Heads. In order to be competitive in MPC, we design $\mathcal{H}_{\mathtt{K}}$ such that Hydra is secure under the assumption that $\mathtt{K}+\mathcal{B}(x+\mathtt{K})$ behaves like a PRP. In particular, the attacker only knows the outputs of the $\mathcal{H}_{\mathtt{K}}$ calls, and cannot choose any inputs with particular statistical or algebraic properties. Hence, the only possibility is to exploit the relations among the outputs of consecutive $\mathcal{H}_{\mathtt{K}}$ calls, which originate from the same (unknown) input $y, z \in \mathbb{F}_p^4$. This can be used when constructing systems of polynomial equations from $\mathcal{H}_{\mathtt{K}}$. Indeed, we will later see that the most competitive attacks are Gröbner basis ones.

7.2 Security Analysis of $\mathcal{B}$

Since $\mathcal{B}$ is heavily based on the Hades construction, its security analysis is also similar. In particular, the external rounds of a Hades design provide security against statistical attacks. Since this part of $\mathcal{B}$ is the same as in HadesMiMC, the security analysis proposed in [32, Sect. 4.1 – 5.1] also applies here. The internal rounds of $\mathcal{B}$ are instantiated with a Lai–Massey scheme, while the internal rounds of HadesMiMC are instantiated with a partial SPN scheme. However, the security argument proposed for HadesMiMC in [32, Sect. 4.2 – 5.2] regarding algebraic attacks can be easily adapted to the case of $\mathcal{B}$.

We refer to [34, Appendix G] for more details. We point out that $x \mapsto \mathtt{K} + \mathcal{B}(x + \mathtt{K})$ is an Even–Mansour construction in which $\mathcal{B}$ is independent of the key, while a key addition takes place among every round in HadesMiMC. This fact is taken care of in the analysis proposed in [34, Appendix G], keeping in mind that the Even–Mansour construction cannot guarantee more than $2 \cdot \log_2(p) \geq \kappa$ bits of security [20,28] (this value is reached when $\mathcal{B}$ resembles a PRP).

Finally, in [34, Appendix H] we show how to choose the matrix that defines the linear layer of the internal rounds of $\mathcal{B}$ in order to break the invariant subspace trails of the Lai–Massey scheme, by modifying the strategy proposed in [36] for the case of partial SPN schemes.

7.3 Statistical and Invariant Subspace Attacks on $\mathcal{H}_{\mathtt{K}}$

It is infeasible for the attacker to choose inputs $\{x_j\}_j$ for $\mathcal{B}$ such that the corresponding outputs $\{y_j\}_j$ satisfy certain statistical/algebraic properties, which makes it hard to mount statistical attacks on the heads $\mathcal{H}_{\mathtt{K}}$. However, it is still desirable that $\mathcal{H}_{\mathtt{K}}$ has good statistical properties.

To this end, the matrix $M_{\mathcal{J}} \in \mathbb{F}_p^{8\times 8}$ is chosen such that no (invariant) subspace trail and probability-1 truncated differential can cover more than 7 rounds (see [34, Appendix H]). Hence, the probability of each differential characteristic over $R_{\mathcal{H}}$ rounds is at most $p^{-\lfloor R_{\mathcal{H}}/8 \rfloor}$, since the maximum differential probability of $S_{\mathcal{J}}$ is p^{-1} (see [34, Appendix I.1]) and at least one $S_{\mathcal{J}}$ function is active every 8 rounds. By choosing $R_{\mathcal{H}} \geq 24$, the probability of each differential characteristic is at most $p^{-3} \leq 2^{-1.5\kappa}$, which we conjecture to be sufficient for preventing differential and, more generally, other statistical attacks in the considered scenario.

7.4 Algebraic and Gröbner Basis Attacks on $\mathcal{H}_{\mathtt{K}}$

It is not possible to mount an interpolation attack, since the input y, z is unknown and the polynomials associated with the various heads differ for each i. Thus, the remainder of this section will be devoted to Gröbner basis attacks.

Note that the variables y and z are clearly not independent, as they both depend on x. Moreover, z can be written as a function of y (the converse does not hold, since the function that outputs z is, in general, not invertible). However, these functions would be dense and reach maximum degree, which implies that the cost of an attack making use of them would be prohibitively expensive. Hence, we will treat y and z as independent variables in the following.

Preliminaries: Gröbner Basis Attacks. The most efficient methods for solving multivariate systems over large finite fields involve computing a Gröbner basis associated with the system. We refer to [18] for details on the underlying theory.

Computing a Gröbner basis (in the grevlex order) is, in general, only one of the steps involved in solving a system of polynomials. In our setting, an attacker is able to set up an overdetermined polynomial system where a unique solution can be expected. In this case it is often possible to read the solution directly from the grevlex Gröbner basis, which is why we will solely focus on the step of computing said basis. There are no general complexity estimates for the running time of state-of-the-art Gröbner basis algorithms such as F_4 [30]. There is, however, an important class of polynomial systems, known as semi-regular (see [7] for a definition), that is well understood. For a semi-regular system the degree of the polynomials encountered in F_4 is expected to reach the degree of regularity D_{reg}, which in this case can be defined as the index of the first non-positive coefficient in the series

$$H(z) = \frac{\prod_{i=1}^{n_e}(1 - z^{d_i})}{(1 - z)^{n_v}}, \tag{10}$$

for n_e polynomials in n_v variables, where d_i is the degree of the i-th equation. The estimated complexity of computing a grevlex Gröbner basis is then

$$\mathcal{O}\left(\binom{D_{\mathrm{reg}} + n_v}{n_v}^{\omega}\right), \tag{11}$$

where $2 \leq \omega \leq 3$ is the linear algebra constant representing the cost of matrix multiplication and D_{reg} the associated degree of regularity [7].

Gröbner Basis Attacks on $\boldsymbol{\mathcal{H}_{\mathtt{K}}}$. There are many possible ways to represent a cryptographic construction as a system of multivariate polynomials, and this choice impacts the performance of the Gröbner basis algorithm. Note that the degree of $\mathcal{H}_{\mathtt{K}}(\mathcal{R}_i(y, z))$ increases with i, and it is therefore not possible to collect enough polynomials for solving by direct linearization at a relatively small degree, as discussed in [34, Appendix G.2]. Instead, we find that the most efficient attack includes only $\mathcal{H}_{\mathtt{K}}(y, z)$ and $\mathcal{H}_{\mathtt{K}}(\mathcal{R}_1(y, z))$ in a representation that introduces new variables and equations for each round. While this increases the number of variables, it keeps the degree low, and allows exploitation of the small number of multiplications in each round. We outline our findings in the following, and we refer to [34, Appendix I.2] for more details on the underlying arguments.

The most promising intermediate modeling can be reduced to a system of $2R_{\mathcal{H}} + 2$ quadratic equations in $2R_{\mathcal{H}} - 2$ variables, where $R_{\mathcal{H}}$ is the number of rounds in $\mathcal{H}_{\mathtt{K}}$. Further analysis shows that the tested systems are semi-regular, and in particular that the degrees encountered in the F_4 algorithm are well-estimated by the series $H(z)$ in Eq. (10). Solving times are also comparable to that of solving randomly generated semi-regular systems with the same parameters. Still, the systems from $\mathcal{H}_{\mathtt{K}}$ are sparser than what can be expected from randomly generated systems. To ensure that this cannot be exploited, we add 2 extra rounds on top of this baseline. Hence, for a security level κ we follow Eq. (11) and define $R^*_{\mathcal{H}} = R^*_{\mathcal{H}}(\kappa)$ to be the minimum positive integer such that

$$\binom{2R^*_{\mathcal{H}} - 2 + D_{\mathrm{reg}}}{2R^*_{\mathcal{H}} - 2}^2 \geq 2^{\kappa}\,, \tag{12}$$

where D_{reg} is computed from Eq. (10) using $n_e = 2R^*_{\mathcal{H}} + 2$ and $n_v = 2R^*_{\mathcal{H}} - 2$. We claim that $R^*_{\mathcal{H}}(\kappa) + 2$ is sufficient to provide κ-bit security against this attack.

Table 1: Online and offline phase performance in MPC for several constructions with state sizes t using a secret shared key. *Prec* is the number of precomputed elements (multiplication triples, squares, inverses). *Depth* describes the number of online communication rounds. The runtime is averaged over 200 runs.

t	Cipher	Rounds	Offline			Online			Combined	
			Prec.	Time ms	Data MB	Depth	Time ms	Data kB	Time ms	Data MB
8	Hydra	6, 42, 39	**171**	**39.99**	**3.86**	131	**6.81**	**5.37**	**46.80**	**3.87**
	Ciminion	90, 14	867	227.47	19.55	735	21.81	28.02	249.29	19.58
	HadesMiMC	6, 71	238	52.66	5.37	79	17.58	5.99	70.24	5.38
	Rescue	10	960	254.80	21.65	**33**	12.65	23.32	267.45	21.68
32	Hydra	6, 42, 39	**294**	**72.67**	**6.63**	134	**13.36**	**9.69**	**86.03**	**6.64**
	Ciminion	90, 14	3207	910.11	72.30	2895	84.37	103.29	994.47	72.41
	HadesMiMC	6, 71	526	137.49	11.87	79	225.86	13.29	363.35	11.88
	Rescue	10	3840	1253.76	86.60	**33**	109.80	92.82	1363.56	86.70
64	Hydra	6, 42, 39	**458**	**119.07**	**10.33**	138	**20.57**	**15.45**	**139.64**	**10.35**
	Ciminion	90, 14	6327	2262.55	142.64	5775	178.66	203.64	2441.21	142.84
	HadesMiMC	6, 71	910	251.44	20.53	79	899.55	23.02	1150.99	20.55
	Rescue	10	7680	2851.56	173.20	**33**	402.34	185.50	3253.90	173.39
128	Hydra	6, 42, 39	**786**	**206.08**	**17.72**	146	**37.49**	**26.97**	**243.58**	**17.75**
	Ciminion	90, 14	12567	4854.43	283.32	11535	328.79	404.34	5183.22	283.72
	HadesMiMC	6, 71	1678	463.59	37.85	79	4371.02	42.47	4834.61	37.89
	Rescue	10	15360	5934.39	346.40	**33**	1549.16	370.84	7483.55	346.77

Concrete Example for $\kappa = 128$. In this case we get $R^*_{\mathcal{H}}(128) = 29$, which in turn yields $n_e = 60$ quadratic equations in $n_v = 56$ variables. By expanding the resulting series in Eq. (10), we get $D_{\text{reg}} = 23$ for this system, and the security estimate $\binom{56+23}{56}^2 \approx 2^{130.8}$ follows. Thus, we claim that $R^*_{\mathcal{H}}(128) + 2 = 31$ is sufficient to provide 128-bit security against Gröbner basis attacks.

8 Hydra in MPC Applications

In this section, we evaluate the performance of Hydra compared to other PRFs in MPC use cases which assume a secret shared key. We implemented Hydra and its competitors using the MP-SPDZ library [41][12] (version 0.2.8, files can be found in [34, Appendix A]) and benchmark it using SPDZ [24,23] with the MASCOT [42] offline phase protocol. Concretely, we benchmark a two-party setting in a simulated LAN network (1 Gbit/s and $\ll 1$ ms average round-trip time) using a Xeon E5-2669v4 CPU (2.6 GHz), where each party is assigned only 1 core. SPDZ, and therefore all the PRFs, is instantiated using a 128-bit prime p, with $\gcd(3, p-1) = 1$, thus ensuring that $x \mapsto x^3$ is a permutation, as required by HadesMiMC, *Rescue*, MiMC, GMiMC, and Hydra. All PRFs are

[12] https://github.com/data61/MP-SPDZ/

instantiated with $\kappa = 128$. Hydra requires $4 \cdot R_{\mathcal{E}} \cdot (\mathrm{hw}(d) + \lfloor \log_2(d) \rfloor - 1) + 2 \cdot R_{\mathcal{I}} + (R_{\mathcal{H}} + 2) \cdot \lceil \frac{t}{8} \rceil - 2$ multiplications, hence $130 + 41 \cdot \lceil \frac{t}{8} \rceil$ in this setting.

We implemented all x^3 evaluations using the technique from [35], which requires one precomputed Beaver triple, one precomputed shared random square, and one online communication round. Furthermore, we implemented $x^{1/3}$ (as used in *Rescue*) using the technique described in [5]. MP-SPDZ allows to precompute squares and inverses from Beaver triples in an additional communication round in the offline phase (see Section 2).

In Table 1, we compare the performance of Hydra to some competitors when encrypting t plaintext words,[13] for a comparison with more PRFs we refer to [34, Appendix J]. We give concrete runtimes, as well as the amount of data transmitted by each party during the evaluation of the offline and online phases. Further, we give the combined number of triples, squares, and inverses which need to be created during the offline phase, as well as the total number of communication rounds (i.e., the depth of the PRF) in the online phase. In the offline phase only the required number of triples, squares, and inverses is precomputed.

Table 1 shows that the offline phase dominates both the overall runtime and the total communication between the parties. Hydra always requires less precomputation than Ciminion, HadesMiMC, and *Rescue*, hence, it has a significantly more efficient offline phase with the advantage growing with t. Looking at the online phase, Hydra is faster and requires less communication than its competitors, which is due to the smaller number of multiplications and the better plain performance. While Ciminion is slow due to the expensive key schedule, HadesMiMC requires many expensive MDS matrix multiplications (see [34, Appendix K]) and *Rescue* requires expensive $x^{1/d}$ evaluations.

For the sake of completeness, in Table 2 we also compare the performance of Hydra to Ciminion and *Rescue* in the case in which the round keys are already present. Comparing Hydra to Ciminion without a key schedule, one can observe that Ciminion's online phase is always faster. However, Hydra's number of multiplications scales significantly better than Ciminion's, hence, for larger state sizes ($t \geq 32$) Hydra has a faster offline phase performance, as well as less communication in the online phase.

To summarize, our experiments show that Hydra is the most efficient PRF in both phases of the MPC protocols. Only if we discard the key schedules, Ciminion is competitive for small state sizes $t < 32$. Thus, using Hydra leads to a significant performance improvement in MPC use cases, especially in high-throughput conditions. In applications, where the offline phase plays a minor role, e.g., when tripples are continuously precomputed and rarely consumed, Hydra still leads to an performance advantage due to requiring less communication between the parties, however, the advantage will be smaller.

[13] The use cases discussed in this paper basically boil down to encrypting many plaintext words using a secret-shared key. Hence, this benchmark is also representative for the use cases from Section 2.1.

Table 2: Online and offline phase performance in MPC for several constructions with state sizes t using a secret shared key. *Prec* is the number of precomputed elements (multiplication triples, squares, inverses). *Depth* describes the number of online communication rounds. The runtime is averaged over 200 runs.

				Offline			Online		Combined	
t	Cipher	Rounds	Prec.	Time ms	Data MB	Depth	Time ms	Data kB	Time ms	Data MB
8	Hydra	6, 42, 39	171	39.99	3.86	131	6.81	5.37	46.80	3.87
	Ciminion (No KS)[a]	90, 14	**148**	**35.64**	**3.34**	107	**3.98**	**5.02**	**39.62**	**3.35**
	Rescue (No KS)[a]	10	480	129.47	10.83	**33**	6.95	11.80	136.42	10.84
32	Hydra	6, 42, 39	**294**	**72.67**	**6.63**	134	13.36	**9.69**	**86.03**	**6.64**
	Ciminion (No KS)[a]	90, 14	328	80.79	7.40	119	**5.42**	11.16	86.21	7.41
	Rescue (No KS)[a]	10	1920	538.19	43.30	**33**	47.35	46.74	585.54	43.35
64	Hydra	6, 42, 39	**458**	**119.07**	**10.33**	138	20.57	**15.45**	**139.64**	**10.35**
	Ciminion (No KS)[a]	90, 14	568	154.38	12.81	135	**8.05**	19.35	162.42	12.83
	Rescue (No KS)[a]	10	3840	1226.39	86.60	**33**	144.14	93.34	1370.53	86.70
128	Hydra	6, 42, 39	**786**	**206.08**	**17.72**	146	37.49	**26.97**	**243.58**	**17.75**
	Ciminion (No KS)[a]	90, 14	1048	274.90	23.63	167	**10.70**	35.74	285.60	23.67
	Rescue (No KS)[a]	10	7680	2943.21	173.20	33	737.84	186.52	3681.05	173.39

[a] Assumes round keys are present, i.e., no key schedule computation in MPC.

The Effect of the Network. The performance of MPC applications depends on the network speed. A lower bandwidth leads to a larger effect of the communication between the parties on the overall performance. Moreover, a longer round-trip time leads to larger contributions of the number of communication rounds. In the offline phase only shared correlated randomness is created, thus the network performance affects all PRFs in the same way. Consequently, if a PRF has a faster offline phase in the LAN setting, it is also faster in a slower network environment. The situation is different in the online phase: In fast networks, the online phase performance is mostly determined by the plain runtime. In a slower network, more time is spent waiting for the network to deliver packages. Hydra has a small number of multiplications, hence a preferable offline phase in all networks. Further, it requires little communication in the online phase, making it suitable for low-bandwidth networks. However, it has a larger depth compared to HadesMiMC and *Rescue*, leading to worse runtimes in high-delay networks where runtime is dominated by `round_trip_time` $\times$ `depth`. Ciminion's key schedule has a large depth and requires lots of communication between the parties (compare *Data* column in Table 1 and Table 2). Thus, Ciminion is only competitive in slow networks if the key schedule does not need to be computed. Overall, Hydra has a good balance between a small number of multiplications, little communication, decent plain performance, and a reasonable depth, making it the preferred PRF for MPC applications in most network environments.

Acknowledgments. Lorenzo Grassi is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA. Morten Øygarden has been funded by The Research Council of Norway through the project "qsIoT: Quantum safe cryptography for the Internet of Things". Roman Walch is supported by the "DDAI" COMET Module within the COMET – Competence Centers for Excellent Technologies Programme, funded by the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry for Digital and Economic Affairs (bmdw), the Austrian Research Promotion Agency (FFG), the province of Styria (SFG) and partners from industry and academia. The COMET Programme is managed by FFG.

References

1. Abram, D., Damgård, I., Scholl, P., Trieflinger, S.: Oblivious TLS via multiparty computation. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 51–74. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_3
2. Albrecht, M.R., Grassi, L., Perrin, L., Ramacher, S., Rechberger, C., Rotaru, D., Roy, A., Schofnegger, M.: Feistel structures for MPC, and more. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019, Part II. LNCS, vol. 11736, pp. 151–171. Springer, Heidelberg (Sep 2019). https://doi.org/10.1007/978-3-030-29962-0_8
3. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 191–219. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_7
4. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_17
5. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Trans. Symm. Cryptol. **2020**(3), 1–45 (2020). https://doi.org/10.13154/tosc.v2020.i3.1-45
6. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: Forkcipher: A new primitive for authenticated encryption of very short messages. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 153–182. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-34621-8_6
7. Bardet, M., Faugére, J.C., Salvy, B., Yang, B.Y.: Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In: Proc. of MEGA. vol. 5 (2005)
8. Bariant, A., Bouvier, C., Leurent, G., Perrin, L.: Algebraic attacks against some arithmetization-oriented primitives. IACR Trans. Symmetric Cryptol. **2022**(3), 73–101 (2022). https://doi.org/10.46586/tosc.v2022.i3.73-101

9. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_34
10. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G.V., Keer, R.V.: Farfalle: parallel permutation-based cryptography. IACR Trans. Symm. Cryptol. **2017**(4), 1–38 (2017). https://doi.org/10.13154/tosc.v2017.i4.1-38
11. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_11
12. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The KECCAK reference (2011), https://keccak.team/files/Keccak-reference-3.0.pdf
13. Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J., Toft, T.: A practical implementation of secure auctions based on multiparty integer computation. In: Di Crescenzo, G., Rubin, A. (eds.) FC 2006. LNCS, vol. 4107, pp. 142–147. Springer, Heidelberg (Feb / Mar 2006). https://doi.org/10.1007/11889663_10
14. Bordes, N., Daemen, J., Kuijsters, D., Van Assche, G.: Thinking outside the superbox. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 337–367. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9_12
15. Chaigneau, C., Fuhr, T., Gilbert, H., Guo, J., Jean, J., Reinhard, J.R., Song, L.: Key-recovery attacks on full Kravatte. IACR Trans. Symm. Cryptol. **2018**(1), 5–28 (2018). https://doi.org/10.13154/tosc.v2018.i1.5-28
16. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1825–1842. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133997
17. Cid, C., Grassi, L., Gunsing, A., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Influence of the linear layer on the algebraic degree in sp-networks. IACR Trans. Symmetric Cryptol. **2022**(1), 110–137 (2022). https://doi.org/10.46586/tosc.v2022.i1.110-137
18. Cox, D., Little, J., O'Shea, D.: Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra. Springer Science & Business Media (2013)
19. Cui, T., Grassi, L.: Algebraic key-recovery attacks on reduced-round Xoofff. In: Dunkelman, O., Jr., M.J.J., O'Flynn, C. (eds.) SAC 2020. LNCS, vol. 12804, pp. 171–197. Springer, Heidelberg (Oct 2020). https://doi.org/10.1007/978-3-030-81652-0_7
20. Daemen, J.: Limitations of the Even-Mansour construction. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT'91. LNCS, vol. 739, pp. 495–498. Springer, Heidelberg (Nov 1991). https://doi.org/10.1007/3-540-57332-1_46
21. Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: The design of Xoodoo and Xoofff. IACR Trans. Symm. Cryptol. **2018**(4), 1–38 (2018). https://doi.org/10.13154/tosc.v2018.i4.1-38
22. Daemen, J., Rijmen, V.: The wide trail design strategy. In: Honary, B. (ed.) Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings. LNCS, vol. 2260, pp. 222–238. Springer (2001). https://doi.org/10.1007/3-540-45325-3_20

23. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (Sep 2013). https://doi.org/10.1007/978-3-642-40203-6_1
24. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_38
25. Dinur, I., Goldfeder, S., Halevi, T., Ishai, Y., Kelkar, M., Sharma, V., Zaverucha, G.: MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 517–547. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_18
26. Dobraunig, C., Grassi, L., Guinet, A., Kuijsters, D.: Ciminion: Symmetric encryption based on Toffoli-gates over large finite fields. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 3–34. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77886-6_1
27. Dobraunig, C., Kales, D., Rechberger, C., Schofnegger, M., Zaverucha, G.: Shorter signatures based on tailor-made minimalist symmetric-key crypto pp. 843–857 (Nov 2022). https://doi.org/10.1145/3548606.3559353
28. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in cryptography: The Even-Mansour scheme revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_21
29. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT'91. LNCS, vol. 739, pp. 210–224. Springer, Heidelberg (Nov 1993). https://doi.org/10.1007/3-540-57332-1_17
30. Faugére, J.C.: A new efficient algorithm for computing Gröbner bases (F_4). Journal of pure and applied algebra **139**(1-3), 61–88 (1999)
31. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 519–535. USENIX Association (Aug 2021)
32. Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., Schofnegger, M.: On a generalization of substitution-permutation networks: The HADES design strategy. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. LNCS, vol. 12106, pp. 674–704. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45724-2_23
33. Grassi, L., Onofri, S., Pedicini, M., Sozzi, L.: Invertible quadratic non-linear layers for mpc-/fhe-/zk-friendly schemes over fnp application to poseidon. IACR Trans. Symmetric Cryptol. **2022**(3), 20–72 (2022). https://doi.org/10.46586/tosc.v2022.i3.20-72
34. Grassi, L., Øygarden, M., Schofnegger, M., Walch, R.: From farfalle to megafono via ciminion: The PRF hydra for MPC applications (2022), https://eprint.iacr.org/2022/342
35. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: MPC-friendly symmetric key primitives. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 430–443. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978332

36. Grassi, L., Rechberger, C., Schofnegger, M.: Proving resistance against infinitely long subspace trails: How too choose the linear layer. IACR Trans. Symm. Cryptol. **2021**(2), 314–352 (2021). https://doi.org/10.46586/tosc.v2021.i2.314-352
37. Guo, C., Standaert, F.X., Wang, W., Wang, X., Yu, Y.: Provable security sp networks with partial non-linear layers. IACR Trans. Symm. Cryptol. **2021**(2), 353–388 (2021). https://doi.org/10.46586/tosc.v2021.i2.353-388
38. Helminger, L., Kales, D., Ramacher, S., Walch, R.: Multi-party revocation in sovrin: Performance through distributed trust. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 527–551. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_22
39. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250794
40. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 1447–1464. USENIX Association (Aug 2019)
41. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1575–1590. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417872
42. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 830–842. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978357
43. Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård, I. (ed.) EUROCRYPT'90. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (May 1991). https://doi.org/10.1007/3-540-46877-3_35
44. Laur, S., Talviste, R., Willemson, J.: From oblivious AES to efficient and secure database join in the multiparty setting. In: Jacobson Jr., M.J., Locasto, M.E., Mohassel, P., Safavi-Naini, R. (eds.) ACNS 13. LNCS, vol. 7954, pp. 84–101. Springer, Heidelberg (Jun 2013). https://doi.org/10.1007/978-3-642-38980-1_6
45. Mennink, B., Neves, S.: Optimal PRFs from blockcipher designs. IACR Trans. Symm. Cryptol. **2017**(3), 228–252 (2017). https://doi.org/10.13154/tosc.v2017.i3.228-252
46. Mohassel, P., Zhang, Y.: SecureML: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy. pp. 19–38. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.12
47. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176
48. Vaudenay, S.: On the Lai-Massey scheme. In: Lam, K.Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT'99. LNCS, vol. 1716, pp. 8–19. Springer, Heidelberg (Nov 1999). https://doi.org/10.1007/978-3-540-48000-6_2