Roman Walch

Improving Efficient Computation on Private Data

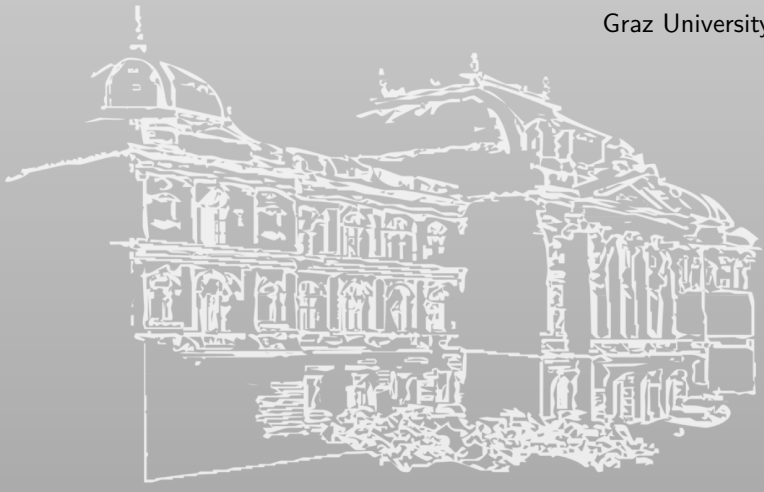
PhD Thesis

Assessors: Prof. Christian Rechberger, Prof. Thomas Schneider

January 2024

Institute of Applied Information Processing and Communications

Graz University of Technology



Roman Walch

Improving Efficient Computation on Private Data

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Christian Rechberger
Graz University of Technology

Assessor: Prof. Thomas Schneider
Technical University of Darmstadt

Institute of Applied Information Processing and Communications
Graz University of Technology

Graz, January 2024

Abstract

Cryptography is one of the fundamental building blocks for securing data in critical online infrastructures, such as online banking or online government services. However, traditional cryptographic primitives can only be used to protect data at rest and in transit, while the emergence of data protection regulations such as the GDPR and the increasing occurrence of outsourced computations demand protection of data in use as well. To this end, privacy enhancing technologies (PETs), such as homomorphic encryption (HE), secure multi-party computation (MPC) and zero knowledge (ZK) proofs have been introduced in recent decades and improved in countless works.

While these PETs have the potential to solve the privacy problems in many real world use cases where sensitive datasets are combined, they usually suffer from high performance penalties and being difficult to efficiently apply to use cases without expert knowledge. Thus, academic research focuses on improving the performance of common building blocks of PETs while also proposing concrete protocols which can directly be used to solve real world problems.

One of these common building blocks are symmetric primitives, such as ciphers and hash functions. These are not only directly used in many applications, such as anonymous credentials or private transactions, they can also be used to speed up any application by drastically reducing communication complexity which limits performance in HE and MPC applications, especially when slow network connections are involved. Unfortunately though, different PETs come with their own cost metrics which is drastically different from the traditional requirement of being efficient in software or hardware. Thus, well-established ciphers and hash functions, such as AES and SHA-3, cannot be used efficiently in combination with PETs.

In this thesis we continue research on efficient PET use cases and building blocks. Concretely, we propose protocols which have the potential to contribute to solving real world problems, such as combining health and location data efficiently to help the containment of the spread of infectious diseases while ensuring the privacy of all involved datasets. Furthermore, we propose new symmetric ciphers, dubbed Pasta and Hydra, optimized for fast encryption when used in combination with HE and MPC. Then we propose the new hash function Monolith which is especially suited for fast hashing in common ZK use cases. Finally, we also investigate alternative solutions to replace Merkle trees, a common building block in ZK applications, with novel MPC-based public key accumulators to gain more efficiency.

Kurzfassung

Kryptografie ist einer der grundlegenden Bausteine zur Sicherung von Daten in kritischen Online-Infrastrukturen wie Online-Banking oder Online-Behördendiensten. Herkömmliche kryptografische Primitive können jedoch nur für den Schutz von Daten im Ruhezustand und bei der Übertragung verwendet werden, während Datenschutzvorschriften wie die Datenschutz-Grundverordnung (DSGVO) und die zunehmende Auslagerung von Berechnungen auch den Schutz von Daten während der Nutzung erfordern. Zu diesem Zweck wurden in den letzten Jahrzehnten Technologien zur Verbesserung des Schutzes der Privatsphäre (Privacy Enhancing Technologies, PETs) wie homomorphe Verschlüsselung (Homomorphic Encryption, HE), sichere Mehrparteienberechnung (secure Multi-party Computation, MPC) und Zero Knowledge (ZK) Proofs eingeführt und laufend verbessert.

Während diese PETs das Potenzial haben die Datenschutzprobleme in vielen realen Anwendungsfällen, in denen sensible Datensätze kombiniert werden, zu lösen, leiden sie in der Regel unter hohen Einbußungen in Performanz und der Schwierigkeit, sie effizient auf Anwendungsfälle ohne Expertenwissen anzuwenden. Daher konzentriert sich die akademische Forschung auf die Verbesserung der Leistung von PETs Bausteinen und schlägt gleichzeitig konkrete Protokolle vor, die direkt zur Lösung realer Probleme verwendet werden können.

Einige dieser gemeinsamen Bausteine sind symmetrische Primitive wie Chiffren und Hash-Funktionen. Diese werden nicht nur direkt in vielen Anwendungen verwendet, z.B. für anonyme Anmeldedaten oder private Transaktionen, sondern können auch zur Beschleunigung jeder Anwendung eingesetzt werden, indem sie die Kommunikationskomplexität, welche die Leistung von HE- und MPC-Anwendungen einschränkt, drastisch reduzieren, insbesondere wenn langsame Netzwerkverbindungen beteiligt sind. Unglücklicherweise haben verschiedene PETs ihre eigenen Metriken für die Performanz. Diese unterscheiden sich oft drastisch von der traditionellen Anforderung in Software oder Hardware effizient zu sein. Daher können etablierte Chiffren und Hash-Funktionen wie AES und SHA-3 oft nicht effizient mit PETs kombiniert werden.

In dieser Arbeit setzen wir die Forschung zu effizienten PET-Anwendungsfällen und -Bausteinen fort. Konkret schlagen wir Protokolle vor, die das Potenzial haben, dazu beizutragen reale Probleme zu lösen, wie z. B. die effiziente Kombination von Gesundheits- und Standortdaten, um die Ausbreitung von Infektionskrankheiten einzudämmen und gleichzeitig den Datenschutz für alle beteiligten Datensätze zu gewährleisten. Darüber hinaus schlagen wir neue symmetrische Chiffren mit den Bezeichnungen Pasta und Hydra vor, die in Kombination mit HE und MPC für eine schnelle Verschlüsselung optimiert sind. Dann schlagen

wir die neue Hash-Funktion Monolith vor, die sich besonders für schnelles Hashing in gängigen ZK-Anwendungsfällen eignet. Schließlich untersuchen wir auch alternative Lösungen, um Merkle-Bäume, einen häufigen Baustein von ZK-Anwendungen, durch neuartige MPC-basierte Public-Key-Akkumulatoren zu ersetzen, um mehr Effizienz zu erreichen.

Acknowledgements

Before I started with my master's thesis my goal was to go to industry directly after getting my degree. However, many talks with my supervisors convinced me to pursue a PhD in cryptography, a decision I now years later do not regret, despite the many ups and downs which admittedly sometimes were highly demanding, exhausting, and stressful. Thus, I especially want to thank my supervisor Christian, who not only gave me the opportunity for doing the PhD, but also supported me during every step and gave me the freedom to pursue many different subfields of cryptography which sparked my interest. Due to his guidance my partitioned research still not only resulted in a thesis where the topics nicely fit together (in my opinion), but also furthered the state-of-the-art in many highly practical and sought after domains. Furthermore, Christians support came without me having to engage in tedious applications for funding or engaging in many EU project meetings, for which I am especially thankful. I also want to thank Christian and Thomas Schneider for their valuable suggestions to improve this thesis and agreeing to being part of the PhD defense committee.

Many thanks also go to Daniel and Lukas, who not only coauthored many of my publications and shared an office with me during a considerable part of my PhD, but also continued to be valuable colleagues after founding the company TACEO – together with Christian, and Stefan. Furthermore, I want to thank all my coauthors, in particular Markus, Lorenzo, and Sebastian, whose discussions, helpful guidance, and sometimes the right amount of push helped in shaping me into the researcher I am now. I especially want to highlight Markus, who is responsible for shifting my focus from just using PETs to symmetric primitives for PET applications after about half of my PhD, but supported me in getting into this topic in time with many fruitful coffee breaks.

Finally, I want to express my gratitude to my parents and my friends, who supported me throughout my whole life, in particular during my time as a PhD student, despite me admittedly not always being the easiest, calm and relaxed person, especially during the many downs every PhD student faces during the studies.

Table of Contents

Part I

Background

1 Introduction

In the recent years increasingly more aspects of our day-to-day live have been shifted to the Internet. Online services, such as online banking, social media, streaming services, and home entertainment systems are already staples of modern society, while services such as digital governments, online meetings, shared online offices, and online document editing have found more widespread adaption due to significant improvements fueled by the recent Covid-19 pandemic. Furthermore, it has become the norm to outsource expensive computations, as well as hosting websites, to cloud services, such as Amazon AWS or Microsoft Azure. Cryptography is the fundamental building block behind the security of these online services, traditionally by using the TLS protocol to guarantee a secure connection between end points of the web.

However, with the increasing privacy awareness of companies and users, and the emergence of data protection laws such as the General Data Protection Regulation (GDPR) [Eur18] and the California Consumer Privacy Act (CCPA) [Bai20] simply securing data in transit using TLS while giving away all data to the service providers is no longer considered to be state-of-the-art of data protection. While traditional cryptographic protocols and primitives are designed to explicitly forbid modifying encrypted data by including tamper resistance into the designs, a new area of cryptography, focusing on private computations using *privacy enhancing technologies* (PETs), emerged in the recent decades. These PETs include new protocols and primitives, such as homomorphic encryption (HE), secure multi-party computations (MPC), and zero knowledge (ZK) proofs. While primitives such as HE allow to compute functions directly on ciphertexts without the requirement of decrypting them first or even knowing the secret decryption key, MPC protocols allow multiple mutually distrusting parties to compute functions on their combined private inputs without leaking them to each

other. Furthermore, ZK proofs allow to prove knowledge of secrets or assertions of statements, such as "I am included in this allowlist", or "I send 5 tokens to user X" without leaking anything beyond the validity of the statements.

The potential of PETs is vast and they have already successfully been used in private statistics and machine learning [GDL+16; JVC18; CJP21; WTB+21], private transactions and cryptocurrencies [OWW+20; Zca; Tor], anonymous credentials [CL02; NJ21], key management [KMO+21], private database joins [MRR20], private recommender systems [CCD+20], and many more. However, the application of these PETs to any use case is still a tedious task that requires domain experts to fine-tune the PET to get secure and efficient protocols. Furthermore, the resulting privacy enhanced use cases usually suffer from a severe performance penalty. On one hand these PETs are still expensive in terms of computational complexity and the overhead compared to the unprotected baseline can range up to several orders of magnitude. On the other hand, the privacy-preserving computations often require to exchange vast amounts of randomized data between the involved parties which can reduce use case performance significantly, especially when slow internet connections are involved. Consequently, research of this modern cryptographic protocols and primitives heavily focuses on making them and their building blocks faster while also simplifying their application to use cases and keeping their security guarantees.

In this thesis, we aim to continue this line of research, by improving performance of PET applications in several different ways. On one hand we apply PETs directly on use cases to efficiently instantiate new protocols with high security and privacy guarantees but still make them usable in practice. Concretely, we propose a protocol which combines health and location data without leaking them to any party while being able to process 8 million subscribers in 1 hour on reasonable hardware. While acknowledging that such an approach needs to be accompanied with various non-technical considerations, it has the potential to contribute to helping in containing the spread of infectious diseases. On the other hand, we propose new and more efficient building blocks which are then used to improve performance in common PET use cases. One of these building blocks is the use of symmetric ciphers and hash functions. While traditional and standardized ciphers and hashes, such as the commonly used AES [DR00; DR02] or SHA-3 [Nat15; BDP+11], could be used, they were not designed for the cost metrics encountered within the PETs, leading to inefficient constructions. Thus, we continue the line of work where new symmetric primitives are proposed for efficient usage within PET use cases and introduce new ciphers optimized for MPC and HE use cases, as well as a hash function which is especially useful in ZK applications.

1.1 Organization

This dissertation is written as a cumulative thesis that consists of two parts. In Part I we give a high-level introduction to the topics covered by this thesis followed by a discussion of the contributions of the thesis. Concretely, Chapter 2 introduces

the privacy enhancing technologies homomorphic encryption (Section 2.2), secure multi-party computation (Section 2.3) and zero knowledge proofs (Section 2.4). Then, in Chapter 3, we discuss the role and design of symmetric primitives in combination with PETs. First, we discuss the use cases where symmetric primitives are used in Section 3.1, then we discuss the history and state-of-the-art of designing efficient symmetric primitives for these PETs in Section 3.2. Finally, in Chapter 4, we summarize the publications included in this thesis.

The second part of the thesis, Part II, consists of the scientific contributions of the author where the full publication is appended to each chapter. Concretely, in Chapter 5 we introduce a practical protocol designed to combine location data with health data for disease analysis use cases while ensuring the privacy of all involved datasets. In Chapter 6 we investigate hybrid homomorphic encryption and introduce the stream cipher Pasta. Then in Chapter 7, we introduce the Megafono design strategy and the PRF Hydra which is optimized for fast encryption in MPC use cases. Chapter 8 then introduces the hash function Monolith which is as fast as SHA-3 for hashing while outperforming other arithmetization-friendly hash functions when used in zero knowledge use cases. Finally, in Chapter 9 we investigate alternative approaches to constructing efficient accumulators for zero knowledge use cases and introduce MPC public key accumulators, concretely an MPC-q-SDH accumulator.

2

Computations on Private Data

In this thesis, we work on multiple aspects and areas of modern cryptographic protocols and primitives, including (fully) homomorphic encryption (HE), secure multi-party computation (MPC), and zero knowledge (ZK) proofs. Thus we introduce these three privacy enhancing technologies (PETs) in more detail in this chapter.

2.1 Notation

Throughout this chapter we use the following notations. We denote a vector as $\vec{x}$. Further, we write $\texttt{rot}_i(\vec{x})$ to indicate a rotation of the vector $\vec{x}$ by i steps to the left. With $x \leftarrow \chi$ we denote sampling a random value x from the distribution χ, and with $x \leftarrow U(\mathbb{Z}_q)$ and $y \leftarrow U(R_q)$ we denote sampling a value uniformly at random from $\mathbb{Z}_q$ and R_q, respectively. With $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$, and $\lfloor \cdot \rceil$ we denote rounding down, up, and to the nearest integer, respectively. We use the notation $[\![x]\!] = (x_1, x_2, ..., x_n)$ to denote a secret sharing of the value x amongst n parties, such that party i has the share x_i. With $\langle x \rangle = ([\![x]\!], [\![\gamma_x]\!])$ we denote an authenticated sharing of the value x with $\gamma_x = \alpha \cdot x$ being the authentication MAC under the MAC key α.

2.2 (Fully) Homomorphic Encryption

Homomorphic encryption (HE) allows to perform computations on ciphertexts without having to decrypt the data or having to know the secret decryption key. Thus, HE has the potential to solve many real world problems where multiple datasets need to be combined while preserving privacy and has often

been labeled as the holy grail of cryptography [Mic10]. The concept of HE was already introduced in 1978 [RAD78], however, early schemes were only *partially homomorphic*, e.g., the RSA cryptosystem [RSA78] only allows to perform multiplications on encrypted data, while Paillier's scheme [Pai99] is homomorphic only for addition. Only after Gentry's groundbreaking dissertation [Gen09] in 2009 we were capable of constructing a *fully homomorphic encryption* (FHE) scheme, a scheme which at least in theory allows to perform an arbitrary computation on encrypted data. The theoretical requirement for a scheme to be fully homomorphic is that it must be capable of performing both addition and multiplications on ciphertexts an arbitrary number of times. Gentry achieved this by constructing his cryptosystem from noisy lattice-based ciphertexts which allow both homomorphic addition and multiplication to be performed a limited number of times and by the introduction of the novel bootstrapping procedure which, informally spoken, removes the noise such that further homomorphic operations can be performed.

Gentry's original construction was too impractical to ever be used, but many followup publications improved upon his original blueprint and led the way to more efficient cryptosystem, such as BFV [Bra12; FV12], BGV [BGV14], CKKS [CKK+17], and TFHE [CGG+20].

2.2.1 (R)LWE Based Homomorphic Encryption

The security of modern HE schemes is based on the *learning with errors* (LWE) [Reg05] hardness assumption and its optimized variant over polynomial rings Ring-LWE (RLWE) [LPR10]. Since the majority of HE schemes we use in this thesis are based on RLWE we restate this hardness assumption in Definition 1, more concretely the hardness of the RLWE decision problem.

Definition 1 (RLWE [LPR10])**.** *Given a dimension $N = 2^n$, a modulus q, and a error distribution χ over a ring $R_q = \mathbb{Z}_q[X]/(X^N+1)$, the RLWE distribution with secret $s \in R_q$ is given as $(b, a) \in R_q^2$ where $a \leftarrow U(R_q)$, $e \leftarrow \chi$ and $b = a \cdot s + e$. The (decisional) RLWE assumption of parameter (N, q, χ) is that it is computationally infeasible to distinguish between a sample drawn from the RLWE distribution and a sample drawn uniformly at random from $(b, a) \leftarrow U(R_q^2)$.*

Remark 1. *In general one can define the RLWE hardness assumption on arbitrary cyclotomic rings $R_q = \mathbb{Z}_q[X]/\Phi_m(X)$, where $\Phi_m(X)$ is the m-th cyclotomic polynomial. However, in this thesis we limit ourselves to rings modulo polynomials $\Phi_{2n}(X) = X^n + 1$ due to their efficiency and practical relevance. Indeed, these rings allow us to use the negacyclic number theoretic transform (NTT) to speed up multiplications, they are recommended by the homomorphic encryption standardization project,[1] and they are used in common libraries and schemes, such as the SEAL library [Sea] and the TFHE scheme [CGG+20].*

Most (R)LWE based HE schemes share the same basic structure to construct a public key encryption scheme [Reg05], where the (R)LWE secret s serves as

[1] https://homomorphicencryption.org/

the private key of the cryptosystem and the ciphertexts are the messages which are blinded by (R)LWE samples. A public key can be created as an encryption of 0, in other words, the public key is another (R)LWE sample. As example, we give the public key encryption scheme as used for BGV in Definition 2.

Definition 2 (BGV public key encryption scheme [BGV14])**.** *The BGV public key encryption scheme, parameterized by the plaintext modulus p, a secret key distribution ψ and the RLWE parameters (N, q, χ) is a tuple (*KGen, Enc, Dec*) of PPT algorithms, which are defined as follows:*

- $\mathsf{KGen}()$*: Sample $s \leftarrow \psi$, $a \leftarrow U(R_q)$, and $e \leftarrow \chi$, set $b = -(a \cdot s + p \cdot e)$. Return $\mathsf{sk} = s$ and $\mathsf{pk} = (b, a)$.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: Given a message $m \in R_p$ and the key $\mathsf{pk} = (b, a) \in R_q^2$, sample $v \leftarrow U(R_2)$ and $e_0, e_1 \leftarrow \chi$, and set $c_0 = v \cdot b + p \cdot e_0 + m$ and $c_1 = v \cdot a + p \cdot e_1$. Return the ciphertext $\mathsf{ct} = (c_0, c_1)$.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Given a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_q^2$, and the key $\mathsf{sk} = s \in R_q$. Calculate the partial decryption $m' = (c_0 + s \cdot c_1)$ and return $m \in R_p$ where $m = m' \mod p$.*

During encryption of an (R)LWE based scheme random noise (e, e_0, and e_1 in Definition 2) is introduced into the ciphertext. This noise grows with every homomorphic operation, by a small amount for a homomorphic addition, but a large amount for homomorphic multiplications. The noise is removed from the ciphertext during decryption. However, if the noise becomes too large and exceeds a certain threshold the plaintext can not be recovered anymore and decryption fails. These schemes, i.e., schemes capable of computing circuits up to a specific multiplicative depth, are called *somewhat* or *leveled homomorphic encryption* (SHE) schemes. Following Gentry's blueprint, (bootstrappable) SHE schemes can be transformed into FHE schemes by applying the *bootstrapping* procedure. This procedure aims to reset the noise in the ciphertexts by homomorphically evaluating the decryption circuit of the SHE scheme. However, for many schemes the bootstrapping procedure is still very expensive to compute and it is often more practical avoiding it altogether and using and SHE scheme with parameters chosen such that the scheme supports the evaluation of a given circuit.

At the time of writing two dominant flavours of how to construct modern HE schemes exist in the literature which we discuss in the next sections. The first one is focused on efficient RLWE-based SHE construction (Section 2.2.2) while the second one is optimized for fast bootstrapping of LWE-based schemes (Section 2.2.3).

2.2.2 Efficient RLWE-Based SHE Schemes

BGV [BGV14], BFV [Bra12; FV12], and CKKS [CKK+17] are HE schemes which follow the same principles on how to construct a HE scheme from the basic RLWE public key encryption scheme (e.g., see Definition 2). The main difference between these schemes is the way in which the noise is embedded in

the ciphertexts during encryption, and how the noise is removed after the partial decryption $m' = c_0 + s \cdot c_1$. While in BGV the message is embedded in the least significant bits of the ciphertext and the noise is scaled with p such that the noise can be removed by reducing mod p (i.e., $b = a \cdot s + p \cdot e + m$), the BFV scheme goes the other direction and scales the message during encryption (i.e., $b = a \cdot s + \lfloor q/p \rfloor \cdot m + e$). From the perspective of a HE practitioner, this difference is barely visible. Indeed, both schemes are used to perform homomorphic encryption over messages in $\mathbb{Z}_p$. On the other hand, CKKS handles the noise differently.

CKKS was introduced to allow efficient homomorphic operations for real numbers $\mathbb{R}$. A simple technique to mimic HE over $\mathbb{R}$ is to translate a messages $m \in \mathbb{R}$ into a fixed-point representation $m^\star = \lfloor \Delta \cdot m \rfloor$ with scaling factor Δ and using BFV/BGV. However, besides the large noise growth in any (R)LWE encryption, a multiplication now also grows the scale Δ to Δ^2. Thus, CKKS introduced a novel homomorphic rescaling operation, which can be used to restore the original scale Δ after a multiplication. Furthermore, CKKS embeds the noise directly in the least significant bits of the scaled messages (i.e., $b = a \cdot s + \lfloor \Delta \cdot m \rfloor + e$) to additionally reduce the noise in the ciphertexts during the rescaling operations. However, since rescaling reduces the ciphertext modulus q by a factor Δ, the size of q limits the number of rescaling operations (consequently multiplications) before a ciphertext has to be bootstrapped.

From Public Key Encryption to HE Scheme. While BGV, BFV, and CKKS differ in the way the noise is embedded, they follow the same blueprint on how to extend the public key encryption scheme into a HE scheme. First, observe that a homomorphic addition comes naturally in the (R)LWE based encryption schemes:

$$m'_1 + m'_2 = (c_{1,0} + s \cdot c_{1,1}) + (c_{2,0} + s \cdot c_{2,1}) = ((c_{1,0} + c_{2,0}) + s \cdot (c_{1,1} + c_{2,1})) \,.$$

Hence, the sum of two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2 \in R_q^2$ is a valid encryption of the sum of the plaintexts $m_1, m_2 \in R_p$. Multiplications, on the other hand, are not so straightforward. First, observe:

$$\begin{aligned} m'_1 \cdot m'_2 &= (c_{1,0} + s \cdot c_{1,1}) \cdot (c_{2,0} + s \cdot c_{2,1}) \\ &= (c_{1,0} \cdot c_{2,0}) + s \cdot (c_{1,0} \cdot c_{2,1} + c_{1,1} \cdot c_{2,0}) + s^2 \cdot (c_{1,1} \cdot c_{2,1}) \\ &= c'_0 + s \cdot c'_1 + s^2 \cdot c'_2. \end{aligned}$$

Consequently, a multiplication produces a ciphertext $\mathsf{ct}' \in R_q^3$ and requires the keys s and s^2 for decryption. To prevent the degrees of the ciphertexts from growing after each multiplication, the schemes use a *relinearization* step based on *key switching*. Key switching is a well known component of modern HE schemes which allows to transform a ciphertext ct decryptable using the key s into a ciphertext ct' decryptable using the key s'. This step requires a key switching key $\mathsf{KS}_{s \to s'}$ which needs to be created by the party knowing the keys s and s'. We recall a well known key switching procedure, as described in e.g., [CDK+21], in Definition 3.

Definition 3 (Key Switching)**.** *Given a gadget decomposition vector [MP12]* $\vec{g} = (g_0, \ldots, g_{d-1})$ *such that the map* $g^{-1} : \mathbb{Z}_q \to \mathbb{Z}^d$ *satisfies* $\langle g^{-1}(a), \vec{g} \rangle \mod q = a$ *for all* $a \in \mathbb{Z}_q$*. Further, given the natural extension of this map to* R_q*, i.e.,* $g^{-1} : R_q \to R^d$ *such that* $a = \sum_i a_i X^i \mapsto \sum_i g^{-1}(a_i) X^i$ *for all* $a \in R_q$*, the key switching procedure is defined by a key switching key generation algorithm* KSKGen *and a key switching algorithm* $\mathsf{KeySwitch}$*:*

- $\mathsf{KSKGen}(s, s')$*: Sample* $\vec{k}_1 \leftarrow U(R_q^d)$*, and* $\vec{e} \leftarrow \chi^d$*, and set* $\vec{k}_0 = -s' \cdot \vec{k}_1 + s \cdot \vec{g} + \vec{e}$*. Return the key* $\mathsf{KS}_{s \to s'} = [\vec{k}_0 | \vec{k}_1] \in R_q^{d \times 2}$*.*
- $\mathsf{KeySwitch}(\mathsf{KS}_{s \to s'}, \mathsf{ct})$*: Given the ciphertext* $\mathsf{ct} = (c_0, c_1) \in R_q^2$ *and a key* $\mathsf{KS}_{s \to s'} \in R_q^{d \times 2}$ *calculate* $\mathsf{ct}' = (c_0, 0) + g^{-1}(c_1) \cdot \mathsf{KS}_{s \to s'}$ *and return* ct'*.*

Using key switching with the key $\mathsf{KS}_{s^2 \to s}$, which is sometimes referred to as *relinearization key*, relinearization of a ciphertext $\mathsf{ct}' = (c'_0, c'_1, c'_2)$ can be performed by setting

$$\mathsf{ct} = (c_0, c_1) = (c'_0, c'_1) + \mathsf{KeySwitch}(\mathsf{KS}_{s^2 \to s}, (0, c'_2)). \tag{2.1}$$

Packing. RLWE based HE schemes, such as BFV, BGV, and CKKS operate on polynomial rings $R_q = \mathbb{Z}_q[X]/(X^N + 1)$ implying that plaintexts are also polynomials $\mathsf{pt} \in R_p$. Homomorphic operations, thus, correspond to polynomial arithmetic in R_p as well. However, practical use cases such as data analysis or machine learning rarely use polynomial arithmetic; integers and real numbers are significantly more common. Luckily, one can use the *discrete fourier transform* (DFT) to encode a vector $\vec{a} \in \mathbb{F}_p^N$ into a polynomial $a' \in \mathbb{R}_p = \mathbb{F}_p[X]/(X^N + 1)$ such that polynomial addition and multiplication corresponds to elementwise vector addition and multiplications [SV14]. This process is usually referred to as *packing*. In other words, using packing one can perform homomorphic additions and multiplications on N integers in parallel. Furthermore, one can use Galois automorphisms $\tau_i : a(X) \mapsto a(X^i)$ to permute the encoded vector. Concretely, for $R_p = \mathbb{F}_p[X]/(X^N + 1)$ the Galois automorphisms correspond to rotating or swapping two vectors $\vec{a}_L, \vec{a}_R \in \mathbb{F}_p^{N/2}$ simultaneously:

$$\begin{bmatrix} \vec{a}_L \\ \vec{a}_R \end{bmatrix} \overset{\text{encode}}{\to} a \in R_p : \qquad \tau_{3^i}(a) \overset{\text{decode}}{\to} \begin{bmatrix} \mathtt{rot}_i(\vec{a}_L) \\ \mathtt{rot}_i(\vec{a}_R) \end{bmatrix}, \qquad \tau_{N-1}(a) \overset{\text{decode}}{\to} \begin{bmatrix} \vec{a}_R \\ \vec{a}_L \end{bmatrix}.$$

Due to the homomorphic nature of Galois automorphisms, one can also apply them on the encrypted polynomials:

$$\tau_i(m') = \tau_i(c_0 + s \cdot c_1) = \tau_i(c_0) + \tau_i(s) \cdot \tau_i(c_1).$$

Consequently, using key switching (Definition 3) homomorphic automorphisms can be computed as

$$\mathsf{ct}' = \mathsf{KeySwitch}\left(\mathsf{KS}_{\tau_i(s) \to s}, (\tau_i(c_0), \tau_i(c_1))\right),$$

where $\mathsf{Dec}(s, \mathsf{ct}) = \mathsf{Dec}(s, (c_0, c_1)) = m$, $\mathsf{Dec}(s, \mathsf{ct}') = \tau_i(m)$, and $\mathsf{KS}_{\tau_i(s) \to s}$ is a key switching key created by the party knowing s. Thus, a homomorphic Galois

automorphism requires a key switching key $\mathsf{KS}_{\tau_i(s)\to s}$ (sometimes referred to as *Galois key*) for each index i.

Many practical use cases leverage packing and rotations to decrease latency by using dedicated algorithms, such as the babystep-giantstep optimized diagonal method [HS14; HS15; HS18] for matrix-vector multiplications. In this thesis, we describe and use this algorithm in Chapter 5 and Chapter 6.

Cost Metric. The main cost metric of SHE schemes is the multiplicative depth of the circuit. Supporting a larger multiplicative depth requires to instantiate the SHE scheme with a large ciphertext modulus q, which in turn requires a larger polynomial degree $N = 2^n$ for security. Thus, larger depths quadratically increase the cost of homomorphic computations. Given a fixed parameter set (N, q), the most costly homomorphic operation is a ciphertext-ciphertext multiplication, followed by the Galois automorphisms due to the requirement of key switching.

2.2.3 Fast LWE-Based Bootstrapping Schemes

Another method to extend the basic (R)LWE public key encryption scheme (see Definition 2) to HE schemes resulted in schemes which are optimized for fast bootstrapping. In this section we will only shortly introduce these schemes, since we mainly use SHE schemes (see Section 2.2.2) in this thesis.

The first step into this branch of HE schemes was GSW [GSW13], a novel scheme which removed the relinearization-based homomorphic multiplications of previous HE schemes (see Equation (2.1)) in favour of an approach based on matrices and eigenvectors. The authors realized that given two ciphertext matrices $C_1, C_2 \in \mathbb{Z}_q^{n\times n}$ encrypting μ_1, μ_2 such that $C_i \cdot \vec{s} = \mu_i \cdot \vec{s} + \vec{e}_i$ (i.e., $\vec{s}$ is an *approximate* eigenvector with eigenvalue μ_i), the product matrix $C = C_1 \cdot C_2$ satisfies $C \cdot \vec{s} = \mu_1\mu_2 \cdot \vec{s} + \vec{e}$. Furthermore, $\vec{e}$ will be a small noise vector if $C_1, \mu_2, \vec{e}_1$ and $\vec{e}_2$ are small. This more natural homomorphic multiplication approach compared to relinearization has further been improved in FHEW [DM15] and TFHE [CGG+16]. We will describe TFHE in the following.

TFHE. In TFHE (Torus-FHE), the authors redefine the (R)LWE hardness assumption over the real Torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, i.e., the real numbers modulo 1, instead of integers in $\mathbb{Z}_q$. Furthermore, they define ciphertexts using the LWE and RLWE hardness assumption over $\mathbb{T}$ and define GSW and Ring-GSW ciphertexts accordingly over $\mathbb{T}$ as well. They call the resulting ciphertexts TLWE, TRLWE, TGSW, and TRGSW respectively. Then they introduce the external product $\boxdot$ derived from the GSW multiplication which serves as the basis for many of their optimizations:

$$\boxdot : \text{TRGSW} \times \text{TRLWE} \to \text{TRLWE}.$$

Contrary to the SHE schemes from Section 2.2.2, ciphertexts in TFHE are not using polynomial rings, but are based on the LWE hardness assumption. The reason for that is that RLWE ciphertexts can be seen as multiple LWE ciphertexts and TFHE bootstrapping requires extracting an LWE ciphertext

from a prepared RLWE one. More concretely, a TLWE ciphertext $(\vec{a}, b) \in \mathbb{T}^{n+1}$ encrypting a message $\mu \in \mathbb{T}$, such that $\langle \vec{a}, \vec{s} \rangle - b = \mu + e$ for a decryption key $\vec{s} \in \mathbb{Z}_2^n$ and the noise $e \in \mathbb{T}$, can be bootstrapped by a sequence of algorithms:

$$\text{Bootstrap}(\cdot) = \text{KeySwitch} \circ \text{Extract} \circ \text{BlindRotate}(\cdot).$$

The most important step in the bootstrapping algorithm is the BlindRotate algorithm. In this step a noiseless trivial TRLWE ciphertext $v' = (\vec{0}, v)$ is prepared, where the i-th coefficient of v is chosen to be the desired plaintext if $\mu + e = -i$. Then, using the external product $\boxdot$ and TRGSW encryptions of each bit of the decryption key $\vec{s}$ as a bootstrapping key, one can repeatedly multiply v' by $X^{a_i \cdot s_i}$ to rotate the coefficients of v. Finally, after all a_i are processed, one multiplies the result with x^{-b} to complete evaluating the decryption circuit $\langle \vec{a}, \vec{s} \rangle - b = \mu + e$. Consequently, the evaluation corresponds to multiplying v by $X^{\mu+e}$ and the desired plaintext corresponding to μ is located in the constant term of the polynomial encrypted in the resulting RLWE ciphertext. The remaining step is an extraction of this constant term as an LWE ciphertext and keyswitching it back to an encryption under $\vec{s}$.

This bootstrapping procedure allows to refresh a ciphertext in less than $0.1\,s$ [CGG+16]. Due to its high efficiency it is usually performed after each multiplication, thus the main cost metric is the number of multiplications in the evaluated circuit. The main drawback of TFHE compared to SHE schemes (e.g., BGV, BFV and CKKS), however, is that ciphertexts are not RLWE ciphertexts, but LWE ones. Consequently, TFHE does not support packing for efficient parallel computations, which is why using SHE schemes is the preferred choice for evaluating highly parallelizable low-depth circuits. On the other hand TFHE is the better choice for large-depth circuits due to its fast bootstrapping procedure.

Since TFHE was introduced in [CGG+16], it has been drastically improved with new features. While initially only plaintexts in $\mathbb{Z}_2$ were supported, modern versions allow for larger plaintexts $\in \mathbb{Z}_{2^q}$ for small values $q \leq 8$.[1] Furthermore, the bootstrapping procedure has been extended to *programmable bootstrapping* [CJP21], where a small negacyclic lookup table is evaluated at no extra cost during bootstrapping by preparing the polynomial v accordingly.

2.3 Secure Multi-party Computation

Secure Multi-party Computation (MPC) allows several mutually distrusting parties to compute a function on their combined private inputs. Thereby, all or a subset of the parties get the correct result without learning anything about the other parties' input data or intermediate results beyond what can be inferred from the output of the protocol. MPC protocols were first introduced in 1982 [Yao82] by Yao and applied to solve the now famous *Millionaires Problem* where two millionaires want to find out who is richer without telling each other their net worth. In the beginning, research on MPC was only of theoretical interest, however,

[1] https://docs.zama.ai/tfhe-rs/getting-started/operations

starting with *Fairplay* [MNP+04] in 2004 first practical applications emerged. Since then, research on MPC protocols and applications rapidly increased and it is nowadays an established field of research with regular publications at top conferences.

While the general concept of MPC is to compute a function $y = f(x_1, x_2, \ldots x_n)$ on the private inputs x_i of party P_i without leaking x_i to other parties, multiple concrete instantiations and protocols differ drastically in how they are constructed, the security models, overall number of supported computing parties, and the total number of tolerated malicious parties. We discuss some security models and concrete protocols in this section.

2.3.1 Security Models

MPC protocols are usually proven secure in the real-ideal world paradigm [GM84] against different adversary models. The main idea behind this paradigm is to investigate and compare the protocol when executed in two different settings, an *ideal world* with the help of a trusted third party, and the *real world* with the real protocol execution.

Ideal World. First, in the ideal world, we define an *ideal functionality* $\mathcal{F}$ and model the protocol as an interaction between the parties P_i with private input x_i and the functionality $\mathcal{F}$. In this model, $\mathcal{F}$ behaves like a trusted third party that gets the inputs x_i from all n parties, computes the function $\mathcal{F}(x_1, x_2, \ldots, x_n)$, and returns the results to the parties. An adversary in this model can only take control over the parties P_i, but not the ideal functionality $\mathcal{F}$ itself. Thus, the adversary only learns the inputs of the parties P_i which are under its control, as well as the output of the computation. These inputs, thereby, are independent to the inputs of the honest parties, while the outputs are consistent for all parties and can be modelled to only be legal for valid inputs.

Real World. In the real world the main goal of MPC protocols is to get secure computation without the presence of a trusted third party. Thus, in the real world we define concrete protocols π used for communication between the parties. These protocols specify a function π_i for each party P_i which produces the message to be sent next based on the security parameter, the private input of party P_i, and the transcript of the messages received by party P_i so far. π_i then either outputs a message with its destination, or instructs the party to terminate or abort. In the real world, parties can be corrupted by adversaries with different capabilities depending on the concrete security models which we discuss next in this section. To argue security, one has to show that everything an adversary can achieve in the real world, can also be achieved in the ideal world. Thus, one can conclude that the real world provides the same security guarantees as the ideal world, given the capabilities of the adversary in the concrete security model.

Semi-honest Security Model. In the semi-honest security model an adversary corrupts one or many parties, such that they follow the protocol honestly, but tries to learn as much as possible from the combined transcript. Thus, the adversaries' *view*, i.e., all the information it sees, consists of all the private inputs, received messages and outputs of the corrupted parties. Since the adversary only observes messages while following the protocol honestly it is often called *passive* or *honest-but-curious* in this model.

An adversary can not deviate from the protocol in this model to influence messages from other parties, thus every attack can just use the adversaries' view (consisting of its inputs, random tape, and messages received from other parties) as its input. To prove security in the real-ideal world paradigm we have to show that every attack in the real world is also applicable in the ideal world. Hence, we have to provide a simulator in the ideal world, that is capable of producing a view of the corrupted parties that is indistinguishable from the adversaries' view in the real world. Since the simulator operates in the ideal world, it can only use messages exchanged between the corrupted parties and $\mathcal{F}$. Consequently, if a simulator exists that produces a view indistinguishable from the adversaries' view in the real world, we have shown that the adversary gains no information from the messages received from the other parties during protocol execution and the protocol π is secure in the semi-honest security model. We give the formal definition from [EKR18] in Definition 4.

Definition 4 (Semi-honest Security [EKR18])**.** *A protocol π securely realizes $\mathcal{F}$ in the presence of semi-honest adversaries if there exists a simulator $\mathtt{Sim}$ such that, for every subset of corrupt parties C and all inputs $x_1, \ldots, x_n$, the distributions $\mathtt{Real}_\pi(\kappa, C; x_1, \ldots, x_n)$ and $\mathtt{Ideal}_{\mathcal{F},\mathtt{Sim}}(\kappa, C; x_1, \ldots, x_n)$ are indistinguishable in the security parameter κ.*

Malicious Security Model. In this model an attacker is additionally capable of arbitrarily deviating from the protocol. Thus, this adversary is often referred to as *active*, since it can try to infer more information by transmitting malicious messages or purposefully forcing false outputs of the computation. To prove security in this model, we again compare the distribution of a simulator from the ideal world with the view of an adversary in the real world. However, in the malicious setting the adversaries' inputs to the real world are not well defined. Since in a secure protocol everything that can be achieved in the real world should also be achievable in the ideal world, we let the simulator choose the inputs of the adversary. Thus the simulator *extracts* an ideal-world input from an adversary that explains the effect of the input in the real world. This process is called *extraction*. We give the formal definition from [EKR18] in Definition 5.

Definition 5 (Malicious Security [EKR18])**.** *A protocol π securely realizes $\mathcal{F}$ in the presence of malicious adversaries if for every real-world adversary $\mathcal{A}$ there exists a simulator $\mathtt{Sim}$ with $\mathtt{corrupt}(\mathcal{A}) = \mathtt{corrupt}(\mathtt{Sim})$ such that, for all inputs for honest parties $\{x_i | i \notin \mathtt{corrupt}(\mathcal{A})\}$, the distributions $\mathtt{Real}_{\pi,\mathcal{A}}(\kappa; \{x_i | i \notin \mathtt{corrupt}(\mathcal{A})\})$ and $\mathtt{Ideal}_{\mathcal{F},\mathtt{Sim}}(\kappa; \{x_i | i \notin \mathtt{corrupt}(\mathtt{Sim})\})$ are indistinguishable in the security parameter κ.*

We want to note that we only consider the honest parties' inputs in Definition 5. This is due to the fact that inputs in the real worlds would merely be suggestions since the adversaries can freely deviate from them, while the malicious inputs are chosen by the simulator in the ideal world.

Further Adversary Restrictions and Models. Besides just defining adversaries as either semi-honest or malicious, further limitations on the capabilities of adversaries and the security of the protocols exists. First, protocols differ in how many parties can be corrupted while still providing security. In *dishonest majority* protocols, such as SPDZ [DPS+12], the adversary is allowed to corrupt up to $n-1$ out of n parties, while in *honest majority* protocols, such as ABY3 [MR18], an adversary is limited to corrupting only strictly less than $n/2$ parties. Furthermore, some protocols are generic to allow up to a parameterizable *threshold* of corrupted parties, as in e.g., Shamir secret sharing [Sha79]. Secondly, maliciously secure protocols also differ in their behavior when the presence of an adversary is detected. Many protocols (e.g., SPDZ) provide security *with abort*, i.e., parties detecting malicious behavior abort the computation. An extension is security *with identifiable abort* (e.g., the threshold signature scheme FROST2 [CKM21]), where the parties which deviate from the protocol are identified. Protocols providing *fairness* guarantee that if one honest party gets an output, all other honest parties get the same output as well. Furthermore, protocols achieving *guaranteed output delivery* (e.g., SWIFT [KPP+21]) guarantee that honest parties will always receive the correct result, even when corrupt parties try to falsify computations by sending malicious messages. Finally, most protocols consider *static corruptions*, i.e., the corrupt parties are fixed during the entire protocol execution. Adversaries capable of *adaptive corruption*, however, can corrupt specific parties based on information learned during the interaction with other parties.

Composability. We briefly want to discuss the problem of composability in the real-ideal world paradigm in this paragraph. In practice, a protocol execution can consist of multiple ideal functionalities. Thus, the question remains whether a protocol that securely realizes the ideal functionality $\mathcal{F}$ can be used as a subprotocol to realize the ideal functionality $\mathcal{G}$. Surprisingly, the real-world paradigm does not guarantee composability. However, an extension called the *universal composability* (UC) framework [Can01] achieves composability by adding the *environment* into the real-ideal world paradigm. It is included in both the real and the ideal world and should not be able to determine in which of both worlds it is instantiated. Thereby, the environment is responsible for choosing the inputs of the honest parties, to interact with the adversary and the simulator, and receives the output of the protocol. We give the formal definition of UC security, as given in [EKR18], in Definition 6.

Definition 6 (UC Security [EKR18])**.** *A protocol π UC-securely realizes $\mathcal{F}$ if for all real-world adversaries $\mathcal{A}$ there exists a simulator $\mathcal{S}im$ with $\mathtt{corrupt}(\mathcal{A}) =$*

corrupt(*Sim*) *such that, for all environments* $\mathcal{Z}$*:*

$$|Pr[Real_{\pi,\mathcal{A},\mathcal{Z}}(\kappa) = 1] - Pr[Ideal_{\mathcal{F},Sim,\mathcal{Z}}(\kappa) = 1]|$$

is negligible in the security parameter κ*.*

The main difference between the real-ideal world paradigm and the UC framework is the power of the simulator. Without the environment, the simulator can arbitrarily interact with the adversary, including rewinding it to previous internal states. In the UC model, on the other hand, the simulator must be a *straight-line simulator* that generates the response to queries from the environment in one pass and it must reply immediately.

All the MPC protocols we use in this thesis are proven to be secure in the UC framework. Furthermore, we prove UC security of the privacy-preserving heatmap protocol (see Chapter 5) and the MPC-q-SDH accumulator (Chapter 9) we propose in this thesis.

2.3.2 MPC Protocols

In this thesis we are mainly working with secret sharing based MPC protocols, such as SPDZ [DPS+12] and one based on Shamir secret sharing [Sha79]. We will introduce these protocols in this section.

Additive Secret Sharing. SPDZ is a protocol that builds upon *additive secret sharing*. For a set of n computing parties, a secret $x \in \mathbb{F}_p$ which should be shared is split by randomly sampling $x_i \leftarrow U(\mathbb{F}_p)$ for $i \in [1, n-1]$ and setting $x_n = x - \sum_{i=1}^{n-1} x_i$. Then, x_i is the share of party i for $i \in [1, n]$. To recompute the secret x from *all* shares $[\![x]\!] = \texttt{Share}(x) = (x_1, x_2, \dots x_n)$ the parties compute $x = \texttt{Open}([\![x]\!]) = \sum_{i=1}^{n} x_i$. Furthermore, parties can compute linear functions on their shares without communication:

$$\begin{aligned}[\![c]\!] &= \texttt{Share}(ax + by + z) = [\![a]\!] \cdot x + [\![b]\!] \cdot y + z \\ &= (a_1 \cdot x + b_1 \cdot y + z, a_2 \cdot x + b_2 \cdot y, \dots, a_n \cdot x + b_n \cdot y).\end{aligned}$$

SPDZ extends additive secret sharing to full MPC protocols, which are secure in the semi-honest and malicious security settings (with abort). First it extends the computational capacities of additive secret sharing with a multiplication protocol using Beaver's technique [Bea91].

Beaver Multiplication and Offline Phase. Given a precomputed Beaver triple $([\![a]\!], [\![b]\!], [\![ab]\!])$, one can compute the multiplication $[\![z]\!] = [\![x]\!] \cdot [\![y]\!]$ as

$$\begin{aligned}m &= \texttt{Open}([\![x]\!] + [\![a]\!]) \\ n &= \texttt{Open}([\![y]\!] + [\![b]\!]) \\ [\![z]\!] &= m \cdot n - m \cdot [\![b]\!] - n \cdot [\![a]\!] + [\![ab]\!]\end{aligned}$$

which requires one communication round to compute m and n. Then, using m, n, computing $[\![z]\!]$ just requires to compute linear functions on the shares. Beaver's trick, thus, reduces the multiplication of given shares $[\![x]\!]$ and $[\![y]\!]$ to the multiplication of *random* shares $[\![a]\!]$ and $[\![b]\!]$. This has the advantage that Beaver triples can be precomputed in a so-called *offline phase* before the inputs of the actual computation are known. Thus, SPDZ is split in to an input-independent offline phase where Beaver triples are pre-computed, and an *online phase* where the actual circuit is computed on the inputs using Beaver triples.

Initially, SPDZ was proposed to use homomorphic encryption to compute random Beaver triples in the offline phase [DPS+12]. Later on, the more efficient MASCOT [KOS16] offline phase was introduced which builds a multiplication protocol based on *oblivious transfer* (OT) [EGL82]. Finally, in Overdrive [KPR18] the homomorphic encryption-based offline phase got improved.

Share Authentication. So far, additive secret sharing and Beaver's multiplication trick only achieve security in the semi-honest setting. Thus, parties can cheat during the protocol execution without being detected. To this end, SPDZ introduces *share authentication* to achieve malicious security with abort.

In the offline phase, a global MAC key value $[\![\alpha]\!]$ is randomly sampled and shared amongst the parties, such that each party gets a share α_i, but no party knows the actual key α. Each share $[\![x]\!]$ is then extended with a MAC value $\gamma_x = \alpha \cdot x$ to an authenticated share $\langle x \rangle = ([\![x]\!], [\![\gamma_x]\!])$. The linear operations on the shares (and in extension the operations in Beaver's multiplication) are then extended with operations on $[\![\gamma_x]\!]$, such that the invariant $\gamma_x = \alpha \cdot x$ is always fulfilled. Consequently, the offline phase protocols [DPS+12; KOS16; KPR18] also need to produce authenticated Beaver triples $(\langle a \rangle, \langle b \rangle, \langle ab \rangle)$ to achieve malicious security (with abort).

Finally, in the output phase of the protocol, before the results are opened to the receiving parties, a MAC-check protocol is executed which checks whether the invariant $\gamma_x = \alpha \cdot x$ was fulfilled for all shares during execution. If yes, the parties get high assurance that the other parties honestly followed the protocol. Otherwise, at least one party was cheating during the execution and all honest parties abort.

SPDZ [DPS+12]. To summarize, SPDZ extends additive secret sharing with Beaver's multiplication, share authentication and splitting the computation into an input-independent offline phase and an input-dependent online phase. It achieves malicious security with abort in a dishonest majority setting. In other words, all shares are required for the computations and an honest party always detects malicious behavior with high probability. Furthermore, semi-honest versions of SPDZ can be instantiated by getting rid of share authentication (and consequently the MAC check) and cheaper offline phases without triple authentication.

Regarding performance, the offline phase in the SPDZ protocol is usually significantly slower compared to the online phase, requires to exchange more

amounts of data between the parties, and scales with the number of precomputed Beaver triples (see benchmarks in Chapter 7 and Chapter 9). Since one multiplication in the online phase requires communication between the parties, as well as a new Beaver triple, the number of multiplications in a circuit is usually regarded as the main cost metric for SPDZ. The multiplicative depth of the circuit, however, also plays a role and influences the online phase performance. Indeed, the openings in the online phase of Beaver's multiplication can be performed in parallel for all independent multiplications. Hence, the multiplicative depth directly corresponds to the number of communication rounds where parties have to exchange data over the network. As a consequence, the depth also has an impact on performance, especially in slow and high-delay network settings.

Shamir Secret Sharing [Sha79]. An alternative way of sharing secrets was introduced by Shamir in 1979. His approach is based on oversampling polynomials to produce a (t, n) *threshold secret sharing* scheme, where for n parties only t shares are required to reconstruct the secret.[1] Concretely, to share a value x a random polynomial $p[X]$ is sampled

$$p[X] = x + a_1 \cdot X + a_2 \cdot X^2 + \cdots + a_k \cdot X^k = x + \sum_{i=1}^{k} a_i X^i$$

where $k = t - 1$ and $a_i \leftarrow U(\mathbb{F}_p)$. The share x_i of party i such that $[\![x]\!] = \texttt{Share}(x) = (x_1, x_2, \ldots, x_n)$ is then computed as $x_i = p[i]$. Since polynomials of degree k can be recomputed from $k+1$ points using e.g., Lagrange interpolation, only t shares are required to recompute $p[X]$ and $x = p[0]$ in Shamir secret sharing. It, thus, adds resistance to parties dropping out during the MPC computation, but is only secure against $< t$ colluding parties.

Similar to additive secret sharing, linear functions can be computed on shares without the need of communication between the parties:

$$\begin{aligned}[\![c]\!] &= \texttt{Share}(ax + by + z) = [\![a]\!] \cdot x + [\![b]\!] \cdot y + z \\ &= (a_1 \cdot x + b_1 \cdot y + z, a_2 \cdot x + b_2 \cdot y + z, \ldots, a_n \cdot x + b_n \cdot y + z).\end{aligned}$$

However, since multiplication of two points $a_i = p_1[i]$ and $b_i = p_2[i]$ results in a point $c_i = p_3[i]$ where $p_3[0] = p_1[0] \cdot p_2[0]$ multiplication can be realized more easily compared to additive secret sharing. The drawback, however, is that the degree of $p_3[X]$ is $k = k_1 + k_2$ where k_1 and k_2 are the degrees of $p_1[X]$ and $p_2[X]$. Thus, for two shares with degree k, $2 \cdot k + 1$ parties are required for reconstruction after a multiplication. Hence, k is bounded by $k \leq \frac{n-1}{2}$. To reduce the degree after a multiplication, one can use two sharings of the random value r and compute $[\![r_1]\!] + \texttt{Open}([\![x]\!] - [\![r_2]\!])$, where $[\![r_1]\!]$ is a degree-k sharing of r and $[\![r_2]\!]$ is a degree-$2k$ sharing of r.

As a result, $2k + 1$ parties are required to participate in the degree reduction using this technique, while only $k + 1$ parties are required for reconstruction after

[1] Since all shares are required in additive secret sharing, we call it a (n, n) sharing scheme.

linear operations. Since $k \leq \frac{n-1}{2}$ is strictly less then half of the parties, a honest majority among the parties is required to ensure security. Alternatively, one can also use Beaver triples from an offline phase to achieve a larger threshold in the online phase.

To achieve malicious security (with abort), multiple options exist. As example, in [LN17] the authors propose dedicated verification protocols to check correctness of shares at the end of the computation, whereas in e.g., [CGH+18] and [CGG+21] authenticated shares and MAC-check protocols similar to SPDZ are used. We further want to note that, due to oversampling of polynomials, Shamir shares inherently provide some resistance against malicious parties and the correct secret can be reconstructed if enough parties followed the protocol honestly. A discussion of algorithms to reconstruct secrets in the presence of malicious shares can be found in e.g., [DGH12].

Given that in Shamir secret sharing linear functions can be directly computed on shares, while multiplications require communication between the parties, the number of multiplications in the evaluated circuit is usually regarded as the main cost metric. Similar to SPDZ, though, the multiplicative depth of the circuit also plays a role, especially in high-delay network settings.

Other Secret Sharing Protocols. Many other styles of secret sharing based MPC protocols exist in the literature, which we shortly discuss in this section. While SPDZ and protocols based on Shamir sharing are generic protocols which can be instantiated for any number of parties, dedicated protocols with a low fixed number of parties can achieve significantly better performance compared to the generic protocols. As example, in the honest-majority setting secret sharing protocols with *replicated sharing* are introduced in [AFL+16]. In these protocols each party does not only know its own share of the computation, but also shares of other parties. Concretely, for the 3-party setting, a share of a party consists of two additive shares of the secret. This has the advantage that linear functions can still be computed on the shares without communication (as inherited from additive sharing) while the multiplication of shares is also simplified. Let i be the party index, its share of $x = x_1 + x_2 + x_3$ is (x_i, x_{i+1}) (with $i+1$ being modular reduced to the set of parties). The equation

$$z_i = x_i y_i + x_i y_{i+1} + x_{i+1} y_i$$

produces a valid additive share z_i of $z = xy$. Hence, a multiplication can be built from a local computation and a re-sharing of the result consisting of sending the share (including a re-randomization by adding a fresh random value to prevent leaking intermediate results) to another party. The latter step transforms the additive share into a replicated share. Furthermore, malicious security can be achieved without SPDZ-style MACs by checking the consistency of multiplications [FLN+17]. This also leads to the crucial advantage that the protocols do not require to work over prime fields $\mathbb{F}_p$ for security (as is the case for SPDZ-style

MACs),[1] but can work over any ring $\mathbb{Z}_{2^k}$ leading to faster computations. In [MR18] this line of work has been extended with efficient subroutines to transform sharings over $\mathbb{Z}_{2^k}$ to binary sharings over $\mathbb{F}_2^k$ and back, while also introducing protocols to bridge to Yao's garbled circuit protocols [Yao86] (see next paragraph). Since then, replicated secret sharing protocols have been further improved in the literature [CCP+19; PS20; KPP+21; DEK21] and have become highly efficient 3 and 4 party honest-majority protocols achieving malicious security in different settings, such as abort, fairness, and guaranteed output delivery.

Garbled Circuit Protocols. We also want to shortly discuss garbled circuit protocols, such as the two-party Yao's garbled circuit protocol [Yao86] and its multiparty extension BMR [BMR90], since many MPC protocols allow to convert between additive sharing and garbled circuit representations. Examples include conversions between ABY [DSZ15] and Yao's protocol, conversions between ABY3 [MR18] and a 3-party version of Yao's protocol [MRZ15], and conversions between Motion [BDS+22] and BMR.

Yao's garbled circuit allows two parties, dubbed the garbler and evaluator, to compute a boolean function on their combined inputs. Thereby, the garbler encrypts the boolean function by assigning two keys k_0^w, k_1^w to each wire w in the circuit representation of the function, and obfuscates the truth table of each gate in the circuit by encrypting the output key with the corresponding input keys and randomly shuffling the entries. The result, called *garbled circuit*, is then send to the evaluator alongside the keys representing the input of the garbler. The evaluator then uses *oblivious transfer* (OT) [EGL82] to receive the keys corresponding to its input without leaking them to the garbler. It is then able to evaluate the whole circuit producing the final result. The main advantage of this protocol is that it has, contrary to the other protocols discussed in this section, a constant number of online rounds. Furthermore, when using all optimizations published in the literature [BMR90; NPS99; KS08; ZRE15], AND gates only require two rows in the garbled truth table, while XOR gates do not require a truth table at all. BMR, which essentially is a multiparty extension of Yao's protocol where the garbled circuit generation is distributed amongst multiple garblers, also achieves a constant number of online rounds.

2.4 Zero Knowledge Proofs

Zero knowledge (ZK) proofs, and especially computational integrity proofs combined with zero knowledge, have become a staple for many use cases (especially blockchain use cases [SYZ+21]) in the recent years. A ZK proof system is a protocol between a prover and a verifier where the prover tries to convince the verifier of the truthfulness of a statement without leaking anything beyond the assertion of the statement. More formally, a zero knowledge proof system achieves the following properties.

[1] We want to mention SPDZ$_{2^k}$ [CDE+18] as well, a SPDZ-like MPC protocol over $\mathbb{Z}_{2^k}$ which mitigates the security issues of the ring-based MAC by moving to larger rings $\mathbb{Z}_{2^{k'}}$.

- **Completeness.** A honest prover is always capable of convincing an honest verifier.
- **Soundness.** A honest verifier will detect a cheating prover with high probability. Thus, a dishonest prover only has a negligible probability to create a valid zero knowledge proof if the assertion it wants to prove does not hold.
- **Zero Knowledge.** A valid proof does not reveal more to the verifier than what can be inferred from the proven statement itself.

2.4.1 Proof of Knowledge, SNARKs and STARKs

In practice, we are usually interested in zero knowledge proofs *of knowledge* (ZKPoK), where the prover knows a witness w that verifies the given statement. In such a case, the zero knowledge property of the proof system prevents the leakage of any information about the witness to the verifier. An example of a well-known ZKPoK is Schnorr's proof [Sch91], where the proven statement is that the prover knows the pre-image k of a public *discrete logarithm commitment* $y = g^k$ for a public generator $g \in \mathbb{G}$. While many proof systems, such as Schnorr's proof, are built from an interactive protocol between the prover and the verifier, they can be made non-interactive by using the well known Fiat-Shamir (FS) transform [FS86]. Furthermore, FS transformed ZK proofs can be turned into digital signature schemes. Examples include Schnorr's signature scheme [Sch91] which is a FS transformed version of Schnorr's proof that has been standardized (in a slightly modified version) as EdDSA [BDL+12], as well as the post-quantum secure Picnic family of signature schemes [CDG+17] which are FS transformed MPC-in-the-head ZK proofs [IKO+07].

While early ZK proof systems have shown how to construct ZK protocols to prove NP relations [GMR89; GMW91], many different protocols for proving arbitrary statements which can be described by boolean circuits or polynomials over finite fields have been introduced in the recent decades. Examples include *zk-SNARKs* (zero knowledge succinct non-interactive arguments of knowledge) [Gro16; GWC19], *zk-STARKs* (zero knowledge scalable transparent arguments of knowledge) [BBH+18b], and ZK proof systems based on MPC, such as [JKO13] which is based on Yao's garbled circuits, MPC-in-the-head proofs [IKO+07], and VOLE-in-the-head proofs [BBSG+22].

In this thesis we are mainly interested in *zk-SNARKs* and *zk-STARKs* protocols, which are usually split into two distinct steps. First, in the *arithmetization* step the statement is translated into a set of polynomials according to the *arithmetization rules* of the concretely used proof system. In the second phase this representation is then transformed into a ZKPoK by using a *functional commitment scheme* in combination with an *interactive oracle proof* (IOP) [BCS16].

While the contributions of this thesis are mainly concerned with the arithmetization phase, we want to briefly mention and compare two of the most commonly used functional commitment schemes, namely *KZG* [KZG10] and *FRI* [BBH+18a].

KZG-based proof systems usually have a trusted setup phase, where for a given key k multiple discrete logarithm commitments $y_1 = g^k, y_2 = g^{k^2}, \ldots, y_d = g^{k^d}$ are published. It is crucial, however, that k must remain unknown to any prover and verifier. KZG then uses bilinear pairings and homomorphic commitments of the arithmetized polynomials to prove and verify statements. While this approach leads to comparably slow proof systems which rely on the security of the discrete logarithm hardness assumption and need to operate over the scalar fields of pairing friendly elliptic curves (e.g., BLS-381 [BLS02]), the generated proofs are small and of constant size.

On the other hand FRI-based proof systems just rely on the security of cryptographic hash functions and the FS transform. This allows FRI to be instantiated with small prime fields which come with fast and efficient algorithms for the modular reduction, such as the 64-bit Goldilocks field with $p = 2^{64} - 2^{32} + 1$ and the 31-bit Mersenne prime field with $p = 2^{31} - 1$. Thus FRI-based proof systems, which use these efficient fields, are usually significantly faster compared to KZG-based ones and do not require a trusted setup. This advantage however comes at the cost of large and variable size proofs.

2.4.2 Arithmetization

As mentioned in the previous section, a statement to be proven with a zk-SNARK or zk-STARK first needs to be transformed into an arithmetic representation. In the literature many different arithmetic representations exist, such as R1CS constraint systems [Gro16], *arithmetic intermediate representations* (AIR) [BBH+18b], Plonk constraints [GWC19], and Plonkish [AST+22], which can be interpreted as a generalization of the former mentioned ones. In this section we investigate some of them in more detail.

Rank-1 Constraint Satisfaction Systems (R1CS). Many zk-SNARKs, such as the famous Groth16 [Gro16] proof system, rely on R1CS arithmetization. In these systems, a statement to be proven needs to be transformed into a set of n constraints such that the statement is only true iff the set of constraints is satisfied. These constraints must be of the form

$$\forall j \in \{1, 2 \ldots, n\} : \quad \left(\sum_{i=1}^{m} u_{i,j} a_i\right) \cdot \left(\sum_{i=1}^{m} u_{i,j} a_i\right) = \left(\sum_{i=1}^{m} w_{i,j} a_i\right)$$

where $a_1, a_2, \ldots, a_m$ are the witness variables (with $a_1 = 1$) which are known by the prover and kept hidden from the verifier by the ZK property of the proof. Furthermore, the public variables $u_{i,j}, v_{i,j}, w_{i,j}$ describe the j-th constraint and depend on the statement to be proven. In other words, R1CS constraints are degree-2 relations of weighted inner products of the witness variables which allow to each capture relations including one multiplication. Since prover complexity scales with the number of constraints n and linear constraints can be embedded into the next multiplicative constraint, the number of multiplications in the

representation of the statement is usually regarded as the main cost metric when using this arithmetization technique.

Plonk and Plonkish. With the Plonk [GWC19] proof system a more flexible arithmetization method was introduced. In the beginning, however, Plonk was arithmetized by using generic constraints of the form

$$q_{L_i} \cdot a_{L_i} + q_{R_i} \cdot a_{R_i} + q_{O_i} \cdot a_{O_i} + q_{M_i} \cdot (a_{L_i} a_{R_i}) + q_{C_i} = 0$$

where a values are again the witness variables and the q values describe the constraints. This form of constraints can be used to instantiate addition and multiplication gates by setting the q values accordingly. However, the way how Plonk translates the arithmetization to a ZK proof does not limit the arithmetization to just addition and multiplication gates following this form. In fact, any kind of polynomial custom gate can be implemented [AST+22]. We describe this approach, later dubbed *Plonkish* arithmetization,[1] in the following.

In Plonkish a statement or computation to be proven is represented as a table where each cell consists of assignments of witness variables. This table is called *trace* and is filled by the prover during proof generation. Furthermore, the proof consists of constraints (sometimes called gates) in the form of multivariate polynomials which describe the relations between the witness cells in each row (and sometimes also cells in different rows given by a fixed offset) of the trace. These gate constraints must evaluated to zero for each row. To construct proofs with different gates in each row, one can use *filter constraints*, which activate the multivariate relation polynomials by multiplying them with selector values, which are either zero or one. Additionally, *copy constraints* (sometimes called *equality constraints*) can be used to ensure equality between witness cells across multiple rows.

We give an example execution trace in Table 2.1. The trace consists of two rows and three witness variables a, b, c. There are two different polynomial constraints (i.e., gates), an addition gate $c - (a + b) = 0$ in the first row and a multiplication gate $c - (a \cdot b) = 0$ in the second row. The gates are combined into one constraint $s \cdot (c - (a \cdot b)) + (1 - s) \cdot (c - (a + b)) = 0$ using the selector column s.

Table 2.1: Plonkish execution trace with two rows, one selector column s and three witness variables a, b, c. Each row satisfies the relation $s \cdot (c - (a \cdot b)) + (1 - s) \cdot (c - (a + b)) = 0$.

step	s	a	b	c
1	0	25	5	30
2	1	4	3	12

During proof generation, each column of the trace is interpolated into a polynomial, and the multivariate constraint polynomials are rewritten using

[1] https://twitter.com/feministPLT/status/1413815927704014850

these interpolated witness polynomials. Then, KZG or FRI is used to prove validity of the resulting polynomials. Since the number of columns is equal to the number of witness polynomials, the number of rows affects interpolation cost of the witness polynomials, and the maximum degree of the constraint polynomials affects performance in KZG and FRI, the prover cost depends on the number of rows n, number of columns m and the maximum constraint degree d and is often estimated as the area degree product $n \cdot m \cdot d$.[1] The cost metric is thus also related to the multiplicative complexity of the circuit, while different tradeoffs (e.g., smaller degree versus more columns) are possible to optimize performance.

Lookup Constraints. In [GW20] the authors introduce the possibility to efficiently prove a lookup relation $y = T[x]$ for a publicly known lookup table T inside a Plonk-based proof system. This lookup argument (and improvements proposed in e.g., Halo2 [Zca22][2] or [ZBK+22]) has the potential to speed up proving any statement with an expensive arithmetic representations that can be described by a small lookup-table. Examples include range checks or the high-degree lookup tables in the hash functions Reinforced Concrete [GKL+22] and Tip5 [SLS+23]. In this thesis, we use the lookup argument to introduce the hash function Monolith which has a plain hashing performance comparable to SHA-3 [Nat15; BDP+11], while being more efficient inside a ZK circuit compared to some instances of the well known arithmetization friendly hash function Poseidon [GKR+21]. We refer to Chapter 8 for more details.

2.5 Simplified Cost Metric Summary

In Table 2.2, we roughly summarize the cost metrics for the different PETs we discussed in this chapter. This table is intended to give a high-level overview of the costs while concrete metrics depend on specific systems and implementations.

Table 2.2: Roughly simplified cost metric summary of different classes of PETs. ● represents the main cost metric, ◑ indicates that the operation plays a role in performance, and ○ corresponds to the operation being considered free. ✗ indicates that the operation is not available.

PET	# Multiplications	Multiplicative Depth	# Additions	# Automorphisms	# Lookups
SHE	◑	●	○	◑	✗
TFHE	●	○	◑	✗	●
SPDZ/Shamir	●	◑	○	✗	✗
R1CS	●	○	○	✗	✗
Plonkish	●	○	◑	✗	●

[1] This metric ignores many terms, such as $\log(n)$ introduced by FFTs in the interpolation of the columns, but can be regarded as a decent and simple estimation of the real cost.

[2] https://zcash.github.io/halo2/design/proving-system/lookup.html

3

Symmetric Primitives for Private Computations

Since privacy enhancing technologies (PETs) have become more prominent in the academic literature, it has also become evident that many use cases profit from using symmetric encryption schemes or hash functions. However, these PETs come with a different cost metric compared to being efficient in either hardware or software. Consequently, traditional symmetric designs are not efficient in combination with PETs and many new symmetric primitives have been proposed in the literature. In this chapter we examine the use cases and different design criteria of these new primitives.

3.1 PET Use Cases using Symmetric Primitives

While symmetric primitives have been used as a common benchmark in early PETs publications to show the improvements of specific schemes (e.g., evaluating AES under homomorphic encryption [GHS12] or MPC [PSS+09] to benchmark BGV and Yao's garbled circuits), it has also become clear that many use cases can use symmetric primitives for more efficient implementations, or outright rely on them. In this section we review these use cases.

3.1.1 Homomorphic Encryption Use Cases

Homomorphic encryption (HE) use cases usually follow a client-server architecture,[1] where the client encrypts some data, sends the ciphertexts to the server, and the server performs the use case on the encrypted data. This produces an encrypted result, which is then transported to the client who has the decryption key. Translating any use case, such as calculating statistics or classifying some data using a machine learning model, to the HE setting however usually comes with a significant performance penalty of several orders of magnitude. But not only computational complexity is affected by homomorphic encryption, communication complexity is also affected by *ciphertext expansion*. This basically means that data encrypted under a HE scheme is orders of magnitude larger compared to the non-encrypted plaintexts. This expansion can result in a significant performance penalty for the whole use case, especially when the client device has a slow bandwidth connection.

One of the main use case for which symmetric primitives are used in combination with HE is *hybrid homomorphic encryption* (HHE) [NLV11] (sometimes also dubbed *transciphering*) which aims to reduce ciphertext expansion and consequently client bandwidth requirements in HE use cases. The workflow is as follows. In a setup phase the client sends a HE encryption $c_k = \mathsf{HE.Enc}(k)$ of a symmetric encryption key k to the server. Then, for the actual use case evaluation, the client sends the data d encrypted under a symmetric encryption scheme $c_{\mathsf{SYM}} = \mathsf{SYM.Enc}_k(d)$ using the key k. Since symmetric encryption schemes usually do not suffer from ciphertext expansion, this transmission step is as costly as in the non-encrypted version of the use case. However, since symmetric encryption lacks the homomorphic properties required to evaluate the use case, the server first needs to transform the symmetric ciphertext c_{SYM} into a homomorphic ciphertext c_{HE}. This is done by homomorphically evaluating the symmetric decryption circuit $c_{\mathsf{HE}} = \mathsf{HE.Eval}(\mathsf{SYM.Dec}, (c_{\mathsf{SYM}}, c_k))$ using the encrypted key c_k from the setup phase. Then the server can continue with performing the actual use case on c_{HE}.

Thus HHE trades a reduction of communication complexity with an increased workload on the server side. The goal for symmetric designers, therefore, is to minimize the cost of the symmetric decryption circuit when evaluated under homomorphic encryption. This requires that the symmetric encryption scheme natively operates on the field of the HE scheme, which is usually either $\mathbb{F}_2$ (e.g., for the binary interface in TFHE) or some larger prime field $\mathbb{F}_p$ (e.g., for many BFV/BGV use cases). Additionally, for SHE schemes, which are able to evaluate a circuit up to a predefined multiplicative depth (see Section 2.2.2) efficiency requires to have a low multiplicative depth in the symmetric decryption circuit. This is especially important when considering that the overall multiplicative depth of the circuit evaluated on the server side includes the depth of the decryption

[1] Usually, only one client is allowed since in multi-client settings each party knowing the secret decryption key can potentially decrypt the secret inputs of other clients. One can circumvent this problem by using *multikey* [LTV12; CDK+19] or *multiparty* [MTB+21] HE schemes.

circuit, as well as the depth of the actual use case. For FHE schemes where an efficient bootstrapping operation is available, the designers usually focus on minimizing the total number of gates, especially the more expensive multiplication and bootstrapping gates, in the decryption circuit.

In this thesis we investigate the usefulness of a large number of symmetric primitives proposed for HHE and introduce a new symmetric cipher, dubbed Pasta, which is highly efficient for HHE in BGV and BFV. We refer to Chapter 6 for more details.

Another application of symmetric primitives in combination with HE is to instantiate an *oblivious pseudorandom function* (OPRF) [PSS+09] which can be used to realize *private set intersection* (PSI) use cases (e.g., similar to [KLS+17; KRS+19] where MPC-based OPRFs are used). The goal of an OPRF is that for two parties P_1 and P_2, the first party owns some data d, while the second party owns a key k. As a result of the protocol execution, P_1 additionally should learn $c = \mathsf{PRF}(d, k)$ while learning nothing about k. On the other hand, P_2 should not learn d or c. By letting P_2 evaluate the PRF on d encrypted under a HE key owned by P_1, P_2 can evaluate the PRF for P_1 without learning d.

3.1.2 Multi-party Computation Use Cases

Symmetric primitives have been used in several ways in combination with secure multi-party computation (MPC) use cases. On one hand MPC has been used for key management to mimic a trusted execution environment to build a virtual hardware security module (v-HSM).[1] The main idea behind this use case is to protect a symmetric encryption key by splitting it into MPC-shares and store them on multiple devices. Then, symmetric encryption/decryption can only be performed when all (or a predefined number of) the devices participate in an MPC protocol using their shares. Thus, encryption/decryption can still be performed, but it is significantly harder for attackers to steal the key since multiple devices need to be breached. Furthermore, by splitting the key into shares owned by multiple parties, the key can only be used if all or a majority of the parties agree to do so, mimicking a secure voting for en- or decryption.

Another use case is the asynchronous transportation of data into an MPC computation [DK10; GRR+16]. Consider the setting of delegated MPC where the computing parties are different to the data providers. In order to get the data into the MPC protocol, a data provider needs to directly share its data with the computation nodes which requires them to be online during the time of the protocol execution. An asynchronous variant can be to share the data, encrypt the shares under the public keys of the computing parties and store the ciphertexts at a public place. The computing parties then can fetch the data when they are ready for the MPC computation. Since this solution requires that the data provider encrypts the dataset n times using slow public key encryptions where n is the number of computing parties, and has to store n times the ciphertexts, a more efficient solution can be to let the client encrypt the dataset

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

using a symmetric encryption scheme, and only share and encrypt the symmetric key. The computational complexity on the data provider side is thus reduced since symmetric encryption is usually significantly faster compared to public key encryption, and the storage requirement is reduced to just the size of the data plus the public key encryption of the shares of the key. Similar to HHE (see Section 3.1.1), these advantages can be important when data providers are embedded devices with weak CPUs and slow bandwidth connections [BPA+23]. The same principles can be applied to the setting where the recipient of the result of the computation is different to the set of computing parties, as well as when the computing parties want to halt the MPC execution and resume at a later point [GRR+16].

In all the use cases discussed so far, the symmetric primitives are used to encrypt or decrypt potentially large amounts of data inside an MPC protocol. Thus, the symmetric ciphers should be designed to natively work over the native datatype of the used MPC protocol for efficiency (which are prime fields for the schemes we use in this thesis) and should minimize the number of multiplications. This reduces the costly preprocessing phase and minimize communication complexity during MPC execution (see Section 2.3). In this thesis we propose a new symmetric cipher, dubbed Hydra, which is efficient for encrypting large amounts of data in these use cases. We refer to Chapter 7 for more details.

Similar to HE (see Section 3.1.1), one can also combine symmetric primitives with MPC to instantiate an OPRF [PSS+09]. Indeed, MPC-based OPRFs have been used to instantiate use cases, such as *private set intersection* (PSI) [KLS+17; KRS+19] or database joins for private databases [LTW13; MRR20].

Furthermore, symmetric ciphers in combination with MPC protocols can be used to instantiate post-quantum secure signature schemes via zero knowledge proofs. Concretely, MPC-in-the-head [IKO+07] based signatures such as Picnic [CDG+17] and VOLE-in-the-head [BBSG+22] based signatures such as FAEST [BBSG+23]. In these signatures a *one way function* (OWF) is instantiated via a PRF which in turn is instantiated with the symmetric cipher. The symmetric encryption key k then acts as the secret signing key, and the public key is a plaintext-ciphertext (p, c) pair such that $c = \mathsf{PRF}(p, k)$. The signature then essentially is a zero knowledge proof proving the PRF relation between the secret key and the public key. In MPC-in-the-head this zero knowledge proof is instantiated via a simulation of an MPC protocol. Since the signature size scales with the communication between the MPC parties, a PRF with low multiplicative complexity should be used. Furthermore, this setting allows to design highly efficient symmetric primitives, such as Rain [DKR+22], since for every secret key only one plaintext-ciphertext pair (i.e., the public key) is ever known to an attacker, limiting the attack vectors which can be used to attack the PRF.

3.1.3 Zero Knowledge Proof Use Cases

Symmetric hash functions are integral parts of many zero knowledge (ZK) proof use cases. Thereby, the goal often is that the output of the hash function or a chain of hash functions is publicly known and a prover proves to know the

input of the (chain of) hash computations. A notable example is a Merkle tree accumulator [Mer89].

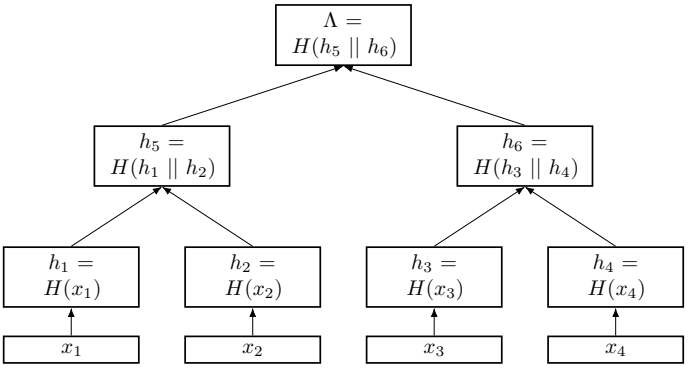


Figure 3.1: A 2-ary Merkle tree build from a set with 4 elements (x_1, x_2, x_3, x_4) and the root node (accumulator) Λ.

A Merkle tree (as depicted in Figure 3.1) is a data structure which succinctly represents a set $(x_1, x_2, \ldots, x_n)$ of n elements as just one element Λ named the accumulator. This is done by interpreting the hash of the elements of the set as leaf nodes of a t-ary tree. The tree nodes in each level then correspond to the hash computed from its t children nodes using a collision resistant hash function $H(\cdot)$. Additionally, one can prove that a element x_i is part of the tree with the help of a witness wit_{x_i} which consists of all hash values h_i such that one can recompute Λ from x_i and wit_{x_i}. As example, the witness wit_{x_2} for the Merkle tree in Figure 3.1 consists of $\mathsf{wit}_{x_2} = ((h_1, \mathtt{l}), (h_6, \mathtt{r}))$ where $(h_1, \mathtt{l})$ indicates that h_1 is the left input for the hash function and $(h_6, \mathtt{r})$ indicates that h_6 is the right input. Thus, using x_2 and wit_{x_2} the accumulator can be recomputed as $\Lambda' = H(H(h_1||H(x_2))||h_6)$. If $\Lambda' = \Lambda$ a verifier gets high assurance that x_2 is part of the accumulated set.

To prevent leaking x_i or the witness wit_{x_i} to the verifier, one can evaluate the verification equation (i.e., recomputing Λ from x_i and the witness wit_{x_i}) inside of a ZK proof. The cost of creating the proof then directly corresponds to the cost of the chain of hash functions.

A Merkle tree is often an integral part of standalone zero knowledge use cases, such as anonymous credentials [CL02; NJ21], or anonymous transactions in cryptocurrencies [OWW+20; Zca; Tor]. Thus, hash functions designed for ZK use cases should minimize the cost metric of a given proof system (see Section 2.4), which often requires minimizing the multiplicative complexity of the equivalent representation (see Section 3.2.4), as well as minimizing the maximum degree of these representations. However, Merkle trees are also used as commitments in recursive SNARKs [BBH+19]. In recursive SNARKs a ZK proof includes a verification of a ZK proof in the previous step, reducing the size of the proof created for a use case at the end of the recursion steps. At each step in the recursion a Merkle tree commitment is computed which is then opened (in ZK)

in the next step [COS20]. Thus, a hash function used for recursive SNARKs should not only be efficient inside a ZK circuit, but also efficiently computable in plain outside the proof to minimize the cost of creating the Merkle tree in the first place. Since most state-of-the-art ZK-friendly hash functions, such as Poseidon [GKR+21], are orders of magnitude slower in plain compared to e.g., SHA-3 [Nat15; BDP+11] we propose a new hash function in this thesis, dubbed Monolith, which provides plain hashing performance comparable to SHA-3, while also outperforming many instances of Poseidon inside a ZK proof. We refer to Chapter 8 for more details.

We want to mention that in this thesis we also investigate the usage of efficient public key accumulators, which are also compatible with efficient zero knowledge proofs [ACN13], instead of Merkle tree accumulators as part of anonymous credential systems [NJ21]. To mitigate the problems of the trusted setup required for public key accumulators, we instantiate a new MPC-based accumulator and investigate its performance for different MPC protocols with different security guarantees. We refer to Chapter 9 for more details.

3.2 Symmetric Design Principles for PETs

In this section, we discuss the history and state-of-the-art principles on how to design a symmetric primitive that is optimized for PET use cases. We start by introducing the substitution-permutation network, a common way to design symmetric primitives, in Section 3.2.1 before we review concrete primitives and design strategies in the remaining sections. Thereby, we focus on the most significant developments in relation to the contributions in this thesis, namely developments which directly impacted our design choices and the most important competitors. Thus, we do not discuss schemes, such as Rain [DKR+22], AIM [KHS+22], Arion [RST23], and others.

3.2.1 Preliminaries: Substitution-Permutation Networks

Modern symmetric primitives are usually built from cryptographic permutations, which can be categorized as either keyed permutations when a symmetric key is used (e.g., for block ciphers) or unkeyed ones which are then used in a mode of operation (e.g., a sponge mode [BDP+08] for hashing). These permutations then usually consist of iterating over one or more round functions, a technique going back to Claude Shannon [Sha49]. Thus a cryptographic permutation π over a finite field $\mathbb{F}$ with an internal state size t and r rounds can be described as $\pi : \mathbb{F}^t \to \mathbb{F}^t$ with

$$\vec{y} = \pi(\vec{x}) = F_r \circ F_{r-1} \circ \cdots \circ F_1$$

where $\vec{x}, \vec{y} \in \mathbb{F}^t$ and $F_i : \mathbb{F}^t \to \mathbb{F}^t$ is an invertible round function.

Nowadays the most common way to instantiate these round functions is via a *substitution-permutation network* (SPN), which splits the round function into a non-linear layer $S(\cdot)$ (substitution) and an affine layer $A(\cdot)$ (permutation) such

that

$$F_i(\vec{x}) = A_i \circ S(\vec{x})$$

where $A_i(\vec{x})$ is often instantiated as matrix-vector multiplication followed by the addition of a publicly known round constant such that

$$A_i(\vec{x}) = M \cdot \vec{x} + \vec{c}^{(i)}$$

with the matrix $M \in \mathbb{F}^{t \times t}$ and the round constants $\vec{c}^{(i)} \in \mathbb{F}^t$. In a keyed version of the SPN, the round function usually also consists of the addition of a round key which is derived from the encryption key. The non-linear layer $S(\vec{x})$ is often instantiated by applying a non-linear function, often dubbed *S-box*, to each word in the state, such that

$$S(\vec{x}) = (S_1(x_1), S_2(x_2), \ldots, S_t(x_t))$$

for S-boxes $S_i : \mathbb{F} \to \mathbb{F}$. In many cases just one S-box is used in the whole layer, i.e., $S_1(\cdot) = S_2(\cdot) = \cdots = S_n(\cdot)$.

The popularity of the SPN stems from its simple and generic construction while allowing for convincing security arguments, e.g., using the wide-trail design strategy [DR01]. Indeed, many well known designs are constructed as SPNs, such as AES [DR00; DR02] or SHA-3 [Nat15; BDP+11]. However, when used in PETs, simply applying a SPN often leads to inefficient constructions. Especially, having a full non-linear layer is often not required for security, but increases the multiplicative complexity of the design, which worsens the performance when used in PETs. Thus, many of the designs which we discuss in this chapter use a *partial SPN* (P-SPN), where only a part of the state is transformed in the substitution layer $S(\cdot)$. Concretely, in a P-SPN $S(\cdot)$ can be defined as

$$S(\vec{x}) = (S_1(x_1), S_2(x_2), \ldots, S_m(x_m), x_{m+1}, x_{m+2}, \ldots x_t)$$

for $1 < m < t$. An example of a P-SPN is LowMC [ARS+15] which we discuss in the next section.

3.2.2 Early Symmetric Designs

As discussed in Chapter 2 the cost metric for the different PETs usually relates to the amount of non-linear operations. In MPC, the total number of multiplications directly corresponds to the amount of data exchanged between the computing parties, while the multiplicative depth is equal to the number of communication rounds. In SHE schemes, the multiplicative depth is the main performance metric, whereas in FHE schemes the multiplications and bootstrapping operations (which in turn also relate to multiplications and the depth) are the most expensive computations. Furthermore, in many arithmetization models for modern ZK proof systems the number of multiplications, or degree of polynomials to represent constraints, directly contributes to the prover cost. Thus, the first symmetric designs tried to minimize the number of multiplications required for secure encryptions. Indeed, LowMC (**low** **m**ultiplicative **c**omplexity) [ARS+15] and MiMC (**mi**nimal **m**ultiplicative **c**omplexity) [AGR+16] are among the first designs proposed for these use cases.

LowMC [ARS+15]. LowMC is a block cipher proposed over binary fields $\mathbb{F}_2$ which achieves a low total number of multiplications by using a P-SPN and a low-degree non-linear layer. Indeed, only a pre-defined, parameterizable number of bits is transformed by 3-bit S-boxes, where each individual S-box only requires 3 AND gates and has an AND-depth of only 1. Furthermore, the linear layers of LowMC are sampled uniformly at random (while ensuring invertibility) during instantiation of a specific LowMC instance. The LowMC round function is depicted in Figure 3.2.

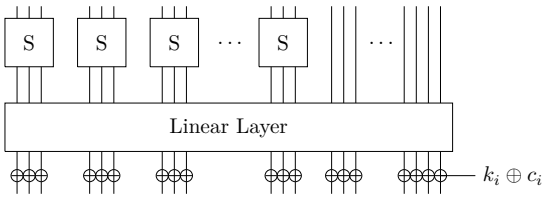


Figure 3.2: One round of encryption with LowMC (modified from [ARS+15]).

LowMC was initially proposed for use cases in MPC and HE and has found success as the *one way function* (OWF) used in the MPC-in-the-head [IKO+07] based post quantum signature scheme Picnic [CDG+17; CDG+20], and as an PRF in private set intersection (PSI) use cases [KRS+19].

Regarding security, many publications with new cryptanalytic results have been published in the literature since LowMC was first proposed. These works showed improvements for multiple attack vectors, such as higher-order differential attack [DEM15], interpolation attacks [DLM+15], difference enumeration attacks [RST18; LIM21; QYS+23], and meet-in-the-middle attacks [LSW+22]. The LowMC instance generation script from the original Github repository[1] already includes updates according to these new cryptanalytic results. However, lots of research also focused on the usage of LowMC in Picnic which limits the data complexity of attacks to only 1 plaintext-ciphertext pair. These results were proposed in [BBD+20; BBV+21; Din21; BBC+22; LMS+22] and suggest that the proposed round numbers are not providing full security when considering an attackers cost model where large memory accesses are free.

MiMC [AGR+16] and GMiMC [AGP+19]. LowMC was designed as a cipher over $\mathbb{F}_2$ to minimize the number of AND gates, whereas MiMC was designed to minimize the number of multiplications while working over $\mathbb{F}_q$ for large $q \in \{2^n, p\}$ (for a prime number p) with sizes up to 256 bits such that $\gcd(3, q-1) = 1$. MiMC's internal state just consists of one field element $x \in \mathbb{F}_q$ where the round function is defined as

$$F_i(x) = (x + k + c^{(i)})^3, \tag{3.1}$$

where $k \in \mathbb{F}_q$ is the key and $c^{(i)} \in \mathbb{F}_q$ is the round constant in round i.

[1] https://github.com/LowMC/lowmc

Having an internal state consisting of just one field element $\in \mathbb{F}_q$ allows one to build an encryption scheme. However building a hash function (as required for, e.g., ZK use cases as discussed in Section 3.1) using the sponge function [BDP+08] requires to split the state into an inner and outer part. Thus, the authors also introduced Feistel-MiMC, a Feistel network consisting of two branches, where the update function is instantiated with Equation (3.1). This approach was then extended to GMiMC in [AGP+19], where Equation (3.1) was used as an update function in various different general Feistel modes of operation to have a flexible state size which is not limited to just one or two field elements. For example, $\text{GMiMC}_{\text{ERF}}$ is instantiated with the round function

$$F_i'(x_1, x_2, ..., x_t) = (x_2 + F_i(x_1), x_3 + F_i(x_1), ..., x_t + F_i(x_1), x_1)$$

where $F_i(\cdot)$ is defined as Equation (3.1).

Since the MiMC and GMiMC design approaches lead to rather large round numbers required for security, they are not usable in combination with SHE schemes. However, they excelled (compared to previous state of the art) in use cases where the total number of multiplications was the main bottleneck, namely MPC and ZK. However, many improvements have been proposed in the literature which further reduce the cost when used in these PETs. We discuss these primitives in the next sections. It is worth noting that many of these designs use a common building block which was introduced by MiMC, namely the low-degree $x \mapsto x^d$ power map over relatively large fields, where d is the smallest integer, such that $\gcd(d, p-1) = 1$ to ensure invertibility.

Regarding cryptanalysis on MiMC, researchers proposed new results on interpolation attacks [LP19], collision attacks [Bon19], and higher-order differential attacks [EGL+20; BCP23]. GMiMC, on the other hand, has seen attacks on the full permutation exploiting zero-sum distinguishers [BCD+20].

3.2.3 Hades and Low-Degree Power Maps

Low degree power maps proved to be effective for constructing symmetric ciphers over $\mathbb{F}_p$ with a small number of multiplications and motivated further research in this direction. In [GLR+20] the authors introduce the Hades design strategy, which combines the advantages of both a classical SPN and a partial one. The Hades design strategy, as depicted in Figure 3.3, uses two round functions. The first one is built as a classic SPN with a full non-linear layer which is intended to provide security against statistical attacks using the wide-trail design strategy [DR01]. These full layers are used in the beginning and in the end of the design. The second one is a P-SPN which is intended to provide security against algebraic attacks while being cheaper in PETs compared to a full round. These partial rounds are located in the middle of the construction.

In [GLR+20] the authors introduce HadesMiMC, which combines the low degree power maps as used in MiMC and GMiMC with *maximum distance separable* (MDS) matrices [Vau94] in the linear layers, which have an optimal branch number of $t + 1$ for a $t \times t$ matrix to force at least $t + 1$ active S-boxes in statistical attacks. It is an instantiation of Hades with only one S-box

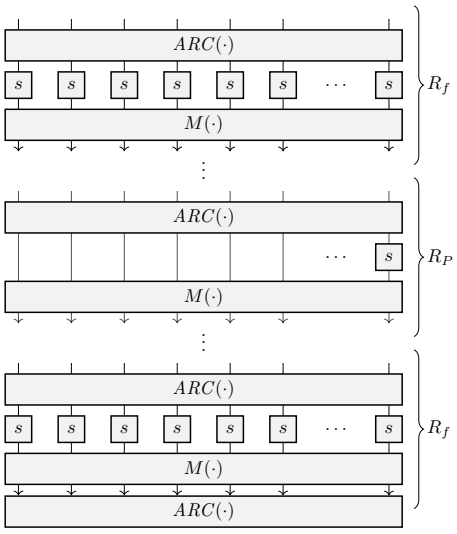


Figure 3.3: The HADES design strategy where $M(\cdot)$ denotes a matrix vector multiplication and $ARC(\cdot)$ denotes round constant additions, which includes adding a round key in a keyed mode of operation (modified from [GLR+20]).

in the partial layers and each S-box being instantiated with $x \mapsto x^d$. This construction proved to be significantly more effective in reducing the number of multiplications compared to MiMC and GMiMC, and was thus the preferable choice for encryption in MPC use cases. Furthermore, in [GKR+21] the authors use HADESMiMC inside the sponge mode of operation to create the hash function POSEIDON optimized for zero knowledge use cases. It has since become the inofficial standard for hash functions for ZK use cases due to its simplicity, efficiency, and widespread adaption [COS20; OWW+20; LG21; KST22; RWG+22; SBV22].

While the partial non-linear layers in HADES allow to instantiate efficient ciphers and hash functions with a small number of multiplications, one needs to be careful when instantiating the accompanying affine layers. Indeed, the results in [BCD+20; KR21; GRS21] show how some unwanted properties, such as invariant subspace trails, can lead to attacks on the whole HADES permutation.

After being successfully used in POSEIDON, HADES has been used as part of many follow-up proposals. In NEPTUNE [GOP+22] the authors use quadratic non-linear functions to further reduce the multiplicative complexity compared to POSEIDON. Furthermore, it uses more efficient linear layers to improve plain performance. In POSEIDON2 [GKS23] the authors aim to make POSEIDON faster for plain hashing by exchanging its linear layers with more efficient ones but keep the $x \mapsto x^d$ power maps for simplicity. The REINFORCED CONCRETE [GKL+22] hash function, inspired by HADES, also employs two different round functions where the first one is designed to protect against statistical attacks and is located at the beginning and the end of the permutation while a single middle round is

used to provide algebraic security. The Monolith hash function, proposed in this thesis, also follows the design of Hades and Reinforced Concrete to have two different layers for statistical and algebraic security (see Chapter 8). The Bars layer, which is used for algebraic security, is only applied to parts of the state resembling the partial parts of Hades. Contrary to Hades though, we use the different layers in each round of the design.

Furthermore, in this thesis we use Hades to instantiate the body $\mathcal{B}$ of Hydra, one of the two permutations in our new Megafono-based cipher optimized for MPC use cases (see Chapter 7). In [Gra23] the author improves on this permutation $\mathcal{B}$ by exchanging the power maps in the full layer of the Hades construction with the SI-lifting function $F(x_1, x_2) = x_1^2 + x_2$ to create the *pseudorandom function* (PRF) Pluto and use it in the Megafono construction to create Hydra++.

3.2.4 Low-Degree Equivalence

In another line of work, researchers combined the low-degree power map x^d with its high-degree inverse power map $x^{1/d}$ where $d \cdot (1/d) = 1 \mod p - 1$ such that $(x^d)^{1/d} = x \mod p$. The reason for this is two-fold. First, when translated to constraints in ZK use cases one can rewrite $y = x^{1/d}$ to $y^d = x$ leading to an equivalent representation that is as efficient as the x^d power map. Secondly, in MPC use cases one can delegate most of the computational work of evaluating $x^{1/d}$ to plain computations which do not require communication and the cost gets reduced to evaluating x^d plus two multiplications inside the MPC protocol (see [AAB+20] and the discussion in Chapter 7).

Friday [AD18] and Rescue [AAB+20]. The first designs to exploit these advantages were Jarvis and Friday [AD18] which however got immediately broken by a Gröbner basis attack in [ACG+19]. After this first unsuccessful attempt to design symmetric primitives using this strategy, the Rescue permutation [AAB+20] (later optimized as Rescue-Prime in [SAD20]) was introduced and used as a symmetric cipher in MPC applications and as a sponge hash function in ZK use cases. Rescue is built as an SPN with the twist that each round of Rescue consists of two SPN rounds. The first round is equal to a full round in HadesMiMC/Poseidon. It consists of an MDS matrix in the linear layer and the $x \mapsto x^d$ power maps in the non-linear one. The second round is constructed similar, but with the difference that the non-linear layer features the high-degree maps $x \mapsto x^{1/d}$ giving Rescue stronger resistance against algebraic attacks. Indeed, only ≈ 10 rounds of Rescue are required for security (depending on the state size t and the prime field $\mathbb{F}_p$) compared to > 60 rounds for Poseidon. While Poseidon still requires a smaller number of multiplications than Rescue (see, e.g., benchmarks in [GHR+23]), Rescue's advantage is visible in ZK proof systems with the AIR arithmetization. Indeed, one Rescue-Prime round described as $\vec{y} = M \cdot (M \cdot \vec{x}^d)^{1/d}$ (ignoring round constant addition for simplicity) can be

arithmetized with the degree-d constraint $(M^{-1} \cdot \vec{y})^d - M \cdot \vec{x}^d = \vec{0}$.[1] Consequently, Rescue won the STARK friendly hash competition [BGL20; Tea21] hosted by StarkWare.[2]

Griffin [GHR+23], Anemoi [BBC+23] and Grendel [Sze21]. The main disadvantage of Rescue is its slow plain encryption and hashing performance due to the costly evaluation of many $x^{1/d}$ power maps. Thus this design approach was improved in the Griffin [GHR+23] and Anemoi [BBC+23] publications, which only focus on permutations for ZK-friendly hash functions and do not propose ciphers for MPC. In Griffin the authors combine the two SPN rounds of each Rescue round into just one round function. Furthermore, only one field element in the internal state of Griffin gets transformed by the x^d power map, and only one gets transformed by $x^{1/d}$. To get sufficient statistical and algebraic security, the authors introduce the novel Horst construction which mixes the output of the two power maps with the rest of the internal state using a low-degree function. Consequently, this approach combines the advantages of Rescue with a smaller number of total multiplications and fewer costly $x^{1/d}$ power maps for faster plain performance. The Anemoi permutation, on the other hand, features the novel Flystel construction, where the state is split into two parts with individual linear layers, and the non-linear layer mixes these branches by having multiple S-boxes where each S-box gets as input one field element from both branches. Each S-box, thereby, uses one $x^{1/d}$ power map which combined with the smaller and more efficient matrices in the linear layers lead to faster plain performance compared to Rescue. Compared to Griffin though, the result is still a slower plain performance. Regarding efficiency in proof systems, Anemoi achieves a similar performance compared to Griffin across many different arithmetization techniques (see, e.g., [GHR+23]).

Another line of work introduced the Grendel [Sze21] permutation, which is a traditional SPN, where the non-linear layers where constructed as $x \mapsto x^d \cdot \left(\frac{a}{p}\right)$ where $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \in \{-1, 0, 1\}$ is the Legendre symbol. The advantage of this approach is that $x \mapsto x^2 \cdot \left(\frac{a}{p}\right)$ becomes invertible, the Legendre symbol is of high algebraic degree, and – as shown by the authors – it has an efficient equivalent representation inside ZK proof systems. However, due to mapping field elements into a set of just three distinct values, the Legendre symbol is vulnerable to guessing attacks as shown in [GKR+22]. This attack led to a significantly higher number of rounds than initially anticipated resulting in Grendel having a worse performance compared to its competitors.

Chaghri [AMT22]. So far we just discussed block ciphers, permutations and hash functions with high-degree components which were optimized for MPC and/or ZK use cases. Interestingly, the Chaghri [AMT22] block cipher follows

[1] This is also the reason why Rescue describes their two SPN rounds as one Rescue round.
[2] https://starkware.co/

a similar approach as RESCUE but is proposed for HE use cases. CHAGHRI is designed to work over the binary extension field $\mathbb{F}_{2^{63}}$. In HE schemes instantiated over binary extension fields (as is possible for, e.g., BGV) one can evaluate the Frobenius automorphism $\theta_i : a(X) \mapsto a(X^i)$ for $i = 2^k$ by permuting the polynomials in the ciphertexts followed by a key switching operation. This is similar to Galois automorphisms as discussed in Section 2.2.2. Using θ_i one can compute $y = x^{2^k}$ without explicitly performing a homomorphic multiplication, and the noise increase just depends on the used key switching protocol. Thus, while being instantiated with high-degree power maps, CHAGHRI can be implemented with a comparably small number of multiplications and depth, leading to a smaller noise consumption during homomorphic evaluation. While CHAGHRI is less efficient in HE use cases than e.g., RASTA [DEG+18], it is the currently most efficient block cipher for HE which does not rely on randomization (see Section 3.2.6). This allows it to be used in e.g., *oblivious pseudorandom function* (OPRF) use cases where randomization is an undesired property. For the sake of completeness we mention that CHAGHRI has been broken in [LAW+23], but a fix with minimal performance overhead is already available in [AMT22].

3.2.5 Reinforced Concrete and Lookup Tables

Low-degree power maps $x \mapsto x^d$ and their high-degree counterparts $x \mapsto x^{1/d}$ with low-degree representations have been prominently used in MPC and especially ZK use cases. However, while being faster in combination with their respective PET, designs featuring these traits are significantly slower compared to AES and SHA-3 for plain encryption and hashing. In some use cases, e.g., recursive ZK proof systems [COS20; Pol22b], this slow plain performance is a bottleneck. Luckily, the introduction of the lookup argument [GW20; ZBK+22] in ZK proof systems gave designers of symmetric primitives a new tool for optimizations. The lookup arguments allow to efficiently prove a lookup table relation $y = T[x]$ inside a proof system. In other words, one can efficiently prove that the witness tuple (x, y) satisfies $y = T[x]$ for a public lookup table T. Since most ZK proof systems are proposed to work over large prime fields $\mathbb{F}_p$ with $\log_2(p) > 64$ it is not obvious how to use lookup tables to speed up a symmetric hash function. In [GKL+22] the authors introduce the hash function REINFORCED CONCRETE which is the first design to come up with a solution to this problem by introducing the novel $y = \text{BAR}(x)$ operation.

The Bar Function. The goal of BAR is to split a field element $x \in \mathbb{F}_p$ into smaller chunks which fit into lookup tables, performing lookups, and transforming the results of the lookups back to field elements while ensuring well-definition over $\mathbb{F}_p$. In other words, $y = \text{BAR}(x)$ needs to be invertible as required for security in SPNs. Thus, the authors define a decomposition function $\mathcal{D} : \mathbb{F}_p \to \mathbb{Z}_{s_1} \times \mathbb{Z}_{s_2} \times \cdots \times \mathbb{Z}_{s_n}$ such that $\mathcal{D}(x) = (x_1, x_2, \ldots, x_n)$ with

$$x = x_1 \cdot s_2 s_3 \cdots s_n, + x_2 \cdot s_3 s_4 \cdots s_n + \cdots + x_{n-1} \cdot s_n + x_n = \sum_{i=1}^{n} x_i \prod_{j>i} s_i$$

for $x_i < s_i$ and $\prod_{i=1}^{n} s_i > p$. Furthermore, the composition function $\mathcal{C} : \mathbb{Z}_{s_1} \times \mathbb{Z}_{s_2} \times \cdots \times \mathbb{Z}_{s_n} \to \mathbb{F}_p$ is defined to be the inverse of $\mathcal{D}$ and can be computed as

$$\mathcal{C}(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i \prod_{j>i} s_i \mod p.$$

Using this (de-)composition one can define Bar as $\text{Bar} : \mathbb{F}_p \to \mathbb{F}_p; x \mapsto \text{Bar}(x) = \mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(x)$ where $\mathcal{S}(x_1, x_2, \ldots, x_n) = (S_1(x_1), S_2(x_2), \ldots, S_n(x_n))$ is finally a lookup table layer where the tables are small enough to fit into ZK circuits and software implementations.

In Reinforced Concrete only one distinct lookup table is applied in Bar, i.e., $S(\cdot) = S_1(\cdot) = S_2(\cdot) = \cdots = S_n(\cdot)$. It is chosen in the following way. Let $(v_1, v_2, \ldots, v_n) = \mathcal{D}(p-1)$, then choose p' such that $p' \leq \min_{1 \leq i \leq n} v_i$. $S(\cdot)$ is then defined as

$$y_i := S(x_i) = \begin{cases} f(x_i) & \text{if } x_i < p', \\ x_i & \text{if } x_i \geq p', \end{cases}$$

where $f(x_i)$ is derived from a MiMC encryption which requires p' to be prime. In [GKL+22] the authors prove invertibility of Bar when constructed this way, which, informally spoken, stems from $S(\cdot)$ to be chosen to be invertible, $\mathcal{D}(\cdot)$ and $\mathcal{C}(\cdot)$ being the inverse functions of each other, and $\mathcal{D}(\cdot)$ uniquely defining a decomposition for each field element $x \in \mathbb{F}_p$.

The advantages of the Bar construction are the following. When Bar is defined as a polynomial over $\mathbb{F}_p$ the polynomial is dense and of large degree leading to strong security against algebraic attacks. Reinforced Concrete (as depicted in Figure 3.4) is constructed as a Hades where full layers are used to argue statistical security and the Bars layer, where Bar is applied to each element in the internal state, is used to argue algebraic security. Since Reinforced Concrete is defined for large prime fields of size $\log_2(p) \approx 256$, only one Bars layer is required for algebraic security of the whole design. This is a significant improvement in terms of required number of rounds, especially when compared to the Hades based Poseidon hash function, where ≈ 60 partial layers are required for algebraic security in the same setting. This leads to a huge improvement in plain hashing runtime (when lookups are used to implement $S(\cdot)$), while keeping ZK circuit sizes to be roughly the same (when the lookup argument is available in the proof system).

Unfortunately, this Bar construction also has some disadvantages. First, the decomposition is tedious to instantiate for different primes p. Second, the decomposition requires to compute a chain of modular reduction mod s_i which is in general a slow operation in plain. This is not a problem in large prime fields $\mathbb{F}_p$ since only one Bars layer is required for security. However, in smaller prime fields as used in, e.g., Plonky2 [Pol22a; Pol22b], where more Bars layers would be required for security, many more decompositions are needed, affecting plain hashing performance. Finally, since computing MiMC inside of $S(\cdot)$ is slow, plain hashing performance of Reinforced Concrete requires that $S(\cdot)$ is

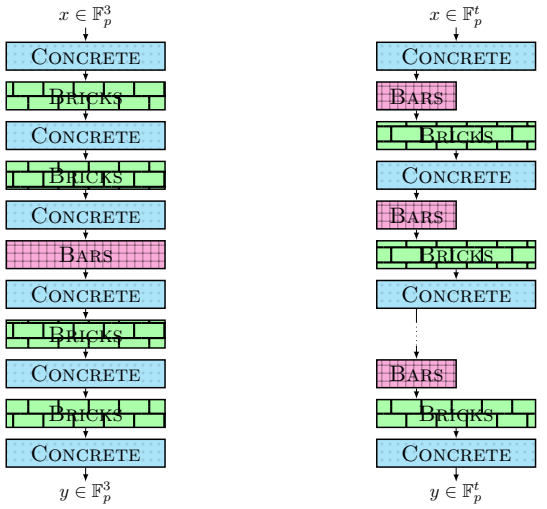


Figure 3.4: Structure of the Reinforced Concrete (left) and the Monolith (right) permutations, where we omit the round constant additions for simplicity and $t \in \{8, 12, 16, 24\}$ for Monolith (from [GKL+23]).

evaluated as a lookup table, making the design potentially susceptible to cache-based side channel attacks [Pag02; Ber05; OST06]. In Chapter 8 of this thesis we improve on Bars' drawbacks and instantiate the Monolith hash function (depicted in Figure 3.4), which features a generalization of Bar for smaller prime fields with faster decomposition and fast constant time implementation. As a result, Monolith is the first ZK-friendly hash function with a plain hashing time comparable to SHA-3, while also being more efficient inside a ZK proof compared to many instances of Poseidon.

We also want to mention the Tip5 [SLS+23] hash function, and the two additional instances Tip4 and Tip4′ [Sal23], which are the first hash functions to use the Bar idea from Reinforced Concrete for the smaller prime field $\mathbb{F}_p$ with $p = 2^{64} - 2^{32} + 1$. However, they keep the same structure of using repeated low-degree power maps (as used in MiMC) to instantiate the lookup tables, leading to some of the same disadvantages of Bar as discussed in this section. Furthermore, we show in this thesis that Monolith outperforms Tip5 both in terms of plain hashing performance and ZK circuit size, while also being a more generic design capable of being instantiated in different smaller prime fields.

3.2.6 Randomized Permutations

So far in this chapter we were only discussing improvements of LowMC and MiMC for MPC and ZK use cases. In this section, we will now focus on the design choices which made new symmetric primitives more efficient in HE use cases, mainly *hybrid homomorphic encryption* (HHE) which is often also referred

to as *transciphering*. The main observation which led to many improvements was the realization that a large class of attacks on symmetric ciphers require multiple plaintext-ciphertext pairs. To mitigate these attacks and, consequently, reduce the number of rounds and multiplicative depth of the cipher, designers came up with novel techniques to randomize the cipher based on a public *initialization vector* (IV).

As we will discuss in this section, this design approach leads to low-depth ciphers which are significantly more efficient in HE use cases compared to traditional ones, such as AES. However, they will not replace them in traditional use cases due to the impact on plain encryption runtime. Indeed, all of the designs discussed in this section are orders of magnitude slower for encryption in software compared to AES. The reason for that is that a *pseudorandom number generator* (PRNG), or an *extendable output function* (XOF), seeded with the IV is used to generate random bit permutations, linear layers, and/or round constants to achieve randomization. This PRNG/XOF is often instantiated using traditional primitives, such as AES or Shake128, adding the runtime of potentially many invocations of these primitives to the encryption time of the randomized ciphers.

FLIP [MJS+16] and FiLIP [MCJ+19]. The first stream cipher to randomize the keystream generation for HE use cases is FLIP. Its design approach, dubbed *filter permutator*, is the following. First a PRNG is seeded with an IV. Then, to generate one bit of the keystream, a new random bit permutation π_i is sampled from the PRNG which is used to permute the register storing the N-bit encryption key K. The result is fed into a boolean filtering function f, which compresses the permuted key into a keystream bit k_i. Thus,

$$k_i = f(\pi_i(K))$$

where π_i is a bit permutation which is pseudorandomly generated from the PRNG. While this novel design approach led to a stream cipher with an AND-depth of only 4, the design was shown to be vulnerable to guess-and-determine attacks in [DLR16]. Thus, in [MCJ+19] the authors introduced the concept of *improved filter permutators* and the stream cipher FiLIP which fixes the vulnerabilities of FLIP. The changes are two-fold. First the secret key in FiLIP is larger compared to FLIP and a pseudorandom subset of the key bits is chosen before the permutation π_i is applied. Secondly, a random constant addition, dubbed *whitening*, is applied before the filtering function f. Thus using FiLIP a keystream bit k_i is generated as

$$k_i = f(\pi_i(S_i(K)) \oplus w_i)$$

where $S_i(K)$ is a random subset of all key bits chosen by the PRNG, π_i is a random bit permutation chosen by the PRNG and w_i is a random bitvector sampled from the PRNG. Due to its structure to produce one keystream bit at a time and comparably low boolean gate count in its filtering function FiLIP is especially suited for being used in combination with the TFHE encryption scheme, as shown in, e.g., [HMR20; CHM+22; DGH+23].

Rasta [DEG+18]. Another approach to build a randomized stream cipher was introduced in [DEG+18]. They introduce the Rasta design strategy (as depicted in Figure 3.5), in which a permutation is applied to the encryption key, followed by a final key addition, to produce the keystream. Contrary to FiLIP, the permutation is built as an SPN that produces a keystream block of n bits at once, where n is the size of the symmetric key. The main novelty of Rasta is the way the permutation is constructed, more specifically the linear layers. In LowMC random invertible matrices are chosen during instantiation of the block cipher. Rasta takes this approach one step further and samples random invertible matrices during encryption from an XOF seeded with a nonce N and a block counter i. Furthermore, round constants are also sampled from the XOF. Thus, each keystream block is created from a different permutation and Rasta naturally resists attacks which use more than one plaintext-ciphertext pair.

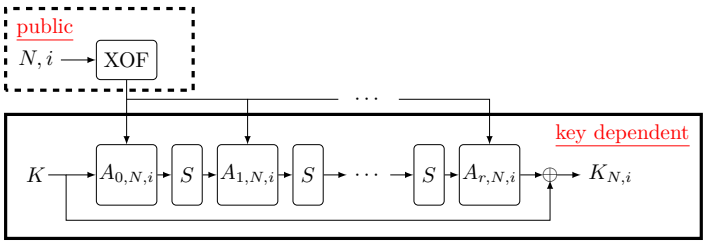


Figure 3.5: The r-round Rasta construction to generate the keystream $K_{N,i}$ for block i under nonce N with affine layers $A_{j,N,i}$ (from [DEG+18]).

Rasta uses the low-degree χ function from Keccak [Nat15] as non-linear layer which has an AND-depth of only 1. To further reduce the AND-depth of the cipher Rasta investigates the tradeoff between the number of rounds and the size of the internal state of the cipher which is equivalent to the keysize in Rasta. In traditional symmetric cryptography, the key is usually chosen to have λ bits where λ is the desired security level. Then a given round function is applied r times, where r is the number of rounds deemed secure by the designers (plus some security margin). In Rasta, where the main goal is to reduce the multiplicative depth, the size of the internal state is made larger instead of increasing the number of rounds. This has a similar effect on security and leads to unconventionally large internal states of 351 bits for a 6-round version of Rasta with an AND-depth of 6 and 525 bits for a 5-round version with an AND-depth of 5.

The structure of producing a keystream block instead of a single bit per permutation makes Rasta especially well suited for packing-friendly HE schemes [CIR22; DGH+23], such as BGV and BFV. It has also been the basis for many followup designs, such as Dasta [HL20] and Fasta [CIR22], and some ciphers we discuss in the next paragraphs.

The main improvements in Dasta and Fasta concern the linear layers. While in Rasta each invertible matrix is sampled from an XOF leading to slow plain encryption time, one invertible matrix is fixed in Dasta and the state is

permuted using predefined permutations. Thus, DASTA removes the requirement of having an XOF altogether leading to faster plain encryption time compared to RASTA while having the same state size, key size, number of rounds, and AND-depth. FASTA on the other hand pseudorandomly samples linear layers with high diffusion which have a structure optimized for fast packed evaluation for specific BGV parameter sets, showing significant improvements over RASTA when these parameter sets are available.

Regarding security, RASTA and DASTA have been analyzed in [LSM+21]. While the main instances of RASTA/DASTA remain secure, the authors showed that their attack successfully reduces the security margin to just one round. However, it has shown to (theoretically) break AGRASTA, the aggressive version of RASTA which was instantiated without security margin.

From Bits to Larger Prime Fields. So far in this section we discussed many symmetric ciphers which were proposed for HE use cases. However, these ciphers operate on plaintexts consisting of bits, while modern HE schemes can be instantiated to operate over larger integers, often prime field elements $\in \mathbb{F}_p$. Since it is not possible to translate a HE ciphertext encrypting elements $\in \mathbb{F}_p$ to ciphertexts encrypting bits and vice versa without bootstrapping, the usefulness of bitoriented ciphers for HE applications is limited. This was independently observed by many publications, including [HKC+20; CHK+21; CHM+22] and our work which is discussed in Chapter 6. While the trend changed to designing new symmetric primitives which naturally operate on elements in $\mathbb{F}_p$, the design principles discussed in this section stayed the same.

The first HE friendly cipher over $\mathbb{F}_p$ is MASTA [HKC+20] which is a direct translation of RASTA to the $\mathbb{F}_p$ setting, except a more efficient strategy in sampling invertible random matrices. Indeed, they observed that checking a random matrix for invertibility is expensive over larger prime fields and chose to directly sample random matrices instead. Their approach involves sampling a random polynomial in $m \in \mathbb{F}_p[X]/(X^t - \alpha)$ and translating it to an equivalent matrix which is then invertible by design. Even though the χ transformation is in general not invertible over $\mathbb{F}_p$ no other change is imposed by the authors of MASTA. While having a non-invertible non-linear layer is usually not a desirable property in symmetric designs, no attack exploiting χ in MASTA is known at the time of writing.

In this thesis we present our own HE friendly symmetric cipher over $\mathbb{F}_p$ using randomized linear layers, dubbed PASTA. We show that our novel techniques for faster linear layers and efficient non-linear ones outperform all presented symmetric ciphers when applied to HHE use cases in BGV and BFV. We refer to Chapter 6 for more details on PASTA.

Transciphering for Real Numbers. In many HE schemes, such as BGV, BFV, and TFHE, no issue arises when decrypting a symmetric ciphertext using a homomorphically encrypted symmetric key, transforming the symmetric ciphertext into a HE ciphertext. However, the approximate nature of CKKS ciphertexts,

in which noise is directly embedded into plaintexts, makes direct transciphering of symmetric ciphertexts using CKKS-encrypted symmetric keys impossible. In [CHK+21] the authors propose a framework to circumvent this problem. On a high-level their approach is simple. First, the symmetric encryption key is encrypted using BFV instead of CKKS. Then transciphering is performed to transform a symmetric ciphertext into a BFV ciphertext, followed by transforming the BFV ciphertext into a CKKS one. Thus, CKKS's noise embedding does not interfere with the correct nature of the symmetric decryption and HHE can be applied to HE use cases over real numbers.

In addition to the transciphering framework for CKKS the authors of [CHK+21] also propose a symmetric cipher over $\mathbb{F}_p$, dubbed Hera, which is optimized for HE use cases. Inspired by Rasta, Hera also involves a XOF to randomize the cipher. However, they do not randomize the linear layers, but apply randomization in a key schedule where they multiply each key-element with a pseudorandomly generated field element to improve plain performance compared to Rasta and Masta. Inspired by AES, Hera is built as a block cipher with a fixed state size of 16 field elements arranged as a 4×4 block, and uses the AES matrix $\mathrm{circ}(2, 3, 1, 1)$ in two linear layers to mix the rows and columns of the 4×4 state. Hera uses the well-established power maps $x \mapsto x^3$ as its non-linear layer. While leading to a fast plain performance in comparison to other HE friendly ciphers, we show in Chapter 6 that Pasta outperforms Hera as well when applied to HHE use cases in BFV and BGV.

In [HKL+22] the authors introduce the stream cipher Rubato, which improves on the Hera cipher in several ways. First, they replace the fixed state size of 16 words with a size of v^2 words and replace the AES matrix with a generic $v \times v$ MDS matrix, preferably a circulant matrix if possible. Secondly, they replace the power maps in the non-linear layer with a quadratic Feistel S-box as introduced in [DGH+23] (see Chapter 6). Finally, and arguably the most impactful improvement is the addition of random gaussian noise to the generated keystream. This noise improves the security argument of the cipher and allows Rubato to reduce the number of rounds and, consequently, the multiplicative depth, compared to Hera. On the other hand, this noise addition makes Rubato incompatible with BGV and BFV where exact decryption is required. For CKKS though, where the noise is already embedded in plaintexts, Rubato is usable and the preferred choice due to its low depth.

Transciphering for Groups and Rings. RLWE based HE schemes, such as BGV and BFV can be instantiated to work over plaintexts in $\mathbb{Z}_q$. However, since packing requires prime fields $\mathbb{F}_p$, HE schemes are usually instantiated using prime field plaintexts. Similar, most symmetric ciphers are defined over finite fields, either prime fields or binary extension fields, since it is easier to argue for security in fields compared to rings. An example can be seen with Rubato. While originally being defined over $\mathbb{Z}_q$ with arbitrary q, the ring instances (i.e., q not being prime) were broken shortly after publication in [GAH+23].

To accommodate the improvements in the newest versions of the TFHE scheme, the authors of [CHM+22] introduce the stream cipher Elisabeth-4 built with the *group filter permutator* strategy, a translation of the improved filter permutator to the group setting. In other words, the structure of Elisabeth follows the structure of FiLIP to produce a keystream as

$$k_i = f(\pi_i(S_i(K)) + w_i)$$

where $S_i(K)$ is a random subset of all key elements (which are $\in \mathbb{G}$) chosen by a PRNG, π_i is a random permutation (re-ordering) chosen by the PRNG and w_i is a random vector of group elements sampled from the PRNG. While Elisabeth-4, defined over $\mathbb{G} = \mathbb{Z}_{2^4}$, is tailored to the requirements of TFHE and shown to significantly outperform FiLIP in this HE scheme [CHM+22], the design is vulnerable to linearization attacks.[1]

3.2.7 Fast Stream Ciphers

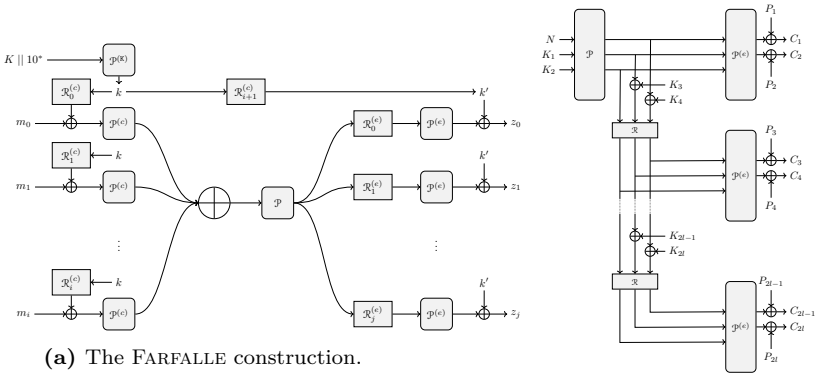
In this chapter we covered many design approaches on how to design symmetric primitives which are efficient when used in combination with PETs. However, sometimes symmetric primitives are designed without PET use cases in mind but turn out to be efficient in these scenarios as well. We are looking at two examples in this section.

Trivium [Can06] and Kreyvium [CCF+16]. Trivium is a well-established and standardized [ISO12] stream cipher introduced for the eSTREAM[2] project which aimed to find new stream ciphers which are efficient in software and hardware. Since Trivium was only defined for a security level of 80 bits, Kreyvium was published in [CCF+16] as a 128-bit security variant of Trivium. In this work the authors also identified the potential of Kreyvium for HE use cases in BFV and BGV. However, as later work shows, the low gate-count of Trivium and Kreyvium seems to be especially suitable in combination with TFHE [DGH+23; BOS23]. Furthermore, due to the low gate count and comparably small AND-depth Kreyvium has been used as a PRF in combination with MPC protocols based on garbled circuits [CCD+20] as well.

Farfalle [BDH+17] and Ciminion [DGG+21]. Farfalle (as depicted in Figure 3.6a) is a versatile mode of operation introduced for building fast PRFs with inputs and outputs being of arbitrary length. It consists of a *compression layer* and an *expansion layer* instantiated with a set of permutations $(\mathcal{P}^{(c)}, \mathcal{P}, \mathcal{P}^{(e)})$, and rolling functions $(\mathcal{R}_i^{(c)}, \mathcal{R}_i^{(e)})$ to mask and update the inputs of the permutations. The advantages of Farfalle compared to, e.g., keyed sponge modes of operation such as the duplex mode [DMA17] are manyfold. On one hand, Farfalle has the inherent possibility to parallelize the permutations in the compression and

[1] At the time of writing, the paper describing the attack is not yet public, but has been announced at the FSE 2023 rump session https://youtu.be/16SZg8SnJqk?t=1684.

[2] https://www.ecrypt.eu.org/stream/project.html



(a) The Farfalle construction.

(b) The Ciminion PRF.

Figure 3.6: Farfalle and Ciminion (modified from [GØS+23]).

expansion layers which is not possible for the sponge mode of operation. On the other hand, in Farfalle an adversary never learns both the input and output of a permutation call. Thus there are fewer requirements on the security of the individual permutations in Farfalle. As a concrete demonstration the authors of [BDH+17] instantiate the PRF Kravatte using the Keccak-p permutation with only 6 rounds, compared to SHA-3 with 24 rounds and KangarooTwelve [BDP+18] using 12. For the sake of completeness we point out that an early version of Kravatte has successfully been broken in [CFG+18], however, the attack was mitigated in [BDH+17] by using non-linear rolling functions.

In [DGG+21] the authors realize that an adapted version of Farfalle allows to significantly reduce the multiplicative complexity of an encryption scheme and propose the PRF Ciminion. As depicted in Figure 3.6b, Ciminion gets rid of the compression layer and only allows for a fixed size input. Furthermore, it instantiates a sponge-based key schedule, uses the first two round keys (and a nonce N) as input of the cipher, adds round keys before each rolling function $\mathcal{R}$, and truncates one element at the output of each permutation $\mathcal{P}^{(e)}$ in the expansion layer. The result is a PRF where the permutation $\mathcal{P}^{(e)}$ can be instantiated with a cheap permutation, since an attacker does not know its inputs (as in Farfalle). Consequently, the more expensive initial permutation $\mathcal{P}$ which only gets computed once, gets amortized when encrypting large amounts of data making Ciminion especially efficient in such a setting. By using the Toffoli gate [Tof80] in the non-linear layers, Ciminion achieves a smaller multiplicative complexity compared to, e.g., HadesMiMC and Rescue making it an efficient PRF for MPC applications.

In this thesis, however, we identify a weakness in the Ciminion design, namely the presence of the heavy (in terms of multiplicative complexity) key schedule. As we elaborate in Chapter 7 this key schedule prevents Ciminion from being efficiently used in a large class of MPC use cases. Thus, we introduce the Megafono design strategy and the PRF Hydra which gets rid of the key schedule altogether. However, Hydra keeps the same efficiency as Ciminion

even when ignoring the cost of the key schedule. We refer to Chapter 7 for more details on our construction.

4

Contributions of this Thesis

As outlined in Chapter 1, this thesis on one hand proposes efficient protocols which combine multiple datasets while ensuring their security and privacy, and on the other hand proposes novel and efficient building blocks to speed up common PET use cases. In this chapter, we discuss the main contributions of this thesis in more detail, and finally conclude Part I of this thesis in Section 4.7.

4.1 Private Covid-19 Heatmap [BBH+22]

In many practical applications utility may be gained by combining multiple datasets to get more useful insights into a problem. For example, during the recent Covid-19 pandemic, combining the location data of infected individuals had the potential of helping to contain the spread of the disease. The main issue with combining datasets, however, is maintaining privacy while getting utility, especially when sensitive data, such as location or health data, is involved. During the Covid-19 pandemic, a lot of technical solutions focused on private contact tracing applications (e.g., [CTV20; CFG+20; GA20]) which should notify people if they have been close to infected individuals while hiding the identity of said people. In this work we designed a different kind of protocol which is not aimed at contact tracing. Our idea was to create a heatmap of aggregated location data of infected individuals by combining the location data gathered from mobile network operators with the data of health authorities who know who is infected. This heatmap does not notify people if they have been in contact with infected individuals, but has the potential to track the spread of the disease itself and help policy makers in applying targeted measures to hotspots and containing the spread.

To help convince government officials to deploy the final protocol, we focused on designing it to be as simple as possible while not compromising on the security and privacy of the involved datasets, as well as keeping good performance. Our goal, thus, was to achieve a 1-round query-response protocol, that in the worst case could be executed by passing around USB drives where encrypted data is stored. To this end, we employed homomorphic encryption (HE) as the main primitive to protect the datasets and constructed a client-server architecture, where health authorities transmit a HE-encrypted, one-hot encoded vector to mobile network operators, where a 0 at index i indicates that person i is not infected, while a 1 indicates infection. The mobile network operator then can create the desired (encrypted) heatmap by multiplying this vector with a location matrix, where the cell $x_{i,j}$ contains the location of person i at cell tower j.

While this approach protects both datasets, two problems were still left to be solved. On one hand, HE only provides semi-honest security and health authorities can try to infer location data by sending malicious query vectors. On the other hand, the final heatmap can leak location data even from honest queries, e.g., when individuals from sparsely populated regions are infected. To solve these issues while keeping the 1-round structure of the protocol, we applied invalidation techniques which randomized the final heatmap if the query was found to be malicious, as well as differential privacy (DP) [Dwo06] to minimize the contribution of any individuals location data to the final heatmap. The invalidation technique, thereby, was designed to provide high statistical security, while also being computable by a low-degree polynomial to make computation under an SHE scheme more efficient.

We implemented this protocol using the BFV [Bra12; FV12] HE scheme and show that it can create a heatmap for 8 million individuals (e.g., the population of smaller countries such as Austria) in 70 minutes on a reasonable hardware despite using slow cryptographic primitives such as HE.

Author's Contribution. The author contributed to this work by collaborating in designing an efficient, useful, and privacy preserving protocol intended to assist with containing the disease. Furthermore, the author implemented the protocol using the BFV HE scheme, and performed all practical evaluations of the protocol.

Impact and Future Work. The final protocol was featured in Austrian newspapers,[1] but not deployed in practice. Thus, future research directions could involve the analysis of the legal and ethical challenges for a real world deployment. In other words, one could investigate whether the application of PETs, such as HE and DP, legally enables governments to process large amounts of sensitive data (e.g., research similar to [HR22]), as well as which additional ethical considerations, such as opt-in processes, can and should be employed.

[1] https://www.derstandard.at/story/2000126812003/tu-graz-kombiniert-gesundheits-und-bewegungsdaten-datenschutzkonform

4.2 Hybrid Homomorphic Encryption and Design of Pasta [DGH+23]

Homomorphic encryption (HE), with its ability to directly compute functions on ciphertexts, has the potential to solve many applications involving the sharing of sensitive data. However, HE has some severe disadvantages as well. On one hand, applying HE to any use case leads to a slow down of several orders of magnitude. On the other hand, HE suffers from ciphertext expansion, meaning that HE ciphertexts are significantly larger than their non-encrypted counterparts. This expansion has a negative impact on the amount of data transferred from the data provider to the party performing the computation and can lead to a sharp increase in total runtime, especially if data providers are embedded devices with limited bandwidth connections. In this work we focus on hybrid homomorphic encryption (HHE) [NLV11], a technique designed to reduce the ciphertext expansion in HE applications (see Section 3.1.1). In theory any symmetric encryption scheme, such as the well known AES [DR00; DR02], could be used in HHE, however, the internal structure of traditional ciphers is not optimized for the cost metrics of modern HE schemes and would thus lead to inefficient constructions. To this end, many new symmetric ciphers (e.g., in [ARS+15; MCJ+19; DEG+18; CCF+16]) have been proposed in the literature which are optimized for decryption under HE mainly by minimizing the multiplicative depth of the decryption circuit.

However, before our work the concrete effects of applying HHE to a use case were not yet understood due to the lack of actual analysis and public implementations. Hence, we wrote a framework for benchmarking 8 different symmetric ciphers in three popular HE libraries and benchmarked them for two different use cases. We also observed that many symmetric ciphers were proposed for binary plaintexts and thus were not suited for a large class of use cases, namely use cases over larger integers for which the used HE scheme is not instantiated over $\mathbb{F}_2$. To this end, our two use cases consist of a small, low-depth use case to highlight the inefficiencies of $\mathbb{F}_2$ ciphers, as well as a larger depth use case to investigate the actual performance of HHE when employing different $\mathbb{F}_p$ ciphers.

Furthermore, we proposed a new symmetric cipher, dubbed Pasta, a cipher natively operating over $\mathbb{F}_p$ with primes $p > 2^{16}$. Pasta's inner structure, thereby, is designed to be packing-friendly and make use of the inner rotation structure of HE ciphertext in BFV [Bra12; FV12] and BGV [BGV14] (see Section 2.2.2) to speed up decryption under these HE schemes. Furthermore, we carefully chose two types of non-linear layers which allow us to reduce the required number of rounds to only three, minimizing the multiplicative depth of the decryption circuit, while keeping the inner state of the cipher small enough for fast performance.

As a result, we show that Pasta outperforms the closest competitor by a factor of up to 6 when benchmarking the two use cases under HHE (instantiated with the BFV and BGV HE schemes) using our framework.

Author's Contribution. The author contributed to this work by implementing the whole benchmarking framework, realizing the shortcomings of binary ciphers,

working on the design of the round function of Pasta to be as efficient as possible in HHE use cases, and performing all performance evaluations in the paper.

Impact and Future Work. After our publication, many new symmetric ciphers for HHE where benchmarked against Pasta using our implementation framework [CIR22; CHM+22]. Furthermore, our implementation of the Kreyvium [CCF+16] stream cipher using the initial TFHE [CGG+20] library became the baseline for the benchmarks in [BOS23], in which the authors show roughly a factor 100 improvement by instead using the updated TFHE-rs[1] library with more capabilities and employing multithreading with 128 virtual CPUs. Additionally, the authors of Rubato [HKL+22], a HE-friendly cipher following the structure of Hera [CHK+21], decided to remove the cubic power map non-linear function used in Hera in favor of a quadratic-Feistel-based one which was first employed in this context by Pasta. Finally, Pasta was chosen as the symmetric cipher in the followup work [BFM22] which used HHE as the main building block in a protocol that allows analysts to perform functions over data collected from multiple clients.

While Pasta is highly efficient for decryption under BFV/BGV, one can consider the following topics for future research. First, Pasta is designed to be efficient in combination with SHE schemes due to its minimized multiplicative depth. However, once bootstrapping becomes more efficient in BFV/BGV, the cost metric might change, e.g., similar to TFHE where the main cost metric has become reducing the total gate count. Thus, future research might introduce new ciphers optimized for this potential cost metric update due to faster bootstrapping. Furthermore, since Elisabeth [CHM+22] is, at the time of writing, the only cipher designed with the capabilities of the newest version of the TFHE scheme in mind, which however got broken by a linearization attack (see Section 3.2.6), future research may focus on introducing a new cipher for this HE scheme.

4.3 Design of Hydra [GØS+23]

The realization that many multi-party computation (MPC) use cases can profit from symmetric ciphers has lead to the introduction of new encryption schemes, mainly optimized for having a small total number of multiplications. Examples include LowMC [ARS+15] and Rain [DKR+22] for private set intersection [KRS+19] and MPC-based post quantum signature schemes like Picnic [CDG+17], or HadesMiMC [GLR+20] and Ciminion [DGG+21] for encryption using a secret shared symmetric key for distributed data storage, data transportation in delegated MPC [GRR+16], or in virtual hardware security modules.[2] The stream cipher Ciminion, due to its Farfalle [BDH+17] based structure, seems to be especially well suited for the latter use cases, even more so when large amounts of data need to be encrypted. However, it has the crucial disadvantages that the security argument of Ciminion requires the presence of an expensive

[1] https://github.com/zama-ai/tfhe-rs
[2] https://www.fintechfutures.com/files/2020/09/vHSM-Whitepaper-v3.pdf

(in terms of number of multiplications) key schedule, which has to be computed in MPC for a secret shared key. Thus, Ciminion is only efficient in those use cases if the round keys do not have to be computed in MPC which is not the case for a large number of use cases. Consequently, the goal of this publication was to introduce a new mode of operation, that allows to build a stream cipher with the same efficiency as Ciminion in MPC use cases, while not relying on an expensive key schedule for security.

Thus, we introduce the Megafono mode of operation and, as a concrete instantiation, the stream cipher Hydra. Megafono, thereby, keeps a similar structure compared to Ciminion by employing two different permutations: An expensive one, $\mathcal{B}$, which has to be computed once in the beginning, and a cheaper one, $\mathcal{H}_\mathtt{K}$, which is computed repeatedly to produce the actual keystream. We especially focused on reducing the cost of the $\mathcal{H}_\mathtt{K}$ permutation in MPC, since the cost of $\mathcal{B}$ gets amortized for encrypting large amounts of data. Thus, for large plaintexts the cost will be proportional to the cost of $\mathcal{H}_\mathtt{K}$. To reduce its cost, we instantiate $\mathcal{B}$ to resemble a pseudorandom permutation (PRP) such that the input to the $\mathcal{H}_\mathtt{K}$ permutations is (computationally) unknown and uncontrollable by an attacker, limiting the possible attack vectors which can be applied to the $\mathcal{H}_\mathtt{K}$ permutations. Furthermore, we introduce a novel technique to artificially increase the size of the state before $\mathcal{H}_\mathtt{K}$ for free to further strengthen the resistance of $\mathcal{H}_\mathtt{K}$ against algebraic attacks. Finally, we make heavy use of different variations of the Lai-Massey [LM90] function in Hydra's non-linear layers, combined with efficient linear layers chosen to break known subspace trails occurring in Lai-Massey constructions [Vau99], to minimize the number of multiplications in each round.

As a result, Hydra does not require any key schedule for security while still being computable with a smaller number of multiplications compared to Ciminion, even when ignoring the multiplications in the key schedule. To show this performance advantage, we implement Hydra in the MP-SPDZ [Kel20] MPC library and benchmark it against Ciminion, HadesMiMC, Rescue [AAB+20], GMiMC [AGP+19], and MiMC [AGR+16]. Concretely, we report the performance for encrypting plaintexts consisting of 8 to 128 field elements, each of size 128 bits, using a secret-shared symmetric key and the SPDZ [DPS+12] MPC protocol instantiated with two parties in a LAN network environment. These benchmarks show, that Hydra outperforms Ciminion even when ignoring the cost of the key schedule, while significantly outperforming the other competitors, with the advantage growing for larger plaintext sizes.

Author's Contribution. The author contributed to this work by identifying the shortcomings of Ciminion's key schedule, working on the design of the round functions to be efficient in MPC, implementing Hydra in the MP-SPDZ library and performing the practical evaluations.

Impact and Future Work. In our work, we mainly focused on optimizing the performance of the $\mathcal{H}_\mathtt{K}$ permutation since the expensive $\mathcal{B}$ permutation gets

amortized for large amounts of data. In the followup work [Gra23], the author used the SI-lifting function $F(x_1, x_2) = x_1^2 + x_2$ to improve the performance of the $\mathcal{B}$ permutation to introduce Hydra++. Furthermore, in our work we introduce a generalization of the Lai-Massey function and use it to instantiate the different non-linear layers in Hydra. In [Gra22], the author expands on this generalization and finally introduces a new type of non-linear layer, dubbed `Amaryllises`.

While we designed Hydra to minimize the number of multiplications when encrypting potentially large amounts of data under MPC, the authors of [GM23] focus on designing a MPC-friendly PRF with minimized multiplicative depth. They benchmark their construction, which is based on the LWE hardness assumption, against our implementations of multiple PRF's in the MP-SPDZ library. As a result, they show that their construction only slightly outperforms Hydra in a network setting without latency when encrypting only two field elements (note that Hydra outputs 8 at once), while requiring significantly more communication between the MPC parties.

As discussed in this thesis, many symmetric ciphers were proposed to natively operate over large fields $\mathbb{F}_p$. However, some MPC protocols natively operate over rings $\mathbb{Z}_{2^k}$ to improve performance on Desktop CPUs (see Section 2.3.2). However, since designing ciphers over rings is in general more difficult compared to fields (see, e.g., the discussion in Section 3.2.6), no MPC-friendly ciphers over rings were proposed in the literature. Consequently, investigating efficient and secure proposals for such ciphers might be an interesting avenue for future research.

4.4 Design of Monolith [GKL+23]

One of the core components in many zero knowledge use cases are cryptographic hash functions. While traditional hash functions, such as SHA-3 [Nat15; BDP+11], could be used, their internal structure makes them inefficient in combination with modern zero knowledge proof systems. To this end many new hash functions optimized for these use cases have been proposed in the literature, starting with MiMC [AGR+16], and recently including Poseidon [GKR+21], Rescue [AAB+20; SAD20], Griffin [GHR+23], and many more. While they achieve a proof system performance orders of magnitude faster compared to SHA-3, their plain performance is significantly slower. However, in some cases performance depends on native hashing alongside proof system performance. Concretely, in recursive STARKs [COS20] at each recursion step a prover computes a Merkle tree over the witness data and then proves some tree openings in ZK. Consequently, when using the above mentioned ZK-friendly hash functions a huge part of prover computations has to be spent on native hashing. Thus introducing a new hash function which is both efficient inside the proof system, as well as for native hashing, is a huge research gap which we address in this work.

To address the problem of slow native hashing performance, researchers have introduced the Reinforced Concrete [GKL+22] hash function (see Section 3.2.5) which leverages lookup arguments to decrease the plain evaluation

runtime while keeping proof system performance reasonable. The main novelty in Reinforced Concrete was the introduction of the Bar round function, where large field elements in $\mathbb{F}_p$ are first decomposed into smaller ones in $\mathbb{F}_{p'}$, then table lookups are performed and the results are combined to elements in $\mathbb{F}_p$ again. While this approach has the advantage of providing strong algebraic security, it also has some drawbacks. First, one has to guarantee, that the result of applying Bar is still a valid field element, while also being able to efficiently constrain that decompositions do not overflow mod p inside the proof system. These two problems make it difficult to generalize Bar for different prime fields. Furthermore, by decomposing into many field elements and applying repeated power maps (as done in Reinforced Concrete and Tip5 [SLS+23], the only other lookup-based ZK-friendly hash function at the time of writing), native hashing performance crucially relies on table lookups which are known to be prone to cache-based side channel attacks [Pag02; Ber05; OST06].

In this work we address these shortcomings and introduce the Monolith family of hash functions. Our main focus lies on small prime fields, such as the 64-bit Goldilocks field identified by $p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$, as well as the 31-bit Mersenne prime field identified by $p_{\texttt{Mersenne}} = 2^{31} - 1$. These fields come with a structure that allows efficient modular reductions for fast performance on CPUs, which is why they have recently been used in the FRI based ZK proof systems Plonky2 [Pol22a; Pol22b] and Plonky3 [Pol23]. Furthermore, these primes allow for cheap constraints preventing the prover to cheat by letting decompositions overflow mod p inside ZK proof systems. We leverage these prime fields to, contrary to Reinforced Concrete and Tip5, decompose field elements into elements in $\mathbb{F}_2^k$ instead of smaller prime field elements in $\mathbb{F}_{p'}$. Then, we apply χ-like functions [Dae95] over $\mathbb{F}_2$ with simple requirements for guaranteeing that the outputs are still valid $\mathbb{F}_p$ elements. This style of decomposition has several advantages. First, since χ-like functions are easily vectorizable, we do not have to explicitly decompose field elements and can apply χ directly on $\mathbb{F}_p$ elements. Thus, plain performance does not rely on lookup tables and our design naturally resists cache-based side channel attacks. Furthermore, inside the proof system we can rely on the lookup argument to get fast prover performance. Finally, this approach also provides strong algebraic security, similar to the original Bar function. In our work, we generalize this new style of Bar layer, dubbed Kintsugi, for any fields with $p = 2^n - 2^\eta + 1$ and $p = 2^n - 1$.

Combined with fast circular MDS matrices as linear layers and a low-degree Type-3 Feistel [ZMI89] non-linear layers, we use Kintsugi to instantiate Monolith. We implement Monolith both natively, as well as in the Plonky2 proof system, and show that it is the first ZK-friendly hash function with constant time native hashing performance being on par with SHA-3. Furthermore, it is more efficient inside the proof system compared to many competitors, such as Poseidon and Tip5.

Author's Contribution. The author contributed to this paper by working on the design of the round function of Monolith to be fast in plain and inside the

proof systems, by implementing Monolith natively, helping with the Plonky2 implementation, performing the practical performance evaluations and working on the proof system performance estimates.

Impact and Future Work. While this work was only recently put on Eprint and is not yet formally published, it has already been integrated independently into the Plonky3 codebase.[1] Furthermore, the Plonky2 extension is available on `crates.io`.[2] Finally, this work was already presented at the ZK Summit 10 by the author.[3]

Since the area of zero knowledge is a fast moving field with regular groundbreaking new discoveries, future research inevitably involves finding new hash functions optimized for the ever changing cost metrics of new proof system proposals. As a concrete example, the new Lasso [STW23] lookup argument allows for larger lookup tables for decomposable circuits and might pose as a new target for designing an optimized hash function.

4.5 MPC Public Key Accumulators [HKR+21]

Cryptographic accumulators allow an efficient representation of a finite set and are an essential part in many cryptographic protocols, such as certificate transparency [Lau14], anonymous credentials [CL02], group and ring signatures [LLN+16; DRS18], and anonymous cash [MGG+13]. Looking at these examples, one can observe that especially Merkle tree [Mer89] accumulators are deployed in practice, often built from arithmetization oriented hash functions when paired with zero knowledge proofs. However, Merkle trees also have some disadvantages. On one hand, they are expensive to compute if the used hash function has a slow plain performance (e.g., when early arithmetization friendly hash functions such as Rescue [AAB+20] are used). On the other hand, witnesses which prove that a specific set element is part of the accumulator do not have a constant size and scale logarithmically in the size of the set. An alternative solution for zero-knowledge friendly accumulators are public-key accumulators, such as the q-SDH one [DHS15]. While these accumulators have the advantage of having constant size witnesses, they suffer from a different problem. They involve trapdoors which when known allow for efficient accumulator modifications (e.g., revoking elements from the set), but also allow to break the security guarantees of the whole scheme. Thus, they involve a trusted setup after which the trapdoor is required to be forgotten. However, accumulator modifications become inefficient without the knowledge of the trapdoor. In this paper we cast the q-SDH accumulator into the multi-party computation (MPC) setting, by splitting the trust of knowing the trapdoor to multiple accumulator managers. Consequently, accumulator updates can efficiently be computed by using the secret-shared trapdoor inside an MPC protocol, leading to significant performance advantages compared to

[1] https://github.com/Plonky3/Plonky3/tree/main/monolith
[2] https://crates.io/crates/plonky2_monolith
[3] https://www.zksummit.com/zksummit-10

a trapdoorless standard q-SDH accumulator. Thereby, we proof security of our construction in the UC-framework [Can01] if a UC-secure MPC protocol is used, while inheriting the security guarantees of said protocol. Thus, one can instantiate our MPC-q-SDH accumulator with different MPC protocols, such as SPDZ [DPS+12] and Shamir secret sharing [Sha79] to inherit their properties, e.g., semi-honest or malicious security with an honest or dishonest majority.

We implemented and benchmarked our construction in the MP-SPDZ [Kel20] MPC library, concretely using semi-honest and malicious variants of SPDZ and Shamir secret sharing for two to five parties in LAN and WAN network environments, which confirm the performance advantage compared to the trapdoorless alternative. This flexibility, performance, and constant witness size allows one to use our MPC-q-SDH accumulator as a viable alternative to Merkle trees in use cases where multiple accumulator managers are present. Concretely, we envision it being used in the Sovrin [KL16] distributed credential system, as well as part of the certificate transparency ecosystem following the protocol in [KOR19]. These two protocols already include multiple non-colluding servers and are, therefore, well suited to integrate accumulators based on MPC.

Author's Contribution. The author contributed to this work by first translating the q-SDH accumulator into the MPC setting, implementing the result in the MP-SPDZ library (including extending the library with the relic[1] library for elliptic curve operations[2]), and performing all performance evaluations in the paper.

Impact and Future Work. While this work was never used in practice, it has been mentioned as an viable option for casting accumulators into a decentralized setting in, e.g., [SCP+22; AKM+21]. Furthermore, it serves as the baseline for the improvements proposed in [JLM22], which adds the anonymity property to accumulator updates and witness verification.

Regarding future work, we see the applications of our accumulator to a real-world protocol, such as Sovrin or certificate transparency, as the main open point. This application would allow an in-depth comparison of our MPC-q-SDH accumulator to traditional Merkle tree accumulators regarding performance, security models, and practical implications.

4.6 Implementation Frameworks

As outlined in this section, the author's contribution to all publications include, among others, practical implementation frameworks. We want to point out that all these frameworks are publicly available. These include a full benchmarking framework for ciphers in hybrid homomorphic encryption, implementations of symmetric encryption in the MP-SPDZ MPC library, an implementation zoo

[1] https://github.com/relic-toolkit/relic
[2] This extension can be found at https://github.com/IAIK/MPC-Accumulator.

comparing ZK-friendly hash functions for plain hashing and zero knowledge use cases, a private heatmap protocol for disease analysis, privacy-preserving machine learning using CKKS, threshold public key accumulators in MP-SPDZ, and an FPGA implementations of LowMC and Picnic. These frameworks can be found at https://github.com/rw0x0/PhD.git.

4.7 Conclusion

In this thesis, we introduced a new protocol for generating a heatmap for tracking the spread of infectious diseases while preserving the privacy of all involved datasets, such as location data and health records. Furthermore, we introduced new symmetric ciphers and hash functions optimized for use cases which require to evaluate them under privacy enhancing technologies (PETs), such as homomorphic encryption (HE), multiparty computation (MPC), and zero knowledge (ZK) proofs. Finally, we introduced multiparty public key accumulators, which can serve as the replacement of Merkle trees, a common building block often also associated with ZK-friendly hash functions, in decentralized use cases.

Since the application of HE, MPC and ZK to any use case still suffers from high performance penalties, the design of efficient protocols and building blocks, such as symmetric primitives, requires deep knowledge and understanding of the capabilities and cost metrics of these PETs. In this thesis we demonstrate this knowledge by carefully optimizing our hash functions and encryption schemes to specific PETs which often also requires introducing novel building block (e.g., Kintsugi) and modes of operations (e.g., Megafono). Furthermore making the resulting protocols secure requires the careful consideration of the concrete guarantees one can get from the specific PETs. As example we show in our heatmap protocol, that techniques from three different PETs are required to achieve the desired security guarantees. Finally, we demonstrate the efficiency of all our protocols and building blocks compared to previous state-of-the-art by providing practical implementation frameworks.

On the same note we want to highlight that our proposals are very recent and none of our designs have been standardized. Thus, as is the case for every new cryptographic protocol and primitive, trust in the security, efficiency, and usefulness of our proposals has yet to be established by the community of researchers and practitioners of PETs. However, with this thesis we contributed to laying the groundwork for future real-world PET applications, as well as standardization projects, such as one on multiparty threshold schemes planned by NIST,[1] which would naturally fit to several of our designs.

[1] https://csrc.nist.gov/pubs/ir/8214/c/ipd

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols." In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 1–45.

[ACG+19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. "Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC." In: *ASIACRYPT (3)*. Vol. 11923. Lecture Notes in Computer Science. Springer, 2019, pp. 371–397.

[ACN13] Tolga Acar, Sherman S. M. Chow, and Lan Nguyen. "Accumulators and U-Prove Revocation." In: *Financial Cryptography*. Vol. 7859. Lecture Notes in Computer Science. Springer, 2013, pp. 189–196.

[AD18] Tomer Ashur and Siemen Dhooghe. "MARVELlous: a STARK-Friendly Family of Cryptographic Primitives." In: *IACR Cryptol. ePrint Arch.* (2018), p. 1098.

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority." In: *CCS*. ACM, 2016, pp. 805–817.

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. "Feistel Structures for MPC, and More." In: *ESORICS (2)*. Vol. 11736. Lecture Notes in Computer Science. Springer, 2019, pp. 151–171.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity." In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 191–219.

[AKM+21] Andreas Abraham, Karl Koch, Stefan More, Sebastian Ramacher, and Miha Stopar. "Privacy-Preserving eID Derivation to Self-Sovereign Identity Systems with Offline Revocation." In: *TrustCom*. IEEE, 2021, pp. 506–513.

[AMT22] Tomer Ashur, Mohammad Mahzoun, and Dilara Toprakhisar. "Chaghri - A FHE-friendly Block Cipher." In: *CCS*. ACM, 2022, pp. 139–150.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. “Ciphers for MPC and FHE.” In: *EUROCRYPT (1)*. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 430–454.

[AST+22] Miguel Ambrona, Anne-Laure Schmitt, Raphael R. Toledo, and Danny Willems. “New optimization techniques for PlonKâ€™s arithmetization.” In: *IACR Cryptol. ePrint Arch.* (2022), p. 462.

[Bai20] Jeeyun (Sophia) Baik. “Data privacy against innovation or against discrimination?: The case of the California Consumer Privacy Act (CCPA).” In: *Telematics Informatics* 52 (2020), p. 101431.

[BBC+22] Subhadeep Banik, Khashayar Barooti, Andrea Caforio, and Serge Vaudenay. “Memory-Efficient Single Data-Complexity Attacks on LowMC Using Partial Sets.” In: *IACR Cryptol. ePrint Arch.* (2022), p. 688.

[BBC+23] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. “New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: ttAnemoi Permutations and ttJive Compression Mode.” In: *CRYPTO (3)*. Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 507–539.

[BBD+20] Subhadeep Banik, Khashayar Barooti, F. Betül Durak, and Serge Vaudenay. “Cryptanalysis of LowMC instances using single plaintext/ciphertext pair.” In: *IACR Trans. Symmetric Cryptol.* 2020.4 (2020), pp. 130–146.

[BBH+18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Fast Reed-Solomon Interactive Oracle Proofs of Proximity.” In: *ICALP*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 14:1–14:17.

[BBH+18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Scalable, transparent, and post-quantum secure computational integrity.” In: *IACR Cryptol. ePrint Arch.* (2018), p. 46.

[BBH+19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. “Scalable Zero Knowledge with No Trusted Setup.” In: *CRYPTO (3)*. Vol. 11694. Lecture Notes in Computer Science. Springer, 2019, pp. 701–732.

[BBH+22] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. “Privately Connecting Mobility to Infectious Diseases via Applied Cryptography.” In: *PoPETs* 2022.4 (2022), pp. 768–788.

[BBSG+22] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. "Publicly Verifiable Zero-Knowledge and Post-Quantum Signatures from VOLE-in-the-Head." In: *CRYPTO (5)*. Vol. 14085. Lecture Notes in Computer Science. Springer, 2022, pp. 581–615.

[BBSG+23] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. *FAEST: Algorithm Specifications (Version 1.1)*. 2023. URL: https://faest.info/faest-spec-v1.1.pdf.

[BBV+21] Subhadeep Banik, Khashayar Barooti, Serge Vaudenay, and Hailun Yan. "New Attacks on LowMC Instances with a Single Plaintext/Ciphertext Pair." In: *ASIACRYPT (1)*. Vol. 13090. Lecture Notes in Computer Science. Springer, 2021, pp. 303–331.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. "Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems." In: *CRYPTO (3)*. Vol. 12172. Lecture Notes in Computer Science. Springer, 2020, pp. 299–328.

[BCP23] Clémence Bouvier, Anne Canteaut, and Léo Perrin. "On the algebraic degree of iterated power functions." In: *Des. Codes Cryptogr.* 91.3 (2023), pp. 997–1033.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs." In: *TCC (B2)*. Vol. 9986. Lecture Notes in Computer Science. 2016, pp. 31–60.

[BDH+17] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. "Farfalle: parallel permutation-based cryptography." In: *IACR Trans. Symmetric Cryptol.* 2017.4 (2017), pp. 1–38.

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures." In: *J. Cryptogr. Eng.* 2.2 (2012), pp. 77–89.

[BDP+08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction." In: *EUROCRYPT*. Vol. 4965. Lecture Notes in Computer Science. Springer, 2008, pp. 181–197.

[BDP+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Note on zero-sum distinguishers of Keccak-f*. 2011.

[BDP+18] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. "KangarooTwelve: Fast Hashing Based on Keccak-p." In: *ACNS*. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 400–418.

[BDS+22] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. "MOTION - A Framework for Mixed-Protocol Multi-Party Computation." In: *ACM Trans. Priv. Secur.* 25.2 (2022), 8:1–8:35.

[Bea91] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization." In: *CRYPTO*. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 420–432.

[Ber05] Daniel J. Bernstein. *Cache-timing attacks on AES*. Avaiable at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf. 2005.

[BFM22] Alexandros Bakas, Eugene Frimpong, and Antonis Michalas. "Symmetrical Disguise: Realizing Homomorphic Encryption Services from Symmetric Primitives." In: *SecureComm*. Vol. 462. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer, 2022, pp. 353–370.

[BGL20] Eli Ben-Sasson, Lior Goldberg, and David Levit. "STARK Friendly Hash - Survey and Recommendation." In: *IACR Cryptol. ePrint Arch.* (2020), p. 948.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) Fully Homomorphic Encryption without Bootstrapping." In: *ACM Trans. Comput. Theory* 6.3 (2014), 13:1–13:36.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. "Constructing Elliptic Curves with Prescribed Embedding Degrees." In: *SCN*. Vol. 2576. Lecture Notes in Computer Science. Springer, 2002, pp. 257–267.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The Round Complexity of Secure Protocols (Extended Abstract)." In: *STOC*. ACM, 1990, pp. 503–513.

[Bon19] Xavier Bonnetain. "Collisions on Feistel-MiMC and univariate GMiMC." In: *IACR Cryptol. ePrint Arch.* (2019), p. 951.

[BOS23] Thibault Balenbois, Jean-Baptiste Orfila, and Nigel P. Smart. "Trivial Transciphering With Trivium and TFHE." In: *IACR Cryptol. ePrint Arch.* (2023), p. 980.

[BPA+23] Amit Singh Bhati, Erik Pohle, Aysajan Abidin, Elena Andreeva, and Bart Preneel. "Let's Go Eevee! A Friendly and Suitable Family of AEAD Modes for IoT-to-Cloud Secure Computation." In: *CCS*. ACM, 2023, pp. 2546–2560.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP." In: *CRYPTO*. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886.

[Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols.” In: *FOCS*. IEEE Computer Society, 2001, pp. 136–145.

[Can06] Christophe De Cannière. “Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles.” In: *ISC*. Vol. 4176. Lecture Notes in Computer Science. Springer, 2006, pp. 171–186.

[CCD+20] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. “SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search.” In: *USENIX Security Symposium*. USENIX Association, 2020, pp. 2111–2128.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. “Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression.” In: *FSE*. Vol. 9783. Lecture Notes in Computer Science. Springer, 2016, pp. 313–333.

[CCP+19] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. “ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction.” In: *CCSW@CCS*. ACM, 2019, pp. 81–92.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. “SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority.” In: *CRYPTO (2)*. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 769–798.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. “Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives.” In: *CCS*. ACM, 2017, pp. 1825–1842.

[CDG+20] Melissa Chase, David Derler, Steven Goldfeder, Daniel Kales, Jonathan Katz, Valdimir Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. *The Picnic Signature Scheme Design Document (version 3)*. 2020. URL: https://github.com/microsoft/Picnic/blob/master/spec/spec-v3.0.pdf.

[CDK+19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. “Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference.” In: *CCS*. ACM, 2019, pp. 395–412.

[CDK+21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. “Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts.” In: *ACNS (1)*. Vol. 12726. Lecture Notes in Computer Science. Springer, 2021, pp. 460–479.

[CFG+18] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jian Guo, Jérémy Jean, Jean-René Reinhard, and Ling Song. "Key-Recovery Attacks on Full Kravatte." In: *IACR Trans. Symmetric Cryptol.* 2018.1 (2018), pp. 5–28.

[CFG+20] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. *PACT: Privacy Sensitive Protocols and Mechanisms for Mobile Contact Tracing.* 2020. arXiv: 2004.03544 [cs.CR].

[CGG+16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds." In: *ASIACRYPT (1)*. Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 3–33.

[CGG+20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "TFHE: Fast Fully Homomorphic Encryption Over the Torus." In: *J. Cryptol.* 33.1 (2020), pp. 34–91.

[CGG+21] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. "Fluid MPC: Secure Multiparty Computation with Dynamic Participants." In: *CRYPTO (2)*. Vol. 12826. Lecture Notes in Computer Science. Springer, 2021, pp. 94–123.

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. "Fast Large-Scale Honest-Majority MPC for Malicious Adversaries." In: *CRYPTO (3)*. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 34–64.

[CHK+21] Jihoon Cho, Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. "Transciphering Framework for Approximate Homomorphic Encryption." In: *ASIACRYPT (3)*. Vol. 13092. Lecture Notes in Computer Science. Springer, 2021, pp. 640–669.

[CHM+22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. "Towards Case-Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher." In: *ASIACRYPT (3)*. Vol. 13793. Lecture Notes in Computer Science. Springer, 2022, pp. 32–67.

[CIR22] Carlos Cid, John Petter Indrøy, and Håvard Raddum. "FASTA - A Stream Cipher for Fast FHE Evaluation." In: *CT-RSA*. Vol. 13161. Lecture Notes in Computer Science. Springer, 2022, pp. 451–483.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks." In: *CSCML*. Vol. 12716. Lecture Notes in Computer Science. Springer, 2021, pp. 1–19.

[CKK+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers." In: *ASIACRYPT (1)*. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437.

[CKM21] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. "How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures." In: *IACR Cryptol. ePrint Arch.* (2021), p. 1375.

[CL02] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials." In: *CRYPTO*. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 61–76.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography." In: *EUROCRYPT (1)*. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 769–793.

[CTV20] Ran Canetti, Ari Trachtenberg, and Mayank Varia. *Anonymous Collocation Discovery: Harnessing Privacy to Tame the Coronavirus*. 2020. arXiv: 2003.13670 [cs.CY].

[Dae95] Joan Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis*. Doctoral Dissertation. Available at https://cs.ru.nl/~joan/papers/JDA_Thesis_1995.pdf. 1995.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. "Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit." In: *CRYPTO (1)*. Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 662–692.

[DEK21] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. "Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security." In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 2183–2200.

[DEM15] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. "Higher-Order Cryptanalysis of LowMC." In: *ICISC*. Vol. 9558. Lecture Notes in Computer Science. Springer, 2015, pp. 87–101.

[DGG+21] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. "Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields." In: *EUROCRYPT (2)*. Vol. 12697. Lecture Notes in Computer Science. Springer, 2021, pp. 3–34.

[DGH12] Casey Devet, Ian Goldberg, and Nadia Heninger. "Optimally Robust Private Information Retrieval." In: *USENIX Security Symposium*. USENIX Association, 2012, pp. 269–283.

[DGH+23] Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Pasta: A Case for Hybrid Homomorphic Encryption." In: *IACR TCHES* 2023.3 (2023), pp. 30–73.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. "Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives." In: *CT-RSA*. Vol. 9048. Lecture Notes in Computer Science. Springer, 2015, pp. 127–144.

[Din21] Itai Dinur. "Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2)." In: *EUROCRYPT (1)*. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 374–403.

[DK10] Ivan Damgård and Marcel Keller. "Secure Multiparty AES." In: *Financial Cryptography*. Vol. 6052. Lecture Notes in Computer Science. Springer, 2010, pp. 367–374.

[DKR+22] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. "Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto." In: *CCS*. ACM, 2022, pp. 843–857.

[DLM+15] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. "Optimized Interpolation Attacks on LowMC." In: *ASIACRYPT (2)*. Vol. 9453. Lecture Notes in Computer Science. Springer, 2015, pp. 535–560.

[DLR16] Sébastien Duval, Virginie Lallemand, and Yann Rotella. "Cryptanalysis of the FLIP Family of Stream Ciphers." In: *CRYPTO (1)*. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 457–475.

[DM15] Léo Ducas and Daniele Micciancio. "FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second." In: *EUROCRYPT (1)*. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 617–640.

[DMA17] Joan Daemen, Bart Mennink, and Gilles Van Assche. "Full-State Keyed Duplex with Built-In Multi-user Support." In: *ASIACRYPT (2)*. Vol. 10625. Lecture Notes in Computer Science. Springer, 2017, pp. 606–637.

[DPS+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption." In: *CRYPTO*. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 643–662.

[DR00] Joan Daemen and Vincent Rijmen. "Rijndael for AES." In: *AES Candidate Conference*. National Institute of Standards and Technology, 2000, pp. 343–348.

[DR01] Joan Daemen and Vincent Rijmen. “The Wide Trail Design Strategy.” In: *IMACC*. Vol. 2260. Lecture Notes in Computer Science. Springer, 2001, pp. 222–238.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DRS18] David Derler, Sebastian Ramacher, and Daniel Slamanig. “Post-Quantum Zero-Knowledge Proofs for Accumulators with Applications to Ring Signatures from Symmetric-Key Primitives.” In: *PQCrypto*. Vol. 10786. Lecture Notes in Computer Science. Springer, 2018, pp. 419–440.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. “ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation.” In: *NDSS*. The Internet Society, 2015.

[Dwo06] Cynthia Dwork. “Differential Privacy.” In: *ICALP (2)*. Vol. 4052. Lecture Notes in Computer Science. Springer, 2006, pp. 1–12.

[EGL+20] Maria Eichlseder, Lorenzo Grassi, Reinhard Lüftenegger, Morten Øygarden, Christian Rechberger, Markus Schofnegger, and Qingju Wang. “An Algebraic Attack on Ciphers with Low-Degree Round Functions: Application to Full MiMC.” In: *ASIACRYPT (1)*. Vol. 12491. Lecture Notes in Computer Science. Springer, 2020, pp. 477–506.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. “A Randomized Protocol for Signing Contracts.” In: *CRYPTO*. Plenum Press, New York, 1982, pp. 205–210.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. “A Pragmatic Introduction to Secure Multi-Party Computation.” In: *Found. Trends Priv. Secur.* 2.2-3 (2018), pp. 70–246.

[Eur18] European Commission. *General Data Protoction Regulation*. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32016R0679. 2018.

[FLN+17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. “High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority.” In: *EUROCRYPT (2)*. Vol. 10211. Lecture Notes in Computer Science. 2017, pp. 225–255.

[FS86] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems.” In: *CRYPTO*. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 186–194.

[FV12] Junfeng Fan and Frederik Vercauteren. “Somewhat Practical Fully Homomorphic Encryption.” In: *IACR Cryptol. ePrint Arch.* (2012), p. 144.

[GA20] Google and Apple. *Apple and Google's exposure notification system*. https://www.apple.com/covid19/contacttracing. 2020.

[GAH+23] Lorenzo Grassi, Irati Manterola Ayala, Martha Norberg Hovd, Morten Øygarden, Håvard Raddum, and Qingju Wang. "Cryptanalysis of Symmetric Primitives over Rings and a Key Recovery Attack on Rubato." In: *CRYPTO (3)*. Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 305–339.

[GDL+16] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy." In: *ICML*. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 201–210.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *STOC*. ACM, 2009, pp. 169–178.

[GHR+23] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications." In: *CRYPTO (3)*. Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 573–606.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Homomorphic Evaluation of the AES Circuit." In: *CRYPTO*. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 850–867.

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation." In: *CCS*. ACM, 2022, pp. 1323–1335.

[GKL+23] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1025. Preprint.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems." In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 519–535.

[GKR+22] Lorenzo Grassi, Dmitry Khovratovich, Sondre Rønjom, and Markus Schofnegger. "The Legendre Symbol and the Modulo-2 Operator in Symmetric Schemes over Fnp Preimage Attack on Full Grendel." In: *IACR Trans. Symmetric Cryptol.* 2022.1 (2022), pp. 5–37.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. "Poseidon2: A Faster Version of the Poseidon Hash Function." In: *AFRICACRYPT*. Vol. 14064. Lecture Notes in Computer Science. Springer, 2023, pp. 177–203.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. "On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy." In: *EUROCRYPT (2)*. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 674–704.

[GM23] Matthias Geihs and Hart Montgomery. "A Low-Round Distributed PRF from Lattices and its Application to Distributed Key Management." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1254.

[GM84] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption." In: *J. Comput. Syst. Sci.* 28.2 (1984), pp. 270–299.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems." In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing But Their Validity for All Languages in NP Have Zero-Knowledge Proof Systems." In: *J. ACM* 38.3 (1991), pp. 691–729.

[GOP+22] Lorenzo Grassi, Silvia Onofri, Marco Pedicini, and Luca Sozzi. "Invertible Quadratic Non-Linear Layers for MPC-/FHE-/ZK-Friendly Schemes over Fnp Application to Poseidon." In: *IACR Trans. Symmetric Cryptol.* 2022.3 (2022), pp. 20–72.

[GØS+23] Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. "From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications." In: *EUROCRYPT (4)*. Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 255–286.

[Gra22] Lorenzo Grassi. "On Generalizations of the Lai-Massey Scheme: the Birth of Amaryllises." In: *IACR Cryptol. ePrint Arch.* (2022), p. 1245.

[Gra23] Lorenzo Grassi. "Bounded Surjective Quadratic Functions over Fnp for MPC-/ZK-/FHE-Friendly Symmetric Primitives." In: *IACR Trans. Symmetric Cryptol.* 2023.2 (2023), pp. 94–131.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments." In: *EUROCRYPT (2)*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 305–326.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. "MPC-Friendly Symmetric Key Primitives." In: *CCS*. ACM, 2016, pp. 430–443.

[GRS21] Lorenzo Grassi, Christian Rechberger, and Markus Schofnegger. "Proving Resistance Against Infinitely Long Subspace Trails: How to Choose the Linear Layer." In: *IACR Trans. Symmetric Cryptol.* 2021.2 (2021), pp. 314–352.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based." In: *CRYPTO (1)*. Vol. 8042. Lecture Notes in Computer Science. Springer, 2013, pp. 75–92.

[GW20] Ariel Gabizon and Zachary J. Williamson. "plookup: A simplified polynomial protocol for lookup tables." In: *IACR Cryptol. ePrint Arch.* (2020), p. 315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge." In: *IACR Cryptol. ePrint Arch.* (2019), p. 953.

[HKC+20] Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. "Masta: An HE-Friendly Cipher Using Modular Arithmetic." In: *IEEE Access* 8 (2020), pp. 194741–194751.

[HKL+22] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. "Rubato: Noisy Ciphers for Approximate Homomorphic Encryption." In: *EUROCRYPT (1)*. Vol. 13275. Lecture Notes in Computer Science. Springer, 2022, pp. 581–610.

[HKR+21] Lukas Helminger, Daniel Kales, Sebastian Ramacher, and Roman Walch. "Multi-party Revocation in Sovrin: Performance through Distributed Trust." In: *CT-RSA*. Vol. 12704. Lecture Notes in Computer Science. Springer, 2021, pp. 527–551.

[HL20] Phil Hebborn and Gregor Leander. "Dasta - Alternative Linear Layer for Rasta." In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 46–86.

[HMR20] Clément Hoffmann, Pierrick Méaux, and Thomas Ricosset. "Transciphering, Using FiLIP and TFHE for an Efficient Delegation of Computation." In: *INDOCRYPT*. Vol. 12578. Lecture Notes in Computer Science. Springer, 2020, pp. 39–61.

[HR22] Lukas Helminger and Christian Rechberger. "Multi-Party Computation in the GDPR." In: *Privacy Symposium 2022*. Springer International Publishing, 2022, pp. 21–39. ISBN: 978-3-031-09901-4.

[HS14] Shai Halevi and Victor Shoup. "Algorithms in HElib." In: *CRYPTO (1)*. Vol. 8616. Lecture Notes in Computer Science. Springer, 2014, pp. 554–571.

[HS15] Shai Halevi and Victor Shoup. "Bootstrapping for HElib." In: *EUROCRYPT (1)*. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 641–670.

[HS18] Shai Halevi and Victor Shoup. "Faster Homomorphic Linear Transformations in HElib." In: *CRYPTO (1)*. Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 93–120.

[IKO+07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Zero-knowledge from secure multiparty computation." In: *STOC*. ACM, 2007, pp. 21–30.

[ISO12] ISO. "Information technology — Security techniques — Lightweight cryptography — Part 3: Stream ciphers." In: *ISO/IEC 29192-3:2012* (2012).

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. "Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently." In: *CCS*. ACM, 2013, pp. 955–966.

[JLM22] Samuel Jaques, Michael Lodder, and Hart Montgomery. "ALLOSAUR: Accumulator with Low-Latency Oblivious Sublinear Anonymous credential Updates with Revocations." In: *IACR Cryptol. ePrint Arch.* (2022), p. 1362.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. "GAZELLE: A Low Latency Framework for Secure Neural Network Inference." In: *USENIX Security Symposium*. USENIX Association, 2018, pp. 1651–1669.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation." In: *CCS*. ACM, 2020, pp. 1575–1590.

[KHS+22] Seongkwang Kim, Jincheol Ha, Mincheol Son, ByeongHak Lee, Dukjae Moon, Joohee Lee, Sangyup Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. "AIM: Symmetric Primitive for Shorter Signatures with Stronger Security." In: *IACR Cryptol. ePrint Arch.* (2022), p. 1387.

[KL16] Dmitry Khovratovich and Jason Law. *Sovrin: digitial signatures in the blockchain area*. 2016. URL: https://sovrin.org/wp-content/uploads/AnonCred-RWC.pdf.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. "Private Set Intersection for Unequal Set Sizes with Mobile Applications." In: *PoPETs* 2017.4 (2017), pp. 177–197.

[KMO+21] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. "Refresh When You Wake Up: Proactive Threshold Wallets with Offline Devices." In: *IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 608–625.

[KOR19] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. "Revisiting User Privacy for Certificate Transparency." In: *EuroS&P*. IEEE, 2019, pp. 432–447.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer." In: *CCS*. ACM, 2016, pp. 830–842.

[KPP+21] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. "SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning." In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 2651–2668.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. "Overdrive: Making SPDZ Great Again." In: *EUROCRYPT (3)*. Vol. 10822. Lecture Notes in Computer Science. Springer, 2018, pp. 158–189.

[KR21] Nathan Keller and Asaf Rosemarin. "Mind the Middle Layer: The HADES Design Strategy Revisited." In: *EUROCRYPT (2)*. Vol. 12697. Lecture Notes in Computer Science. Springer, 2021, pp. 35–63.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. "Mobile Private Contact Discovery at Scale." In: *USENIX Security Symposium*. USENIX Association, 2019, pp. 1447–1464.

[KS08] Vladimir Kolesnikov and Thomas Schneider. "Improved Garbled Circuit: Free XOR Gates and Applications." In: *ICALP (2)*. Vol. 5126. LNCS. Springer, 2008, pp. 486–498.

[KST22] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes." In: *CRYPTO (4)*. Vol. 13510. Lecture Notes in Computer Science. Springer, 2022, pp. 359–388.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications." In: *ASIACRYPT*. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 177–194.

[Lau14] Ben Laurie. "Certificate Transparency." In: *ACM Queue* 12.8 (2014), pp. 10–19.

[LAW+23] Fukang Liu, Ravi Anand, Libo Wang, Willi Meier, and Takanori Isobe. "Coefficient Grouping: Breaking Chaghri and More." In: *EUROCRYPT (4)*. Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 287–317.

[LG21] Duc Viet Le and Arthur Gervais. "AMR: autonomous coin mixer with privacy preserving reward distribution." In: *AFT*. ACM, 2021, pp. 142–155.

[LIM21] Fukang Liu, Takanori Isobe, and Willi Meier. "Cryptanalysis of Full LowMC and LowMC-M with Algebraic Techniques." In: *CRYPTO (3)*. Vol. 12827. Lecture Notes in Computer Science. Springer, 2021, pp. 368–401.

[LLN+16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. "Zero-Knowledge Arguments for Lattice-Based Accumulators: Logarithmic-Size Ring Signatures and Group Signatures Without Trapdoors." In: *EUROCRYPT (2)*. Vol. 9666. Lecture Notes in Computer Science. Springer, 2016, pp. 1–31.

[LM90] Xuejia Lai and James L. Massey. "A Proposal for a New Block Encryption Standard." In: *EUROCRYPT*. Vol. 473. Lecture Notes in Computer Science. Springer, 1990, pp. 389–404.

[LMS+22] Fukang Liu, Willi Meier, Santanu Sarkar, and Takanori Isobe. "New Low-Memory Algebraic Attacks on LowMC in the Picnic Setting." In: *IACR Trans. Symmetric Cryptol.* 2022.3 (2022), pp. 102–122.

[LN17] Yehuda Lindell and Ariel Nof. "A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority." In: *CCS*. ACM, 2017, pp. 259–276.

[LP19] Chaoyun Li and Bart Preneel. "Improved Interpolation Attacks on Cryptographic Primitives of Low Algebraic Degree." In: *SAC*. Vol. 11959. Lecture Notes in Computer Science. Springer, 2019, pp. 171–193.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings." In: *EUROCRYPT*. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 1–23.

[LSM+21] Fukang Liu, Santanu Sarkar, Willi Meier, and Takanori Isobe. "Algebraic Attacks on Rasta and Dasta Using Low-Degree Equations." In: *ASIACRYPT (1)*. Vol. 13090. Lecture Notes in Computer Science. Springer, 2021, pp. 214–240.

[LSW+22] Fukang Liu, Santanu Sarkar, Gaoli Wang, Willi Meier, and Takanori Isobe. "Algebraic Meet-in-the-Middle Attack on LowMC." In: *ASIACRYPT (1)*. Vol. 13791. Lecture Notes in Computer Science. Springer, 2022, pp. 225–255.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption." In: *STOC*. ACM, 2012, pp. 1219–1234.

[LTW13] Sven Laur, Riivo Talviste, and Jan Willemson. "From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting." In: *ACNS*. Vol. 7954. Lecture Notes in Computer Science. Springer, 2013, pp. 84–101.

[MCJ+19] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. "Improved Filter Permutators for Efficient FHE: Better Instances and Implementations." In: *INDOCRYPT*. Vol. 11898. Lecture Notes in Computer Science. Springer, 2019, pp. 68–91.

[Mer89] Ralph C. Merkle. "A Certified Digital Signature." In: *CRYPTO*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 218–238.

[MGG+13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin." In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013, pp. 397–411.

[Mic10] Daniele Micciancio. "A first glimpse of cryptography's Holy Grail." In: *Commun. ACM* 53.3 (2010), p. 96.

[MJS+16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. "Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts." In: *EUROCRYPT (1)*. Vol. 9665. Lecture Notes in Computer Science. Springer, 2016, pp. 311–343.

[MNP+04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. "Fairplay - Secure Two-Party Computation System." In: *USENIX Security Symposium*. USENIX, 2004, pp. 287–302.

[MP12] Daniele Micciancio and Chris Peikert. "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller." In: *EUROCRYPT*. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 700–718.

[MR18] Payman Mohassel and Peter Rindal. "ABY^3: A Mixed Protocol Framework for Machine Learning." In: *CCS*. ACM, 2018, pp. 35–52.

[MRR20] Payman Mohassel, Peter Rindal, and Mike Rosulek. "Fast Database Joins and PSI for Secret Shared Data." In: *CCS*. ACM, 2020, pp. 1271–1287.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. "Fast and Secure Three-party Computation: The Garbled Circuit Approach." In: *CCS*. ACM, 2015, pp. 591–602.

[MTB+21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. "Multiparty Homomorphic Encryption from Ring-Learning-with-Errors." In: *PoPETs* 2021.4 (2021), pp. 291–311.

[Nat15] National Institute of Standards and Technology. "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions." In: *Federal Information Processing Standards Publication (FIPS)* (202 2015).

[NJ21] Nitin Naik and Paul Jenkins. "Sovrin Network for Decentralized Digital Identity: Analysing a Self-Sovereign Identity System Based on Distributed Ledger Technology." In: *ISSE*. IEEE, 2021, pp. 1–7.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. "Can homomorphic encryption be practical?" In: *CCSW*. ACM, 2011, pp. 113–124.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. "Privacy preserving auctions and mechanism design." In: *EC*. ACM, 1999, pp. 129–139.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES." In: *CT-RSA*. Vol. 3860. Lecture Notes in Computer Science. Springer, 2006, pp. 1–20.

[OWW+20] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. "Scaling Verifiable Computation Using Efficient Set Accumulators." In: *USENIX Security Symposium*. USENIX Association, 2020, pp. 2075–2092.

[Pag02] Dan Page. "Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel." In: *IACR Cryptol. ePrint Arch.* (2002), p. 169.

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes." In: *EUROCRYPT*. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238.

[Pol22a] Polygon. *Introducing Plonky2*. 2022. URL: https://blog.polygon.technology/introducing-plonky2/ (visited on 10/20/2022).

[Pol22b] Polygon. *Plonky2: Fast Recursive Arguments with PLONK and FRI*. 2022. URL: https://github.com/mir-protocol/plonky2/blob/main/plonky2/plonky2.pdf (visited on 04/12/2023).

[Pol23] Polygon. *Plonky3*. 2023. URL: https://github.com/Plonky3/Plonky3 (visited on 06/12/2023).

[PS20] Arpita Patra and Ajith Suresh. "BLAZE: Blazing Fast Privacy-Preserving Machine Learning." In: *NDSS*. The Internet Society, 2020.

[PSS+09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. "Secure Two-Party Computation Is Practical." In: *ASIACRYPT*. Vol. 5912. Lecture Notes in Computer Science. Springer, 2009, pp. 250–267.

[QYS+23] Wenxiao Qiao, Hailun Yan, Siwei Sun, Lei Hu, and Jiwu Jing. "New cryptanalysis of LowMC with algebraic techniques." In: *Des. Codes Cryptogr.* 91.5 (2023), pp. 2057–2075.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms." In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography." In: *STOC*. ACM, 2005, pp. 84–93.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." In: *Commun. ACM* 21.2 (1978), pp. 120–126.

[RST18] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. "Cryptanalysis of Low-Data Instances of Full LowMCv2." In: *IACR Trans. Symmetric Cryptol.* 2018.3 (2018), pp. 163–181.

[RST23] Arnab Roy, Matthias Johann Steiner, and Stefano Trevisani. "Arion: Arithmetization-Oriented Permutation and Hashing from Generalized Triangular Dynamical Systems." In: *CoRR* abs/2303.04639 (2023).

[RWG+22] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. "zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure." In: *IACR Cryptol. ePrint Arch.* (2022), p. 878.

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. "Rescue-Prime: a Standard Specification (SoK)." In: *IACR Cryptol. ePrint Arch.* (2020), p. 1143.

[Sal23] Robin Salen. "Two additional instantiations from the Tip5 hash function construction." In: *https://toposware.com/paper_tip5.pdf* (2023).

[SBV22] Samuel Steffen, Benjamin Bichsel, and Martin T. Vechev. "Zapper: Smart Contracts with Data and Identity Privacy." In: *CCS*. ACM, 2022, pp. 2735–2749.

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards." In: *J. Cryptol.* 4.3 (1991), pp. 161–174.

[SCP+22] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. "Hyperproofs: Aggregating and Maintaining Proofs in Vector Commitments." In: *USENIX Security Symposium*. USENIX Association, 2022, pp. 3001–3018.

[Sea] *Microsoft SEAL (release 4.1)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Jan. 2023.

[Sha49] Claude E. Shannon. "Communication theory of secrecy systems." In: *Bell Syst. Tech. J.* 28.4 (1949), pp. 656–715.

[Sha79] Adi Shamir. "How to Share a Secret." In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[SLS+23] Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, and Bobbin Threadbare. "The Tip5 Hash Function for Recursive STARKs." In: *IACR Cryptol. ePrint Arch.* (2023), p. 107.

[STW23] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. "Unlocking the lookup singularity with Lasso." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1216.

[SV14] Nigel P. Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations." In: *Des. Codes Cryptogr.* 71.1 (2014), pp. 57–81.

[SYZ+21] Xiaoqiang Sun, F. Richard Yu, Peng Zhang, Zhiwei Sun, Weixin Xie, and Xiang Peng. "A Survey on Zero-Knowledge Proof in Blockchain." In: *IEEE Netw.* 35.4 (2021), pp. 198–205.

[Sze21] Alan Szepieniec. "On the Use of the Legendre Symbol in Symmetric Cipher Design." In: *IACR Cryptol. ePrint Arch.* (2021), p. 984.

[Tea21] StarkWare Team. "ethSTARK Documentation." In: *IACR Cryptol. ePrint Arch.* (2021), p. 582.

[Tof80] Tommaso Toffoli. "Reversible Computing." In: *ICALP*. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 632–644.

[Tor] *Tornado Cash Privacy Solution Version 1.4*. https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf. 2021.

[Vau94] Serge Vaudenay. "On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER." In: *FSE*. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 286–297.

[Vau99] Serge Vaudenay. "On the Lai-Massey Scheme." In: *ASIACRYPT*. Vol. 1716. Lecture Notes in Computer Science. Springer, 1999, pp. 8–19.

[WTB+21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. "Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning." In: *PoPETs* 2021.1 (2021), pp. 188–208.

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)." In: *FOCS*. IEEE Computer Society, 1982, pp. 160–164.

[Yao86] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets (Extended Abstract)." In: *FOCS*. IEEE Computer Society, 1986, pp. 162–167.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. "Caulk: Lookup Arguments in Sublinear Time." In: *CCS*. ACM, 2022, pp. 3121–3134.

[Zca] *ZCash protocol specification*. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf. 2022, 1st February.

[Zca22] Zcash. *halo2*. 2022. URL: https://zcash.github.io/halo2/index.html (visited on 12/12/2022).

[ZMI89] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses." In: *CRYPTO*. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 461–480.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. "Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates." In: *EUROCRYPT (2)*. Vol. 9057. LNCS. Springer, 2015, pp. 220–250.

Part II

Publications

List of Papers

During the time of my thesis, the author contributed to 7 published and peer-reviewed papers, one published book chapter, and 2 papers with preprints available which are currently under submission. Out of these papers, 3 are published at a conference ranked A* according to the CORE ranking.[1]

Papers Used in the Thesis

▶ *Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. "Privately Connecting Mobility to Infectious Diseases via Applied Cryptography." In:* PoPETs *2022.4 (2022), pp. 768–788.*

▶ *Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Pasta: A Case for Hybrid Homomorphic Encryption." In:* IACR TCHES *2023.3 (2023), pp. 30–73.*

▶ *Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. "From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications." In:* EUROCRYPT (4). *Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 255–286.*

▶ *Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations." In:* IACR Cryptol. ePrint Arch. *(2023), p. 1025. Preprint.*

▶ *Lukas Helminger, Daniel Kales, Sebastian Ramacher, and Roman Walch. "Multi-party Revocation in Sovrin: Performance through Distributed Trust." In:* CT-RSA. *Vol. 12704. Lecture Notes in Computer Science. Springer, 2021, pp. 527–551.*

[1] http://portal.core.edu.au/conf-ranks/

Other Contributions

▶ *Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications." In:* CRYPTO (3). *Vol. 14083. Lecture Notes in Computer Science. Springer, 2023, pp. 573–606.*

▶ *Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation." In:* CCS. *ACM, 2022, pp. 1323–1335.*

▶ *Daniel Kales, Sebastian Ramacher, Christian Rechberger, Roman Walch, and Mario Werner. "Efficient FPGA Implementations of LowMC and Picnic." In:* CT-RSA. *Vol. 12006. Lecture Notes in Computer Science. Springer, 2020, pp. 417–441.*

▶ *Roman Walch, Samuel Sousa, Lukas Helminger, Stefanie N. Lindstaedt, Christian Rechberger, and Andreas Trügler. "CryptoTL: Private, efficient and secure transfer learning." In:* CoRR *abs/2205.11935 (2022). Preprint.*

▶ *Christian Rechberger and Roman Walch. "Privacy-Preserving Machine Learning Using Cryptography." In:* Security and Artificial Intelligence. *2022, pp. 109–129.*

5

Privately Connecting Mobility to Infectious Diseases via Applied Cryptography

Publication Data

Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. "Privately Connecting Mobility to Infectious Diseases via Applied Cryptography." In: *PoPETs* 2022.4 (2022), pp. 768–788

The appended paper is the author's version available at https://eprint.iacr.org/2020/522 which has been changed to use the design and formatting of this thesis.

Contributions

Main author.

Privately Connecting Mobility to Infectious Diseases via Applied Cryptography

Alexandros Bampoulidis[4], Alessandro Bruni[3], Lukas Helminger[1,2], Daniel Kales[1], Christian Rechberger[1], Roman Walch[1,2]

[1] Graz University of Technology, Graz, Austria
[2] Know-Center GmbH, Graz, Austria
[3] Katholieke Universiteit Leuven, Leuven, Belgium
[4] Research Studio Data Science, RSA FG, Vienna, Austria

Abstract

Recent work has shown that cell phone mobility data has the unique potential to create accurate models for human mobility and consequently the spread of infected diseases [WET+12]. While prior studies have exclusively relied on a mobile network operator's subscribers' aggregated data in modelling disease dynamics, it may be preferable to contemplate aggregated mobility data of infected individuals only. Clearly, naively linking mobile phone data with health records would violate privacy by either allowing to track mobility patterns of infected individuals, leak information on who is infected, or both. This work aims to develop a solution that reports the aggregated mobile phone location data of infected individuals while still maintaining compliance with privacy expectations. To achieve privacy, we use homomorphic encryption, validation techniques derived from zero-knowledge proofs, and differential privacy. Our protocol's open-source implementation can process eight million subscribers in 70 minutes.

1 Introduction

Human mobility plays a crucial role in infectious disease dynamics. It leads to more contact between receptive and infected individuals and may introduce pathogens into new geographical regions. Both cases can be responsible for an increased prevalence or an outbreak of an infectious disease [WBEM+16]. In particular, human travel history has been shown to play a critical role in the propagation of infectious diseases like influenza [FCC+05] or measles [GBK01]. Therefore, understanding the spatiotemporal dynamics of an epidemic is closely tied to understanding the movement patterns of infected individuals.

Until a few years ago, researchers had to rely on general data, such as relative distance and population distribution, to model human mobility. This model was then combined with a transmission model of a particular disease into an epidemiological model, which was then used to improve the understanding of the geographical spread of epidemics. Mobile phones and their location data have the unique potential to improve these epidemiological models further. Indeed, recent work [WET+12] has shown that substituting this inaccurate mobility data with

mobile phone data leads to significantly more accurate models. Integrating such up-to-date mobility patterns allowed them to identify hotspots with a higher risk of contamination, enabling policymakers to apply focused measures.

While prior studies have exclusively relied on aggregated data of all mobile network operator's subscribers' it may be preferable to contemplate aggregated mobility data of infected individuals only. Indeed, a cholera study [FGM+16] observed that although their model succeeded in showing that some mass gatherings had major influences in the course of the epidemic, it performed less well when the cumulative incidence is low. They speculated that demographic stochasticity could be one reason for the bad performance of their model. In other words, the infected individuals' mobility pattern may not be precisely reflected by the population's mobility if the prevalence is low. To mitigate this problem, we propose the usage of infected individuals' mobile phone data, which should lead to an improvement in the predictive capabilities of epidemiological models, especially in highly dynamic situations.

1.1 Privacy Goals

Our system should report a heatmap of aggregated mobile phone location data of infected individuals without revealing an individual's location or whether an individual has been infected. To that end we combine various state-of-the-art privacy-preserving cryptographic primitives to design a two-party client-server protocol for which the epidemiological researcher or a health authority inputs patients' identifiers, and the mobile network operator (MNO) inputs its subscribers' location data.

Our solution should, thereby, be able to combine both datasets without leaking the inputs to the other party. Furthermore, no party should be able to gain any information on the other dataset by cheating during protocol evaluation, e.g., by providing malicious inputs. Even if both parties follow the protocol honestly, the resulting heatmap of aggregated location data can still leak sensitive information about individuals. Thus, our protocol must also prevent this inherent output leakage while still preserving the usefulness of the resulting heatmap.

1.2 Roadmap

In Section 2, we discuss the relevant related work. Section 3 provides the necessary preliminary definitions and notations. Section 4 first states the problem we want to solve in this article. It then gradually develops a solution by introducing privacy protection mechanisms step by step. In Section 5, we perform a dedicated security and privacy analysis of our solution. Section 6 elaborates on the implementation of our solution as well as demonstrating the performance. Section 7 concludes with a discussion about legal considerations for an actual roll-out and how multiple parties can be included. We defer to the appendix missing proofs of our security analysis (Appendix A) and additional material regarding differential privacy (Appendix C).

2 Related Work

Numerous research directions have previously sought to model the spread of infectious diseases. Most closely related to this paper is work connecting mobile phone data to infectious diseases.

2.1 Mobility and Infectious Diseases

Mobile phone data provides an opportunity to model human travel patterns and thereby enhance understanding of infectious diseases' transmission [WBEM+16]. Location data derived from call detail records (CDRs) – phone calls and text messages – have been used to understand various infectious diseases' spatial transmission better, see Table 5.1. Each of the studies got their CDRs from one MNO, which most of the time had the largest market share and coverage. The common understanding is that biases such as Multi-SIM activity and different mobile phone usage across different geographical and socio-economic groups have a limited effect on general estimates of human mobility [WEN+13].

Table 5.1: Studies connecting mobile phone data to diseases.

	Disease	Country	Year of dataset	Subscribers (millions)	Period (months)
[TQS+09]	Malaria	Tanzania	2008	0.8	3
[WET+12]	Malaria	Kenya	2008-09	14.8	12
[IMS15]	HIV	Kenya	2008-09	14.8	12
[WME+15]	Rubella	Kenya	2008-09	14.8	12
[BGL+15]	Cholera	Haiti	2010	2.9	2
[THN+14]	Malaria	Namibia	2010-11	1.5	12
[WQB+15]	Dengue	Pakistan	2013	39.8	7
[FGM+16]	Cholera	Senegal	2013	0.1	12

The most common model was to assign an individual a daily location. More concretely, each subscriber was assigned to a study area on each day based on the cell tower with the most CDRs or the last outgoing CDR. Further, the primary study area ("home") was computed for each subscriber by taking the study area where the majority of days were spent.

All of the studies emphasized that preserving individuals' privacy is mandatory. In all cases – to the best of our understanding – the involved MNO anonymized the CDRs before handing them over to the health authority. In addition, we found that the MNO aggregated the CDRs in at least two cases. However, none of the studies discussed privacy definitions or the potential risk of de-identification, which is exceptionally high for location data [Kru09]. Therefore, it is hard to assess if they achieved their goal of preserving individuals' privacy.

2.2 Exposure Notification

Many technological approaches were developed to help reduce the spread and impact of the Covid-19 pandemic [CTV20; CFG+20; DLZ+20; GA20; PR20; TSS+20; Tro+20]. Most of them focus on exposure notification, where the main challenges include privacy-friendliness, scalability, and utility. These approaches crucially rely on sizable parts of the population using smartphones, enabling Bluetooth, and installing a new app. In contrast, our proposal does not help with contact tracing, but gives potentially useful epidemiological information to health authorities without requiring people to carry around smartphones or installing an app. Indeed, any mobile phone is sufficient.

In subsequent work [HJM+20], the authors propose to use a threshold PIR-SUM protocol to allow performing privacy-preserving epidemiological modeling on top of existing contact-tracing information. Their PIR-SUM protocol is based on a multi-server private information retrieval protocol, which is not suitable for our use case where a single entity (e.g., the mobile network operator) holds all location data. While the threshold PIR-SUM protocol can in theory be built using a single-server PIR, these protocols are significantly more expensive than the multi-server PIR they use. Furthermore, their protocols require a mix-net [APY20] to provide unlinkability of their participants messages, which already requires multiple servers, and it is not immediatly obvious how to apply their ad-hoc MPC protocol to verify the validity of queries to a single-server PIR. For single-server PIR protocols based on homomorphic encryption, our input validation procedure from Section 4.3 might be an alternative.

2.3 PSI-CA and PSI-SUM Protocols

Several works attempt to improve contact tracing by enabling users to query a database, while learning nothing more then the number of intersections using PSI-CA (Private Set Intersection Cardinality) [TSS+20; Dit+20; DPT20] protocols, or while learning nothing except the sum of the associated values of the items in the intersection using PSI-SUM [Ion+20; MPR+20] protocols.

While a PSI-SUM protocol perfectly matches our use case in theory, an application of these PSI-SUM protocols in a straightforward fashion for our main scenario in Section 6.6 – where we want to calculate the sum of vectors of length $k = 2^{15}$ for a subset of $n = 2^{23}$ identifiers – would result in impractical communication cost (multiple TB).

In [LPR+20], the authors propose a method to build PSI-SUM from their *PIR-with-default* primitive. This approach allows one to greatly reduce the communication to be linear in the smaller set size (the size of the queried subset of the population in our scenario). They present two approaches, where the first one has an expensive setup phase (multiple GB transferred for our scenario) and then has very performant queries. However, our scenario's associated values are temporal location data and would change for new protocol executions, meaning the setup phase would have to be repeated each time. Their second approach does not rely on a setup phase and – for a database size of $n = 2^{25}$ identifiers

and $t = 2^{12}$ queried elements – requires 379 MB of data transfer. However, this again only calculates the sum of a single item. A naive $k = 2^{15}$-times repetition of the approach would again result in impractical communication cost. One could investigate if the protocol can be further optimized for large associated data, since the PSI-part of the protocol does not need to be repeated.

An additional problem of the protocol in [LPR+20] is, that it cannot ensure that a query does not contain an item multiple times. Applied to our use case, this leads to problems in combination with differential privacy [Dwo06], since a larger noise needs to be added for privacy, limiting utility. The protocol in [HJM+20] solves the same issue by executing a separate MPC protocol to ensure that the queries are valid and do not contain duplicates.

2.4 Generic Multi-Party Computation

Generic Multi-Party Computation (MPC) protocols allow several parties to securely evaluate a function without having to disclose their respective inputs. Several protocols [Yao86; GMW87; DPS+12; Dam+13] and efficient implementations for generic MPC exists [Heab; Kel20], amongst many others.

We do not use generic MPC since all efficient MPC protocols exchange data linear in the size of the computed circuit and are therefore not well suited for the large databases considered in this work. Both, secret sharing and garbled circuit based MPC, would require the (secure) transmission of the server's database (either in secret-shared form or embedded in a circuit) to the client, requiring several GB of communication (e.g., $2^{23} \times 2^{15}$ matrix of 32 bit integers has a size of 1 TB). Furthermore, the most efficient secret sharing schemes, such as the popular SPDZ [Dam+13; DPS+12], require a so-called Beaver triple (for multiplying values) which has to be precomputed in an expensive offline phase and can not be reused. Computing enough triples for our protocol (i.e., one triple per database entry) would require 2^{38} triples. This triple generation alone would require > 700 hours and more then 1000 TB of communication on our hardware using the MP-SPDZ library [Kel20].

3 Preliminaries

Here we will first introduce the notations and then describe homomorphic encryption (HE) and differential privacy (DP).

We write vectors in bold lower case letters and matrices in upper case letters. We use x_i to access the i-th element of a vector $\vec{x}$. For $m \in \mathbb{N}$ and $x \in \mathbb{Z}$, let $\vec{x}^m$ be defined as the vector of powers of x: $\vec{x}^m = (1, x^1, ..., x^{m-1})$. We denote by $\vec{c} \circ \vec{d}$ the element-wise multiplication (Hadamard product) of the vectors $\vec{c}$ and $\vec{d}$. For a positive integer p, we identify $\mathbb{Z}_p = \mathbb{Z} \cap [-p/2, p/2)$. For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r.

3.1 Homomorphic Encryption

Homomorphic encryption (HE) [Gen09] allows to operate on encrypted data and, thus, has the potential to realize many 2-party protocols in a privacy-preserving manner. Compared to MPC, protocols using HE usually require less data communication and only one communication round, at the cost of more expensive computations.

Modern HE schemes [BGV12; Bra12; FV12; CKK+17; CGG+16] base their security on the learning with errors [Reg05] hardness assumption and its variant over polynomial rings [LPR10]. They allow to perform both addition and multiplication on ciphertexts. During the encryption of a plaintext, random noise is introduced into the ciphertext. This noise grows with each homomorphic operation, negligible for additions but significantly for homomorphic multiplications. Once this noise becomes too large and exceeds a threshold, the ciphertext cannot be decrypted correctly anymore. We call such a scheme, that allows evaluating an arbitrary circuit over encrypted data up to a certain depth, a somewhat homomorphic encryption scheme (SHE). The specific depth depends on the choice of encryption parameters, and choosing parameters for larger depths comes, in general, with a considerable performance penalty. In this work, we use the BFV [Bra12; FV12] SHE scheme to encrypt the inputs of our protocol.

Besides semantic security, two-party protocols based on HE additionally either require the notion of circuit privacy [Gen09], or function [GHV10] (evaluation [ACC+19]) privacy, to hide the function applied on the ciphertexts. Function privacy is often easier to achieve than circuit privacy in practice. It requires that the outputs of evaluating different circuits homomorphically on the same encrypted data need to be indistinguishable. In other words, a party decrypting the final result of a function private HE scheme can not learn anything about the circuit applied to the input data. Like many HE schemes, BFV does not naturally provide function privacy, however, it can be added by applying noise flooding [Gen09].

3.2 Differential Privacy

Differential privacy (DP) [Dwo06] defines a robust, quantitative notion of privacy for individuals. The main idea is that the outcome of a computation should be as independent as possible (usually defined by a privacy parameter ϵ) from the data of a single input. Applied to our use case, DP makes the final heatmap independent to the contribution of any individual, preventing it from leaking sensitive information.

DP is highly compatible with existing privacy frameworks and has successfully been applied to several real-world applications. Recent work [NBW+17] showed that DP satisfies privacy requirements set forth by FERPA[1]. Even before this analysis, several businesses were already using DP. For example, Apple [App20] and Google [Goo14] have applied differential privacy to gather statistics about

[1] Family Educational Rights and Privacy Act of 1974, U.S.

their users without intruding on individual's privacy. Recently, the U.S. Census 2020 uses differential privacy as a privacy protection system [Bur20].

The most prevalent technique to achieve DP is to add noise sampled from a zero-centered Laplace distribution to the outcome of the computation. The distribution is calibrated with a privacy parameter ϵ and the global sensitivity Δq of the computation and has the following probability density function:

$$Lap(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}, \qquad \text{with } b = \frac{\Delta q}{\epsilon}$$

To add noise to a protocol operating on integers, we discretize the Laplace distribution by rounding the sampled value to the nearest integer. For a formal definition of DP, we refer to Appendix C.

4 Privacy Preserving Heatmap Protocol

We first describe our goal and then introduce each privacy protection mechanism step by step.

4.1 The Desired Functionality

Our aim is to accumulate the location data of infected individuals to create a heatmap, assisting governments in managing an epidemic. Two parties controlling two different datasets are involved: A health authority who knows which individuals are infected; and an MNO who knows the location data of their subscribers. More specifically, the MNO knows how long each of their subscribers is connected to which cell towers (CDRs are a subset of this information). Based on this mobility data, our protocol could answer several questions. One in line with epidemiological literature is to look at the individuals' mobility data in the incubation period (e.g., 5-7 days for COVID-19). The final heatmap will show areas with a higher chance of getting infected with the disease. A natural extension would be to study mobility patterns after the incubation period but before confirmation/quarantine. So our protocol is generic regarding the time unit or the granularity of location data. When discussing privacy guarantees that depend on the actual data, we will explicitly outline the chosen setting.

Protocol Description.

If the MNO knows which of its subscribers is infected, it can do the following to create the desired heatmap:

- Initialize a vector $\vec{h}$ of k elements with zeros, where k is the total number of cell towers. Each element of this vector corresponds to one cell tower.
- For each infected individual, add the amount of time it spent at each cell tower to the corresponding element of the vector $\vec{h}$.

- Then the vector $\vec{h}$ contains the final heatmap, i.e., h_j contains the accumulated time spent of all infected individuals at cell tower j.

Let us rewrite this process into a single matrix multiplication. First, we encode all N subscribed individuals into a vector $\vec{x} \in \mathbb{Z}_2^N$, with $x_i \in \mathbb{Z}_2$ indicating, whether the individual i is infected ($x_i = 1$) or not ($x_i = 0$). Then we encode the location data in a matrix $Z = (\vec{z_1}, \vec{z_2}, \ldots, \vec{z_k}) \in \mathbb{Z}^{N \times k}$ such that the vector $\vec{z_j}$ contains all the location data corresponding to the cell tower identified by j. In other words, the i-th element of the vector $\vec{z_j}$ contains the amount of time individual i spent at cell tower j. Now, we can calculate the heatmap as $\vec{h} = \vec{x}^T \cdot Z$.

We depict the basic protocol, involving the health authority as a client and the MNO as a server, in Figure 5.1, assuming the health authority and the MNO already agreed on identifying all subscribed individuals by indices $i \in 1, ..., N$.

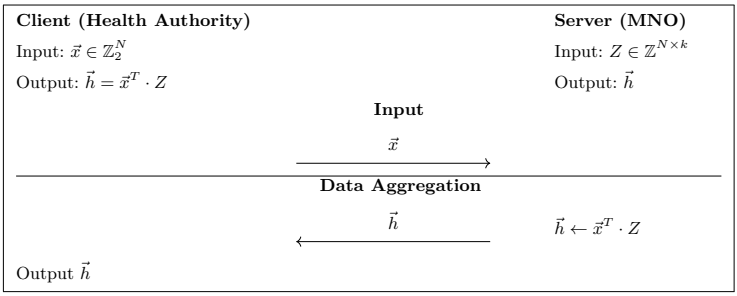


Figure 5.1: Basic protocol without privacy protection.

Remark 2 (Agreeing on database indices)**.** *The protocol in Figure 5.1 already assumes that the two parties agree on the indices of individuals in the database. In practice, the individuals can be identified using several methods, such as phone numbers, mail addresses or government ids. We now give two options to get a mapping from a phone number to a database index, while noting that any other identifier can be trivially used instead:*

- *The MNO and health authority engage in a protocol for Private Set Intersection (PSI) with associated data (e.g., [CHL+18; CMG+21]). In such a protocol, the health authority and the MNO input their list of phone numbers. The health authority gets as the protocol's output the phone numbers that are in both sets, as well as the associated data from the MNO. The associated data would be the index in the database in our case.*

- *The MNO sends a mapping of all phone numbers to their database index in plain. This approach is efficient and straightforward, but it discloses all subscribed individuals to the health authority. However, this is essentially a list of all valid phone numbers in random order and does not leak anything more than the validity of that number. Still, this may be an issue in some scenarios.*

While the PSI-based solution has some overhead compared to the plain one, the performance evaluation in [CMG+21] shows that a protocol execution with 2^{22} *MNO items and* 4096 *health authority items takes about* 1.4 *seconds online (excluding a precomputable offline phase taking* 467 *seconds) with a total communication of* 8.3 *MB – a minor increase when looking at the overall protocol. While PSI-SUM protocols [Ion+20; MPR+20; LPR+20] could be used to calculate the final heatmap without revealing which identifiers are present in the MNO's set, their additional overhead is not worth the minor privacy gain, considering that for the type of identifier we are using (phone numbers), one can often already publicly check if a phone number is associated with a mobile network operator*[1]*. We therefore relax our setting to allow revealing which identifiers are present in the MNO's set to take advantage of the reduced communication of our approach compared to the full PSI-SUM approaches mentioned above.*

Executing the protocol described in Figure 5.1 would enable the MNO to learn about infected individuals, which is a massive privacy violation. On the other hand, the health authority could query a single individual's location data by sending a vector $\vec{x} = (1, 0, \ldots, 0)$, violating privacy. In the following, we describe our techniques to protect against these violations.

4.2 Adding Encryption

To protect the vector send by the health authority, and therefore who is infected and who is not, we use a HE scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. Before executing the protocol, the health authority runs KGen to obtain a secret key sk and an evaluation key evk. We assume that the MNO knows evk, which is required to perform operations on encrypted data, before running the protocol.

In the updated protocol, the health authority now uses sk to encrypt the input vector $\vec{x}$ and sends the resulting ciphertext vector $\vec{c} \leftarrow \mathsf{Enc}_{\mathsf{sk}}(\vec{x})$ to the MNO. The MNO then uses evk to perform the matrix multiplication on the encrypted input vector and sends the resulting ciphertext vector $\vec{h}^* \leftarrow \mathsf{Eval}_{\mathsf{evk}}\left(\vec{c}^T \cdot Z\right)$ back to health authority. The health authority can now use sk to decrypt the result and get the final heatmap $\vec{h} = \mathsf{Dec}_{\mathsf{sk}}(\vec{h}^*)$.

Informally, if the used HE scheme is semantically secure, then the MNO cannot learn which individuals are infected by the disease and which are not.

4.3 Input Validation

In the simple protocol, the health authority could use a manipulated input vector $\vec{x}$ to include an individual multiple times (e.g., setting the corresponding vector entry to 100 instead of 1). Such an individual could most likely be filtered out in the final heatmap. Since the input vector is encrypted, the MNO cannot trivially check if the vector is malicious or not. Also, comparing encrypted elements is not trivially possible in most HE schemes. However, the required check can be

[1] as an example, using services such as https://dexatel.com/carrier-lookup/, or often also just calling the number

encoded, such that it outputs 0 if everything is correct, and a random value otherwise. We then can add this value to the final output as a masking value which randomizes the MNO's response if the input vector is malicious. We describe how to generate this masking below.

Masking Against Non-Binary Query Vector.

Note that the HE schemes plaintext space usually is $\mathbb{Z}_p$, i.e., the integers modulo a prime p. Therefore, the inputs to our protocol – the vector $\vec{x}$ and the matrix Z – consist of elements in $\mathbb{Z}_p$. As outlined above, it is crucial to the protocol's privacy that the input vector is binary, i.e., only contains 0s and 1s. If this is not the case, the health authority could arbitrarily modify a single person's contribution to the overall aggregated result. It is essential for DP considerations to bound the maximum possible contribution of a single individual (sensitivity).

Since the MNO only receives an encryption of the input vector, simply checking for binary values is not an option. However, we can use similar techniques to the ones used in Bulletproofs [Bün+18] to provide assurance that the query vector $\vec{x} \in \mathbb{Z}_p^N$ contains only binary elements. First, we will exploit the following general observation. Let $\vec{d} = \vec{x} - \vec{1}$, then $\vec{x} \circ \vec{d}$ is the zero vector iff $\vec{x}$ is binary. Note that the MNO can compute an encryption of $\vec{d}$ from the encrypted input vector. The result of the Hadamard product $\vec{x} \circ \vec{d}$ can be aggregated into a single value by calculating the inner product $\langle \vec{x}, \vec{d} \rangle$, which will again be zero if $\vec{x}$ is binary. The MNO also multiplies $\vec{x}$ with powers of a random integers y to reduce the probability of the health authority cheating by letting several entries of $\vec{x}$ cancel each other out during the inner product, which gives the mask:

$$\mu_{\texttt{bin}'} = \langle \vec{x}, (\vec{d} \circ \vec{y}^N) \rangle \,, \tag{5.1}$$

where $\vec{y}^N = (1, y^1, ..., y^{N-1})$ is y's vector of powers.

For the generic case of a vector $\vec{v}$ and a randomly chosen y, $\langle \vec{v}, \vec{y}^N \rangle = 0$ will hold for $\vec{v} \neq 0$ only with probability N/p [Bün+18]. Using a ν bit modulus p ($p \approx 2^\nu$), translates to a soundness error of $\nu - \log_2(N)$ bits. For details of this calculation see Section A.1. In particular, if we look at $N = 2^{23}, \nu = 60$, parameters sufficient for small nation-states (see Section 6.6), we get 37-bit statistical security. Standard literature suggest a statistical security parameter of at least 40-bit; therefore, we developed a method to enhance the statistical security without significant overhead.

Boosting Soundness.

The high-level idea is that we lower the probability of cheating successfully by a random linear combination of separate masks. Intuitively, a malicious health authority would have to guess correctly for every single mask. Thus, the soundness converges to the underlying field size in the number of terms of the linear combination. For our purpose, two terms already suffice for an appropriate

security level:

$$\mu_{\texttt{bin}} = \langle \vec{x}, (\vec{d} \circ \vec{y_1}^N) \rangle \cdot r_1 + \langle \vec{x}, (\vec{d} \circ \vec{y_2}^N) \rangle \cdot r_2$$

where $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p \setminus \{0\}$ are two random values. Therefore, the statistical security level increases to $\nu - 1$ bit ($= 59$ bits for $\nu = 60$). We refer to Lemma 2 in Section A.1 for a proof of this statement.

Applying the Mask.

Once the $\mu_{\texttt{bin}}$ is calculated, it gets added to the final output of the protocol. However, if the masking value is not zero, we have to make sure that a different random value is added to each element of the output vector to prevent leaking the mask if some output vector values are known beforehand. Therefore, the final mask $\vec{\mu}$ can be calculated using a random vector $\vec{r} \xleftarrow{\$} (\mathbb{Z}_p \setminus \{0\})^k$ as follows:

$$\vec{\mu} = \mu_{\texttt{bin}} \cdot \vec{r} \tag{5.2}$$

The final mask $\vec{\mu}$ is now equal to $\vec{0}^k$ if $\vec{x}$ is a binary vector, random otherwise.

Remark 3 (PSI-SUM with Indices)**.** *So far the protocol securely implements a functionality dubbed PSI-SUM with Indices. For completeness, we included a description and its ideal functionality in Appendix B.*

4.4 Adding Differential Privacy

The aggregated location data can still leak information about the location data of individuals. For example, the health authority could abuse the heatmap to track an individual by just querying him or by querying him alongside individuals from a completely different area. The location data of the targeted individual would be visible as an isolated zone in the resulting heatmap. Applying DP with suitable parameters will protect against such an attack since the overall goal of DP is to decrease the statistical dependence of the final result to a single database entry. In our use case, therefore, DP achieves that it is highly unlikely to distinguish between heatmaps, in which we include a single individual in the accumulation, and heatmaps, in which we do not.

Choosing proper parameters, however, highly depends on the underlying dataset. On the one hand, the chosen ϵ should be small enough to satisfy privacy concerns; on the other hand, it should be big enough not to overflow the result with noise, creating hotspots on its own. We discuss one method to choose suitable parameters in Section 5.2.

4.5 Final Protocol

Finally, with all measures to protect privacy in place, we present the final protocol in Figure 5.2.

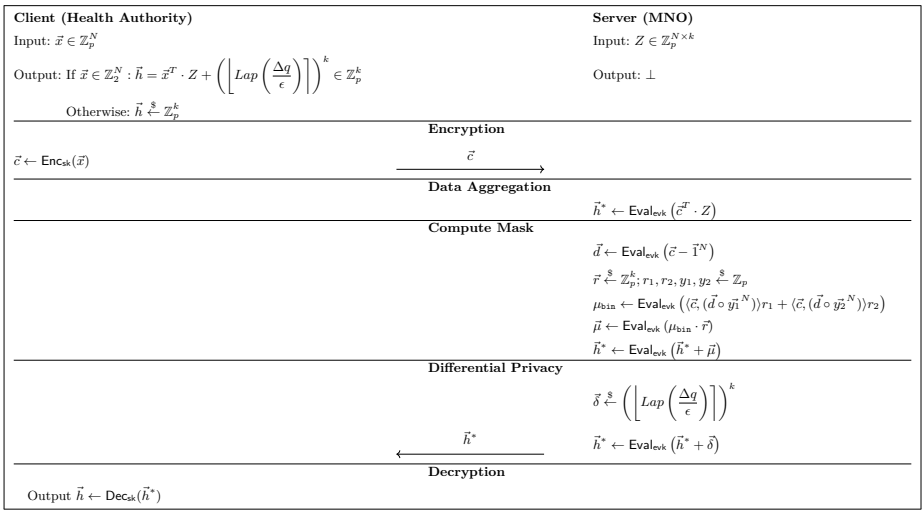


Figure 5.2: Privacy preserving heatmap protocol.

5 Security & Privacy Analysis

On the one side, the protocol provides input security against a malicious MNO, i.e., even if the MNO deviates from the protocol, it cannot determine the patient's identifiers (see Section 5.1). On the other side, individuals' location data are protected even against a malicious health authority, i.e., the health authority cannot track individuals (see Section 5.2)).

5.1 Security

Two-party protocols are usually proven secure with the real-ideal world paradigm [Can01]. Roughly speaking, one has to prove that the protocol does not leak any additional information than when computed with the help of a trusted third party. The trusted third party is modeled as an ideal functionality presented in Figure 5.3.

Semi-Honest Security.

Before we discuss malicious security, we will show that our protocol achieves semi-honest security.

Lemma 1. *Let us assume* HE *is an IND-CPA secure homomorphic encryption scheme that provides function privacy. Then the protocol in Figure 5.2 securely realizes* $\mathcal{F}_{Hmap}$ *against static semi-honest adversaries.*

The high-level idea is two-fold. Firstly, by the definition of semantic security, the MNO can not learn anything from encrypted data, hence, we reduce our protocol's security against the MNO's corruption to the semantic security of

$\mathcal{F}_{Hmap}$

Parameters: $t, N \in \mathbb{N}, \beta \in \mathbb{R}_+$. Parties P_1 and P_2.

1. Upon receiving an input $(\mathsf{input}, sid, P_1, P_2, \vec{x})$ from a party P_1, verify that $\vec{x} \in \mathbb{Z}_p^N$, else ignore input. Next, record $(sid, P_1, P_2, \vec{x})$. Once $\vec{x}$ is recorded, ignore any subsequent inputs of the form $(\mathsf{input}, sid, P_1, P_2, \cdot)$ from P_1.
2. Upon receiving an input $(\mathsf{input}, sid, P_1, P_2, Z)$ from party P_2, verify that $Z \in \mathbb{Z}_p^{N \times *}$, else ignore input. Proceed as follows: If there is a recorded value $(sid, P_1, P_2, \vec{x}, w)$, compute $\vec{h} \leftarrow \vec{x}^T Z + (\lfloor Lap(\beta) \rceil)^k$ provided that $\vec{x} \in \mathbb{Z}_2^N$, otherwise $\vec{h} \xleftarrow{\$} \mathbb{Z}_p^k$, and send (sid, P_1, P_2, k) where k is the number of columns of Z to the adversary. Then output $(\mathsf{result}, sid, P_1, P_2, \vec{h})$ to P_1, and ignore subsequent inputs of the form $(\mathsf{input}, sid, P_1, P_2, \cdot)$ from P_2.

Figure 5.3: Ideal functionality $\mathcal{F}_{Hmap}$ of the above solution.

the underlying HE scheme. Second, function privacy guarantees that the health authority learns nothing more about the MNO's matrix, than what can be derived from the input $\vec{x}$ and the output $\vec{h}$. The formal proof can be found in Appendix A.

Malicious Security.

Achieving simulation-based security against a malicious MNO would be similar to verified HE. While some theoretical constructions exist [LDP+14], they are not practical.

Instead, we show input security against a malicious MNO, also known as one-sided simulation security. This notion has been first considered in the context of oblivious transfer [NP01], was then formalized [HL08], and recently used [CHL+18] in the realm of PSI. In our protocol, one-sided simulation guarantees that the patients' identifiers are protected even in the presence of a malicious MNO (one that deviates from the protocol). For a formal definition, see Appendix A.

Theorem 1. *Let us assume* HE *is an IND-CPA secure homomorphic encryption scheme that provides function privacy. Then the protocol in Figure 5.2 securely realizes* $\mathcal{F}_{Hmap}$ *with one-sided simulation in the presence of a maliciously controlled MNO.*

Proof 1. *From Lemma 1, we already know that the protocol is secure against semi-honest adversaries. The only thing left to show is input privacy of the health authority against a malicious MNO, i.e., the MNO is not able to learn any information from the health authority's input (patients' identifier). Now, due to the fact that the MNO's view only includes an encryption of the health authority's input, by the semantic security of* HE*, we have that the MNO learns nothing about the health authority's input.* □

5.2 Privacy

The protocol's output exposes aggregate information, namely the amount of time spent by individuals at a cell tower, to the health authority. In the worst case, only one individual is present in the aggregation. Even in this case the health authority should not be able to single out any individual. To mitigate this threat, we propose to use *differential privacy* (DP).

Privacy-Utility Tradeoff.

It is a challenge to choose the right amount of noise to protect individuals' privacy while still preserving utility. Ultimately, this tradeoff is not only technical but also has to take into account normative considerations [NBW+17]. Here, we only explore the technical tradeoff and leave the latter one to policymakers.

There has been limited research specifically addressing the technical tradeoff [LC11; KL18; oth14]. However, the methods of [LC11; KL18] are not applicable to our protocol since they require either input from the individuals [KL18] or "*knowing the queries to be computed*" [LC11]. Therefore, we choose to follow [oth14]'s rigorous method to find real-world parameters for DP.

Choosing the Right ϵ.

The model in [oth14] provides a principled approach to choose ϵ. It can be split into two major steps resulting in two constraints that have to be satisfied simultaneously. First, one chooses the desired utility by defining a confidence interval. The parameters of the confidence interval give the first constraint on the required minimum number w of infected individuals and ϵ. Note that if the health authority does not provide the number of infected individuals w or lies about it, privacy is not affected. Only utility cannot be guaranteed any more.

Further, the method requires setting a bound on the expected privacy harm per individual and estimating the expected cost (baseline cost) of not being part of the outcome (e.g., database breach). This leads to the second constraint on ϵ. Every pair of parameters, ϵ, and the number of infected individuals w that simultaneously fulfill both constraints, is a reasonable privacy-utility tradeoff choice.

To illustrate this method, we now provide a possible set of values for this example. Choosing these values requires a few assumptions. We want to highlight that our assumptions are, at best, educated guesses. The real-world values have to be adjusted to the concrete circumstances and be discussed by a group of privacy, ethical, legal, epidemiological and policy experts.

First, the time unit is days for consistency with previous epidemiological studies, see Section 2.1. We decided to aim for a margin of error of $\pm 5\%$ with a probability of 0.95 (confidence). In terms of privacy, the method requires us to estimate the expected base costs (harm) that arise for an individual by using the MNO's services (data breach at the MNO would leak the location data), i.e., without even being part of the computation. We assume that without performing our protocol, this probability is less than 0.00001. In the case of a leakage, we

set the monetary harm inflicted to an individual to an exemplary amount of \$1000 per day. This seems reasonable since most smartphone users divulge exact location data for far less than that amount to companies. Now, we can calculate the expected baseline cost as $0.00001 \cdot \$1000 = \$0.01$ per day of leakage. We think performing the protocol is justified if the cost of participating does not exceed \$0.02. We arrive at the following two constraints (see Section C.1 for details)

$$\exp\left(-0.05 \cdot \frac{w\epsilon}{2}\right) \leq 0.05 \qquad \text{(utility)}$$
$$0.01 \cdot (e^{\epsilon} - 1) \leq 0.02, \qquad \text{(privacy)}$$

which are illustrated in Figure 5.4.

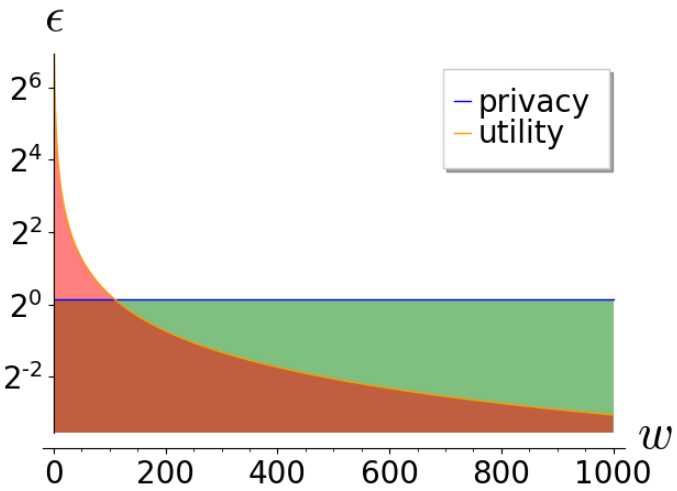


Figure 5.4: Privacy-utility tradeoff: The green area are possible combination for ϵ and w (# infected individuals). Above privacy cannot be guaranteed; below utility is not satisfied.

If the health authority wants to release a heatmap to inform the public about hotspots or justify their policies, it must add additional noise to the map. Otherwise, the MNO could subtract the noise, which itself added in the first place, thus removing the protection provided by DP. The addition of noise by both parties does not violate privacy because DP enjoys composability [DR14]. More concretely, if the heatmap produced by the MNO is ϵ_1-differentially private and the health authority adds noise corresponding to ϵ_2 to it, then the final heatmap is $(\epsilon_1 + \epsilon_2)$-differentially private. The same methodology as above should be applied to choose ϵ_2. It is crucial to find parameters such that the points (w, ϵ_1), (w, ϵ_2), and $(w, \epsilon_1 + \epsilon_2)$ are in the plot's green area.

To illustrate the trade-off of Figure 5.4 for a practical example, we performed experiments on the London subset of the publically available *gowalla*

dataset [CML11]. This dataset consists of thousands of check-in's where each check-in consists of a user-id, GPS coodinates and a timestamp. To stay consistent with the methodology we discuss in the rest of the paper, we mapped all check-in locations to the locations of the nearest cell towers in London and treat multiple check-ins from the same user to the same cell tower as just one check-in. The final dataset consists of 4571 people and 9994 cell towers. Figure 5.5 depicts a snipped of the original heatmap and the heatmaps resulting by applying DP, having $w = 600$ randomly chosen infected people and varying ϵ. The generated figures visually confirm our expectations based on the calculations above: One can observe that the heatmap without DP (Figure 5.5a) is very similar to the heatmap with too little noise (Figure 5.5d), indicating that the noise is not enough to guarantee privacy. On the other hand, the heatmap with too much noise (Figure 5.5b) clearly provides no utility due to the noise creating too many hotspots. In the correctly parameterized heatmap (Figure 5.5c), one can observe some difference to Figure 5.5a due to noise, however the biggest hotspots remain the same. In other words, privacy and utility are preserved. Government officials now can use Figure 5.5c to set new policies (e.g., closing public locations in the hotspot areas) without the possibility to track the location of individuals.

Remark 4. *Several queries could contain the same individual. Since the overall movement pattern for the same individual changes slowly over time, we model this as an identical database. Therefore the total number of queries has to be limited to the total privacy budget. For example, if we follow the values of the analysis above and the health authority queries once a week for two months (=* 8 *queries), the privacy budget suffices to provide utility as long as the number of infected individuals* w *is above* 750 *per week.*

5.3 Summary and Limitations

To summarize, the patients identifiers are encrypted during the whole protocol, hence, the semantic security of the HE scheme protects the privacy of the patients even if the MNO is cheating. The functional privacy of the HE scheme prevents, that the MNO's computation leaks anything about any location data to the health authority. The binary mask guarantees that each individual is only present at most once in the query and prevents that a cheating health authority can amplify the contribution of individual's location data in the final heatmap. Differential privacy then prevents that location data from individual's can be singled out from the resulting heatmap. Consequently, the location data of individuals is protected even if the health authority is cheating. Hence, all sensitive information is always kept private from other parties during the whole protocol.

Even though privacy of input data is guaranteed, the protocol has some practical limitations. The protocol cannot guarantee, that either the health authority, or the MNO use truthful data in the first place. In other words, malicious health authorities can randomly mark individuals as infected and MNO's can use fake location data to create the heatmap. The protocol then would guarantee privacy of these wrong inputs, but the produced heatmap would

(a) Original data without noise.

(b) $\epsilon = 0.05$: utility is not satisfied (red area in Fig. 5.4).

(c) $\epsilon = 0.6$: utility is satisfied and privacy is guaranteed (green area in Fig. 5.4).

(d) $\epsilon = 3$: privacy is not guaranteed (above the privacy line in Fig. 5.4).

Figure 5.5: Influence of different ϵ values on an artificial heatmap created by mapping the *gowalla* [CML11] dataset onto Londoner cell towers. Figure 5.5a shows the unmodified heatmap providing no privacy. While Figure 5.5d has too little noise for privacy (practically no difference compared to Figure 5.5a), Figure 5.5b has too much noise for utility. Figure 5.5c provides both privacy and utility. While the noise clearly influences the image, the hotspots remain the same.

be useless. This dependence on the truthfullness of the input data is, unfortunately, a generic problem to *any* computation (plain and privacy preserving) and can not be prevented by cryptographic measures. We, therefore, propose that independent officials perform a yearly audit, e.g., at the end of the year, of the involved data to expose cheating parties.

Another limitation of our protocol is, that the utility of the heatmap scales with the prevalence of the disease. Concretely, the more people are infected, the smaller the impact of differential privacy on the final outcome. Conversely, the less people are infected the larger the impact of the noise and the utility drops. Thus, for very small prevalences it might not be possible to achieve high utility while maintainig privacy with our protocol.

6 Implementation & Performance

The data aggregation of our protocol requires only homomorphic ciphertext-ciphertext addition and homomorphic plaintext-ciphertext multiplication; how-

ever, the evaluation of the binary mask additionally requires homomorphic ciphertext-ciphertext multiplication. For our implementation we chose to use the BFV [Bra12; FV12] SHE scheme, which fulfills these requirements. More specifically we use its implementation in the SEAL v3.6 [Heaa] library, a fast, actively developed open-source library maintained by Microsoft Research.

The computationally most expensive phase in the protocol is the Data Aggregation phase, in which the MNO multiplies a huge matrix to a homomorphically encrypted input vector. Therefore, the main objective of our implementation is to perform this huge matrix multiplication as efficiently as possible.

6.1 Packing

Modern HE schemes (including BFV) allow packing a vector of n plaintexts into only one ciphertext. Performing an operation on this ciphertext then is implicitly applied to each slot of the encrypted vector, similar to single-instruction-multiple-data (SIMD) instructions on modern CPUs (e.g., AVX). However, the ciphertext size does not depend on the exact number ($\leq n$) of encoded plaintexts. The HE schemes support various SIMD operations, including slot-wise addition, subtraction and multiplication, and slot-rotation. However, one can not directly access a specific slot of the encoded vector. We can use the SIMD encoding to speed up the matrix multiplication of our protocol significantly.

In the BFV scheme (and its implementation SEAL), the number of available SIMD slots equals the degree of the cyclotomic reduction polynomial ($x^n + 1$); thus, it is always a power of two. In the ciphertexts, the n slots are arranged as matrix of dimensions ($2 \times n/2$). A ciphertext rotation affects either all rows or all columns of the matrix simultaneously. Therefore, we can think of the inner matrix as two rotatable vectors, which can be swapped.

6.2 Homomorphic Matrix Multiplication

Since SEAL does not provide algorithms for plain-matrix times encrypted vector multiplication, we implement the baby-step giant-step (BSGS) optimized matrix-vector multiplication [HS14; HS15; HS18] on our own and optimize it to fully leverage all slots (i.e., both rotatable vectors) of the homomorphic ciphertexts.

BSGS Matrix Multiplication.

The SIMD encoding can be used to efficiently speed up matrix multiplication by using the diagonal method introduced by Halevi and Shoup [HS14], and its optimized version based on the BSGS algorithm [HS15; HS18]:

$$
\begin{aligned}
Z \cdot \vec{x} &= \sum_{i=0}^{m-1} \texttt{diag}(Z, i) \circ \texttt{rot}(\vec{x}, i) \qquad (5.3)\\
&= \sum_{k=0}^{m_2-1} \texttt{rot}\left(\sum_{j=0}^{m_1-1} \texttt{diag}'(Z, km_1 + j) \circ \texttt{rot}(\vec{x}, j), km_1 \right)
\end{aligned}
$$

where $m = m_1 \cdot m_2$ and $\texttt{diag}'(Z, i) = \texttt{rot}\left(\texttt{diag}(Z, i), -\lfloor i/m_1 \rfloor \cdot m_1\right)$.[1] Note, that $\texttt{rot}(\vec{x}, j)$ only has to be computed once for each $j < m_1$, therefore, Equation (5.3) only requires $m_1 + m_2 - 2$ rotations of the vector $\vec{x}$ in total.

Extension to Bigger Dimensions.

In our protocol, we want to homomorphically evaluate $\vec{x}^T \cdot Z = (Z^T \cdot \vec{x})^T$, where $\vec{x} \in \{0,1\}^N$ and $Z \in \mathbb{Z}_p^{N \times k}$, for big parameters N and k. As described in Section 6.1, the inner structure of the BFV ciphertext consists of two vectors of size $n/2$ each, and it does not allow a cyclic rotation over the whole input vector of size n. However, a rotation over the whole input vector is required by the BSGS algorithm. Therefore, we only can perform a BSGS multiplication with a $(n/2 \times n/2)$ matrix using this packing. Fortunately, we can use the remaining $n/2$ slots (i.e., the second vector in the inner structure of the BFV ciphertext) to simultaneously perform a second $(n/2 \times n/2)$ matrix multiplication. Therefore, after a homomorphic BSGS matrix multiplication, the result is a ciphertext c, where each of the two inner vectors encodes the result of a $(1 \times n/2) \times (n/2 \times n/2)$ vector-matrix multiplication. The sum of those two vectors can easily be obtained by rotating the columns of the ciphertext c and adding it to the first result:

$$c_{sum} = c + \texttt{rot}_{\texttt{col}}(c) \tag{5.4}$$

Thus, we can use one $(n/2 \times n/2)$ BSGS matrix multiplication and Equation (5.4) to implement a homomorphic $(1 \times n) \times (n \times n/2) = (1 \times n/2)$ vector-matrix multiplication.

Taking this into account, we split the huge $(N \times k)$ matrix into $n_v \cdot n_o$ submatrices of size $(n \times n/2)$, with $n_v = \lceil \frac{N}{n} \rceil$ and $n_o = \lceil \frac{2k}{n} \rceil$, padding the submatrices with zeros if necessary. We split the input vector $\vec{x}$ into n_v vectors of size n (padding the last vector with zeros if necessary) and encrypt each of these vectors to get n_v ciphertexts c_i. The final result of the $\vec{x}^T \cdot Z$ matrix multiplication can be computed with the following equation:

$$\tilde{c}_i = \sum_{j=0}^{n_v - 1} \texttt{MatMul}(\texttt{SubMat}(Z, j, i)^T, c_j) \;\; \forall 0 \leq i < n_o \tag{5.5}$$

where, $\texttt{SubMat}(Z, j, i)$ returns the submatrix of Z with size $(n \times n/2)$, starting at row $n \cdot j$ and column $\frac{n}{2} \cdot i$, and $\texttt{MatMul}(Z, c)$ performs the homomorphic BSGS matrix multiplication $Z \cdot c$ followed by Equation (5.4).

Equation (5.5) produces n_o ciphertexts $\tilde{c}_i$, with the final results being located in the first $n/2$ slots of the ciphertexts. Overall, our algorithm to homomorphically calculate $\vec{x}^T \cdot Z$ requires $n_v \cdot n_o$ BSGS matrix multiplications and the total multiplicative depth is 1 plaintext-ciphertext multiplication.

[1] In Equation (5.3), $\lfloor i/m_1 \rfloor$ is equal to k.

6.3 Homomorphic Evaluation of the Mask

To calculate the binary vector masking value (Equation (5.1)), we need to calculate the inner product of two homomorphically encrypted ciphertexts c and d. After an initial multiplication $c \cdot d$, the inner product requires $\log_2(n/2)$ rotations and additions, followed by Equation (5.4) to produce a ciphertext, where the result is encoded in each of the n slots. Our implementation uses rejection sampling and the SHAKE128 algorithm to cryptographically secure sample all the required random values in $\mathbb{Z}_p$. The total multiplicative depth to homomorphically evaluate the final mask (Equation (5.2)) is 1 ciphertext-ciphertext multiplication and 2 plaintext-ciphertext multiplications.

6.4 BFV parameters

In BFV, one can choose three different parameters which greatly impact the runtime, security, and the available noise budget (i.e. how much further noise can be introduced until decryption will fail). These paramaters are the degree of the reduction polynomial $n = 2^k$, the plaintext modulus p, which needs to be prime and $p \equiv 1 \pmod{2 \cdot n}$ to enable packing, and the ciphertext modulus q. We test our implementation for a computational security level of $\kappa = 128$ bit for different plaintext moduli p using the smallest n (and its default value for q) providing enough noise budget for correct evaluation of our protocol.

6.5 Function Privacy and Noise Flooding

Function privacy can be achieved by re-randomization and noise flooding, where the MNO adds an encryption of zero with a sufficiently large noise [Gen09; DS16] to the protocol's output. Following the smudging lemma [Ash+12], one needs to add a ciphertext with noise being $\lambda_{\text{FP}} + \log_2(n) + \log_2(n_o)$ bits larger than the upper bound of our protocol's original output's noise to achieve a statistical distance of $2^{\lambda_{\text{FP}}}$ between different executions.

We implement noise flooding by creating an encryption of zero (c_0) with large noise (in practice, we set the noise as large as possible while ensuring decryption is still possible). Adding c_0 to the output of our protocol (c) results in a ciphertext which has $\lambda_{\text{FP}} = \text{NOISEBUDGET}(c) - \text{NOISEBUDGET}(c_0) - \log_2(n) - \log_2(n_o)$ statistical function privacy. In our concrete parameter sets, we ensure that $\lambda_{\text{FP}} > \nu$.

However, like most efficient instantiations of function privacy, noise flooding provides security against semi-honest adversaries only (see [DS16] and contained references), and so our implementation also only provides semi-honest security. Still, once available, our implementation can use efficient maliciously function-private FHE schemes instead and benefit from security against a malicious health authority.

6.6 Benchmarks

We benchmark our prototype implementation[1] on an c5.24xlarge AWS EC2 instance (96 vCPU @ 3.6 GHz, 192 GiB RAM) running Ubuntu Server 20.04 in the Region Frankfurt with a current price of $4.656 per hour.

In our benchmarks, we focus on evaluating the runtime of the Data Aggregation phase of our protocol. Since in our use cases N is much bigger than k, we implemented multithreading, such that the threads split the number of rows in the matrix (more specifically, the number of submatrices in the rows n_v) equally amongst all available threads. Therefore, each thread has to perform at most $\lceil \frac{n_v}{\#\text{threads}} \rceil \cdot n_o$ `MatMul` evaluations, which will be combined at the end by summing up the intermediate results.

The evaluation of the proving mask with its higher multiplicative depth requires BFV parameters providing a bigger noise budget, however, its actual evaluation does not impact the overall runtime of the protocol since we perform it in an extra thread in parallel to the data aggregation. Furthermore, adding DP, noise flooding, as well as the computations of the health authority (encryption and decryption), have negligible runtime.

The runtime of our protocol is $\mathcal{O}(n_v n_o)$, i.e., it scales linearly in the number of `MatMul` evaluations. This can be seen in Figure 5.6 in which we summarize the runtime of the homomorphic matrix multiplication for different matrix dimensions using only one thread. For real-world matrix dimensions, some added runtime has to be expected due to thread synchronization and the accumulation of the intermediate thread results.

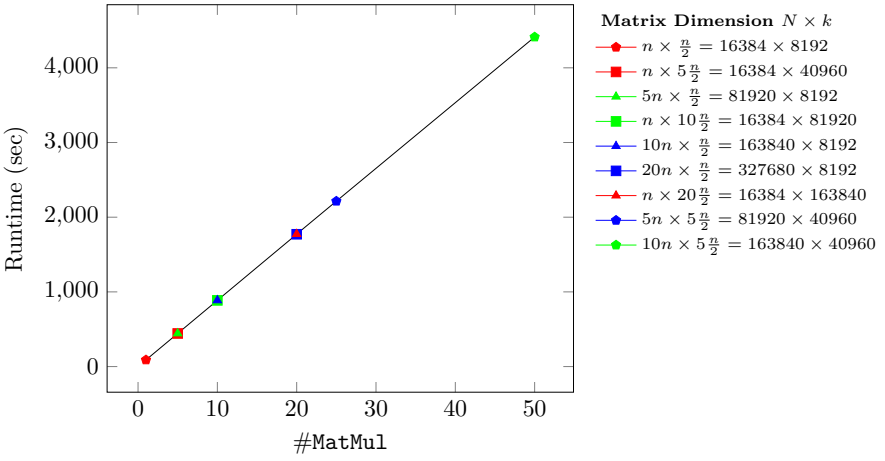


Figure 5.6: Linear dependency of the runtime of the overall matrix multiplication to the number of `MatMul` evaluations. BFV parameters are: $\log_2(p) = 42$, $n = 16384$, $\kappa = 128$.

[1] The source code is available at https://github.com/IAIK/CoronaHeatMap.

Real World Matrix Dimensions.

In our benchmarks, we want to evaluate our protocol with parameters suitable for smaller nation states and set the matrix dimensions to N being larger then the total population of small countries, and k to be larger then the total number of cell towers in these countries. Concretely, we set $N = 2^{23}$ and $k = 2^{15}$, parameters enough to evaluate our protocol, for example, for Austria [Sta21; Rad20], Singapore [Wik21; Fro17], Kenya [WET+12], New York City, Paraguay or New Zealand. In Table 5.2 we list the runtime for a homomorphic $(1 \times 2^{23}) \times (2^{23} \times 2^{15})$ matrix multiplication, for different BFV parameters, using (at most) 96 threads. We also provide the total number of `MatMul` evaluations and the (maximum) number of evaluations per thread. We give performance numbers for a plaintext prime p of size 42 bit, i.e., the smallest size to achieve $\nu = 41$ bit statistical privacy against malicious health authorities using our proving mask. To capture use cases, where a 42 bit plaintext modulus is not big enough, we also benchmark our protocol for a 60 bit prime p (the maximum value supported by SEAL), providing $\nu = 59$ bit statistical security. Further, we also give the achieved statistical function privacy λ_{FP} in bits for both benchmarks. As Table 5.2 shows, the MNO's computation takes 70 minutes for a 42 bit plaintext prime and 1 hour 25 minutes for the bigger 60 bit prime.

Table 5.2: Runtime for the MNO's computations for different parameters using 96 threads. $N = 2^{23}$, $k = 2^{15}$, $\kappa = 128$.

BFV			#`MatMul`	Time	AWS
$\log_2(p)$	n	λ_{FP}	total (thread)	min	price
42	16384	165	2048 (24)	69.33	$5.38
60	16384	96	2048 (24)	83.23	$6.46

Data Transmission.

The data sizes which have to be transmitted between the MNO and the health authority are listed in Table 5.3. Each row corresponds to a different parameter set from Table 5.2. The sizes were obtained by storing each of the described elements on the file system on the benchmarking platform. The table lists the size of the ciphertexts (ct), the public key (pk), Galois keys (gk), and relinearization keys (rk). The public key is required for noise flooding to achieve function privacy, whereas Galois keys are required to perform homomorphic rotations. Each rotation index requires one Galois key, plus an additional key for rotating the columns. When using the BSGS algorithm, we need a key for the index 1 to calculate $\texttt{rot}(\vec{x}, j)$, and a key for the indices $k \cdot m_1$, $\forall 0 < k < m_2$. Also, for masking, we need the keys for the power-of-2 indices to calculate the inner product of two ciphertexts. The relinearization key is required to linearize the result of a ciphertext-ciphertext multiplication. We want to stress that the public key (pk), Galois keys (gk), and relinearization keys (rk) only need to be sent

once before our protocol's first evaluation in a data-independent setup phase. Subsequent uses of the protocol can reuse these keys and only require transmitting the ciphertexts.

Table 5.3: Data transmission in MiB for parameters in Table 5.2.

Health Authority					MNO	Total
ct	pk[a]	gk[a]	rk[a]	Total	ct	
445.9	1.0	557.5	7.8	1012.2	1.7	1013.9
445.9	1.0	557.5	7.8	1012.2	1.7	1013.9

[a] One-time transmission (data-independent).

As Table 5.3 shows, health-authority-to-MNO communication is significantly more extensive than the response of the MNO. The main parts of the communication are the initial ciphertexts and the Galois keys. One reason for the size difference between the ciphertexts in the query and the response is that the parameter k is significantly smaller than N. Another reason is that our implementation performs a so-called modulus-switch after the computation, reducing the ciphertext modulus q to only one of the moduli q_i it is composed of. Further observe, that the plaintext modulus p does not affect the communication cost.

6.7 Price Estimation for Larger Countries

Here we give an estimate of the costs of evaluating our protocol to create a COVID-19 heatmap for a larger country, more specifically, for Germany. About 83 million people live in Germany, and a total of 80000 cell sites are deployed [Inf20]. With the BFV parameters of the first entry in Table 5.2, i.e., $n = 16384, \nu = 41, \kappa = 128$, this corresponds to $n_v \cdot n_o = 5066 \cdot 10 = 50660$ `MatMul` evaluations.

To get $n_o = 10$ `MatMul` evaluations per thread, we would have to acquire 53 CPU's capable of handling 96 threads each. Assuming a runtime of 30 min per thread (calculated from Table 5.2), and a price of \$4.656/h per CPU, we estimate the cost of evaluating the homomorphic matrix multiplication including the proving mask in a total time of 30 min to \$124 using AWS.[1] This estimate shows that it is likely very feasible to create a heatmap once a week to gain valuable insight into the spread of the disease, even for larger countries. We, however, note that care has to be taken when outsourcing this computation to cloud providers to ensure user privacy in accordance to privacy regulations.

7 Considerations and Conclusion

Our solution shows that privacy-preserving health data analytics is possible even on a national scale. We achieved this by combining three PETs. Each of

[1] In practice, additional costs for handling the databases, network traffic, key management, human resources, among some other costs are to be expected.

them has their known limitations, but filtering out their strengths and applying them purposefully lead to a real-world cryptographic protocol. More broadly, we wanted to convey the following message: Even in times of crisis where it is tempting to lower data protection standards for purposes of big data analytics, there are technical methods to keep data protection standards high. And those technical methods are practical and available.

In the following we discuss considerations when instantiating our protocol with multiple health authorities or MNO's, as well as a summary of the key takeaways from a legal case study we conducted. More concretely, we focused on the EU General Data Protection Regulation (GDPR) [Eur18], which is known to be on of the most strict privacy framework.

Multiple MNO's or Health Authorities.

Even though it has already been shown that just using the largest MNO of a country for modelling disease dynamics is highly effective in practice [WET+12], one might consider to use data from multiple MNO's. Our protocol can easily be extended to this setting by performing the protocol with each MNO individually and summing up the resulting heatmaps. As long as DP parameters (Section 5.2) are chosen, such that parameters (w_i, ϵ_i) for the i-th MNO, as well as $(\sum_i w_i, \sum_i \epsilon_i)$, fulfill the privacy-utility tradeoff, no additional information is leaked.

Multiple health authorities (e.g., for different provinces in a country) can be included using techniques from [MTB+21]. These multiple health authorities can agree on commen public keys, while keeping the decryption key hidden from all parties. After each health authority has agreed on database indices with the MNO (Remark 2), each authority can encrypt their queries using the common public key and the MNO can simply sum them up and proceed with the protocol as usual. After the protocol, the authorities proceed with the keyswitch protocol to output the final heatmap to some specified recipient (e.g., government officials). This adaptation is equivalent to the inital protocol with the same security and privacy guarantees, as long as each patient is registered with only one health authority. Otherwise, the heatmap will be a random output, due to the binary mask.

Legal Considerations.

The health authority has used HE for COVID-19 positive individuals' ids, while the MNO has used DP to protected personal data. The MNO does not enter into possession of the decryption key of the health authorities data sets. Therefore, the computations performed should be considered carried out on anonymized data [SS16], which are data that cannot identify, directly or indirectly, the data subject. In fact, data encrypted both by the health authority and the MNO is not accessible by an entity other than the one carrying out the encryption protocol. Hence, the data should be considered anonymized data – whose processing falls out of the scope of application of the GDPR (Article 29 Working Party) – for all

other entities. A similar argument holds for the aggregated location data, which are protected from singling out attacks by DP [CN20; ACN+20]. Nevertheless, the processing of data by health authorities and MNO remains bound to GDPR provisions. In particular, the process to encrypt and make such data inaccessible is a processing activity under the GDPR. Thus, it should comply with legal requirements enshrined in the GDPR. In our use case, a lawful basis for processing personal data can be found in, e.g., Art. 9 (2) (i) GDPR, which deals with data processing in a public health context. It is one reasons why it is likely that there is a legal basis for our protocol.

Therefore, both MNO and health authority's processing activity protected through state-of-the-art PETs should be considered in compliance with GDPR provisions. From a legal perspective, the added value of the provided solution is represented on the one hand by the possibility to transform personal data into anonymized data. On the other hand, the processing activity of anonymizing data and limiting access to personal data ensure data subjects respect their fundamental rights as encoded in the EU privacy and data protection framework.

Acknowledgments

We thank our shepherd Robert Cunningham for his constructive feedback and helpful insights for improving the paper. This work was supported by EU's Horizon 2020 project Safe-DEED under grant agreement n°825225, EU's Horizon 2020 project KRAKEN under grant agreement n°871473, and 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.

A Security Proofs

We now prove security using the real-ideal-paradigm [EKR18]. In this paradigm a protocol execution is secure if it behaves the same as when the parties send their input to a trusted third party (the ideal functionality) which does the computation and provides them with the outputs. More formally, an environment should not be able to distinguish between an observation of the protocol with a possible adversary and a simulator interacting with the ideal functionality. More specifically, most of the time, computational indistinguishability is required between the ideal and the real world. In contrast, we require (κ, ν)-indistinguishability to analyze the cheating probability more thoroughly.

Definition 1 ((κ,ν)-indistinguishability [LP12])**.** *Let $X = \{X(a,\kappa,\nu)\}_{\kappa,\nu,\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(a,\kappa,\nu)\}_{\kappa,\nu,\in\mathbb{N},a\in\{0,1\}^*}$ be probability ensembles, so that for any $\kappa,\nu \in \mathbb{N}$ the distribution $\{X(a,\kappa,\nu)\}$ (resp. $\{Y(a,\kappa,\nu)\}$) ranges over strings of length polynomial in $\kappa+\nu$. We say that the ensembles are (κ,ν)-indistinguishable if for every polynomial-time adversary $\mathcal{A}$, it holds that for every $a \in \{0,1\}^*$:*

$$|\Pr[\mathcal{A}(X=1)] - \Pr[\mathcal{A}(Y=1)]| < \frac{1}{p(\kappa)} + 2^{-\mathcal{O}(\nu)},$$

for every $\nu \in \mathbb{N}$, every polynomial $p(\cdot)$, and all large enough $\kappa \in \mathbb{N}$.

A.1 Binary Mask

Lemma 2. *Let p be a integer of bit-length $\nu \in \mathbb{N}$, and let $N \leq 2^{\nu/2}$. Further, let $\vec{x}$ and $\mu_{\texttt{bin}}$ be defined as in Section 4.3, then it holds that*

$$\Pr[\vec{x} \texttt{ not binary} \ \wedge \ \mu_{\texttt{bin}} = 0] =\leq \frac{1}{2^{\nu-1}}.$$

Proof 2.

$$\mu_{\texttt{bin}} = \underbrace{\langle \vec{x}, (\vec{d} \circ \vec{y_1}^N) \rangle \cdot r_1}_{:=\alpha} + \underbrace{\langle \vec{x}, (\vec{d} \circ \vec{y_2}^N) \rangle \cdot r_2}_{:=\beta} = \alpha + \beta$$

We are now interested in the events when the binary mask evaluates to zero even though $\vec{x} \notin \mathbb{Z}_2^N$. This undesired behaviour can only happen in two ways, either $\alpha = \beta = 0$ or $\alpha = -\beta$. Next, we calculate the probability of these two cases.

First, since $r_1, r_2 \neq 0$ and assuming $\vec{x} \neq \vec{0}^k$ ($\vec{x} = \vec{0}^k$ is a valid input and should result in a zero mask), we have $\Pr[\alpha = 0] = \Pr[\beta = 0] = {}^N/_p$ [Bün+18]. Hence,

$$\Pr[\alpha = \beta = 0] = \frac{N}{p} \cdot \frac{N}{p} = \frac{N^2}{p^2}. \tag{5.6}$$

Consequently, the probability of α being non-zero is $1 - {}^N/_p$. Further, the probability of β being $-\alpha$ is ${}^1/_p$. Combing these probabilities gives us

$$\Pr[\alpha = -\beta] = \left(1 - \frac{N}{p}\right)\frac{1}{p} = \frac{1}{p} - \frac{N}{p^2}. \tag{5.7}$$

We get the final probability by putting together Equation (5.6) and Equation (5.7)

$$\begin{aligned}\Pr[\alpha+\beta=0] &= \frac{N^2}{p^2}+\frac{1}{p}-\frac{N}{p^2} < \frac{1}{p}+\frac{N^2}{p^2} \\ &\leq \frac{1}{2^\nu}+\frac{2^\nu}{2^{2\nu}} = \frac{1}{2^{\nu-1}}, \text{ because } N \leq 2^{\nu/2}.\end{aligned}$$

□

A.2 Proof of Lemma 1

π_{Hmap}

1. A party P_1 on input $(\mathsf{input}, sid, P_1, P_2, \vec{x})$ from the environment verifies that $\vec{x} \in \mathbb{Z}_p^N$, else ignores the input. Next, samples a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$, and computes $\vec{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\vec{x})$. It records $(sid, P_1, P_2, \mathsf{sk})$, and sends $(sid, P_1, P_2, \mathsf{pk}, \vec{c})$ to P_2. P_1 ignores subsequent inputs of the form $(\mathsf{input}, sid, P_1, P_2, \cdot)$ from the environment.

2. On a later input of the form $(sid, P_1, P_2, \vec{h}^*)$ from P_2, P_1 computes $\vec{h} \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\vec{h}^*)$, and outputs $(\mathsf{result}, sid, P_1, P_2, \vec{h})$ to the environment.

3. A party P_2 on input $(\mathsf{input}, sid, P_1, P_2, Z)$ from the environment and $(sid, P_1, P_2, \mathsf{pk}, \vec{c})$ from P_1 verifies that $Z \in \mathbb{Z}_p^{N\times k}$, else ignores the input. Next, computes the mask vector $\vec{\mu}$ and the noise $\vec{\delta}$ according to Figure 5.2. Then computes $\vec{h}^* \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\vec{c}^T \cdot Z + \vec{\delta} + \vec{\mu})$. P_2, sends $(sid, P_1, P_2, \vec{h}^*)$ to P_1 and ignores all subsequent inputs of the form $(\mathsf{input}, sid, P_1, P_2, \cdot)$ from the environment.

Figure 5.7: Formalized protocol π_{Hmap}

Proof 3. *We use Lemma 2 to prove that to any polynomial time environment the execution π_{Hmap} with a possible adversary $\mathcal{A}$ is (κ, ν)-indistinguishable from a simulator $\mathcal{S}$ interacting with the ideal functionality $\mathcal{F}_{Hmap}$. More concretely, we claim that as long as the event that $\vec{x}$ is not binary and at the same time the mask $\vec{\mu} = \vec{0}^k$ does not occur, the executions of the ideal and real world are computational indistinguishable. Once we have proven this claim, we are done, since we have already shown that the probability of the above event is exponentially small in the statistical security parameter. Note that for the proof, we have rewritten the protocol in a more formal description π_{Hmap}, see Figure 5.7.*

First consider a polynomial time environment which does not corrupt any of the parties. Any meaningful environment will interact with π_{Hmap} or $\mathcal{F}_{Hmap}$ in the following way.

1. *It picks a vector $\vec{x} \in \mathbb{Z}_p^n$ and inputs $(\mathsf{input}, sid, P_1, P_2, \vec{x})$.*

2. *It sees $(sid, P_1, P_2, \mathsf{pk}, \vec{c})$.*

$\mathcal{S}_{Hmap}$

P_1, P_2 **not corrupted:** It starts by sampling a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\kappa)$, and sets $\vec{x} \leftarrow 0^N$. Then it computes $\vec{c} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\vec{x})$. It then instructs P_1 to send $(sid, P_1, P_2, \mathsf{pk}, \vec{c})$ to P_2. On later input of the form $(sid, P_1, P_2, \mathsf{pk}, \vec{c})$ from P_1 it samples $Z \leftarrow \mathbb{Z}_p^{N \times k}$. Then it computes $\vec{h}^* \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\vec{c}^T \cdot Z + \vec{\delta} + \vec{\mu})$. It instructs P_2 to send $(sid, P_1, P_2, \vec{h}^*)$ to P_1.

P_1 **not corrupted,** P_2 **corrupted:** Similar as before but it does not have to simulate Z because it learns the input Z from P_2. Then it computes $\mathsf{Eval}_{\mathsf{pk}}(\vec{c}^T \cdot Z + \vec{\delta} + \vec{\mu})$.

P_1 **corrupted,** P_2 **not corrupted:** It learns the input $\vec{x}$ from P_1. Then it proceeds as in the first case until it has to simulate the message to P_1. In order to do this it runs a copy of π_{Hmap} internally, where it corrupts P_1. Thereby, it learns $\vec{x}^T \cdot Z + \vec{\delta} + \vec{\mu}$ and sets $\vec{h}^* \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\vec{x}^T \cdot Z + \vec{\delta} + \vec{\mu})$.

P_1, P_2 **corrupted:** It learns the inputs $\vec{x}$ from P_1 resp. Z from P_2. It runs the protocol with the inputs, and outputs $(\mathsf{input}, sid, P_1, P_2, \vec{x})$ and $(\mathsf{input}, sid, P_1, P_2, Z)$ to the ideal functionality, which makes $\mathcal{F}_{Hmap}$ output $(\mathsf{result}, sid, P_1, P_2, \vec{x}^T \cdot Z + \vec{\delta} + \vec{\mu})$.

Figure 5.8: Simulator $\mathcal{S}_{Hmap}$.

3. *It picks a matrix* $Z \in \mathbb{Z}_p^{N \times k}$ *and inputs* $(\mathsf{input}, sid, P_1, P_2, Z)$.

4. *It sees* $(sid, P_1, P_2, \mathsf{pk}, \vec{h}^*)$.

5. *It sees* $(\mathsf{result}, sid, P_1, P_2, \vec{h})$.

Let us now assume to the contrary there is such an environment $\mathcal{E}$ *that can distinguish the two systems* $\pi_{Hmap} \circ \mathcal{A}$ *and* $\mathcal{F}_{Hmap} \circ \mathcal{S}$ *with non-negligible advantage. Then we can turn* $\mathcal{E}$ *into a polynomial time system* $\mathcal{E}'$ *which wins in the IND-CPA game with non-negligible probability:*

1. $\mathcal{E}'$ *receives* pk.

2. $\mathcal{E}'$ *runs* $\mathcal{E}$ *to see which message* $(sid, P_1, P_2, \vec{x})$ *gets recorded.*

3. $\mathcal{E}'$ *inputs* $(\vec{x}, \vec{0}^N)$ *to the IND-CPA game and gets back an encryption* $\vec{c}$*, where* $\vec{c}$ *is either an encryption of* $\vec{x}$ *(if* $b = 0$*) or an encryption of* $\vec{0}^N$ *(if* $b = 1$*).*

4. $\mathcal{E}'$ *samples* $Z \leftarrow \mathbb{Z}_p^N$*. It runs* $\mathcal{E}$ *and provides input* $(\mathsf{input}, sid, P_1, P_2, \vec{x})$*,* $(\mathsf{input}, sid, P_1, P_2, Z)$*,* $(sid, P_1, P_2, \mathsf{pk}, \vec{c})$*,* $(sid, P_1, P_2, \mathsf{Enc}_{\mathsf{pk}}(\vec{c}^T \cdot Z + \vec{\delta} + \vec{\mu}))$ *and* $(\mathsf{result}, sid, P_1, P_2, \vec{x}^T \cdot Z + \vec{\delta} + \vec{\mu})$.

5. $\mathcal{E}'$ *waits until* $\mathcal{E}$ *outputs its guess* b'*, then* $\mathcal{E}'$ *outputs* b'.

If $b = 0$, then $\mathcal{E}$ observes the interaction it would see when interacting with the protocol π_{Hmap}, and if $b = 1$, then $\mathcal{E}$ observes the interaction it would see when interacting with the ideal functionality and the simulator $\mathcal{F}_{Hmap} \circ \mathcal{S}$. By assumption $\mathcal{E}$ can distinguish $\pi_{Hmap} \circ \mathcal{A}$ and $\mathcal{F}_{Hmap} \circ \mathcal{S}$ with non-negligible advantage. Therefore, $\mathcal{E}'$ will guess b with probability significantly better than $1/2$. This is a contradiction to the IND-CPA security of HE*, as $\mathcal{E}'$ is polynomial time.* □

A.3 One-Sided Simulation

To define one-sided simulation security, we have the notion of a protocol execution view. Let $VIEW^{\mathcal{A}}_{\pi,\mathcal{A}}(x, y)$ denoted the protocol execution view of the adversary $\mathcal{A}$, i.e., the corrupted parties' view (input, randomness, all received messages) after execution of π with input x resp. y from P_1 resp. P_2.

Definition 2. *Let $EXEC_{\pi,\mathcal{A},\mathcal{E}}$ resp. $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ denote the random variables describing the output of environment $\mathcal{E}$ when interacting with an adversary $\mathcal{A}$ and parties P_1, P_2 performing protocol π, resp. when interacting with a simulator $\mathcal{S}$ and an ideal functionality $\mathcal{F}$, where only P_1 receives output. Protocol π securely realizes functionality $\mathcal{F}$ with one-sided simulation if*

1. *for any adversary $\mathcal{A}$ that controls P_2 there exists a simulator $\mathcal{S}$ such that, for any environment $\mathcal{E}$ the distribution of $EXEC_{\pi,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are indistinguishable,*
2. *and for any adversary $\mathcal{A}$ controlling P_1 the distribution $VIEW^{\mathcal{A}}_{\pi,\mathcal{A}}(x, y)$ and $VIEW^{\mathcal{A}}_{\pi,\mathcal{A}}(x, y')$, where $|y| = |y'|$ are indistinguishable.*

B PSI-SUM with Indices

In Figure 5.9 we give the ideal functionality for a *PSI-SUM with Indices* primitive. Such a primitve computes the sum of the private values associated with the intersection elements of two databases and reveals the indices present in the intersection to one party. This can be seen as an relaxed version of the *Private Intersection-Sum with Cardinality* primitive introduced in [MPR+20].

C Differential Privacy

Definition 3 (ϵ-Differential Privacy [Dwo06])**.** *A randomized mechanism $\mathcal{A}$ gives ϵ-differential privacy if for any neighboring datasets D and D', and any $S \in Range(\mathcal{A})$: $Pr[\mathcal{A}(D) = S] \leq e^{\epsilon} Pr[\mathcal{A}(D') = S]$.*

One can achieve ϵ-DP by adding noise from a zero-centered Laplace distribution to the final result of the computation. The noise is calibrated with the privacy budget ϵ and the global sensitivity Δq of the computation q: $\Delta q = \max_{D,D'} ||q(D) - q(D')||$ for all neighboring D and D'. The global sensitivity, thus,

$\mathcal{F}_{PSI-Sum}$

Parameters: $t, N \in \mathbb{N}$. Parties P_1 and P_2.

1. Upon receiving an input $(\mathsf{input}, sid, P_1, P_2, \vec{x})$ from a party P_1, verify that $\vec{x} \in \mathbb{Z}_p^N$, else ignore input. Next, record $(sid, P_1, P_2, \vec{x})$. Once $\vec{x}$ is recorded, ignore any subsequent inputs of the form $(\mathsf{input}, sid, P_1, P_2, \cdot)$ from P_1.
2. Upon receiving an input $(\mathsf{input}, sid, P_1, P_2, Z)$ from party P_2, verify that $Z \in \mathbb{Z}_p^{N \times *}$, else ignore input. Proceed as follows: If there is a recorded value $(sid, P_1, P_2, \vec{x}, w)$, compute $\vec{h} \leftarrow \vec{x}^T Z$ provided that $\vec{x} \in \mathbb{Z}_2^N$, otherwise $\vec{h} \xleftarrow{\$} \mathbb{Z}_p^k$, and send (sid, P_1, P_2, k) where k is the number of columns of Z to the adversary. Then output $(\mathsf{result}, sid, P_1, P_2, \vec{h})$ to P_1, and ignore subsequent inputs of the form $(\mathsf{input}, sid, P_1, P_2, \cdot)$ from P_2.

Figure 5.9: Ideal functionality of PSI-SUM with Indices.

represents the maximum possible value of each element in the dataset. The Laplace distribution for a scale factor b is given as $Lap(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}$, where usually $b = \frac{\Delta q}{\epsilon}$.

C.1 An economic method to choose ϵ

We aim to provide a confidence interval for the proportion μ of individuals in the general population (or subpopulation) with a specific property. Assume a database D_N and let $g : D_N \to \mathbb{R}$ be the mechanism computing the sample mean with sensitivity $1/N$. If (for privacy reasons), we add Laplace noise ν to the outcome of g, we introduce an error source. Modeling each individual as a random variable with Bernoulli distribution allows us to bound this error by the tail bound. Hence, we can define the utility by a confidence interval with accuracy $T \in [0, 1]$, and confidence $1 - \alpha$ for $\alpha \in [0, 1]$

$$\Pr[|g(D_N) + \nu(\epsilon) - \mu| \geq T] \leq e^{\left(-\frac{TN\epsilon}{2}\right)} \leq \alpha.$$

The idea of DP is that an individual's expected harm (cost) of being in the database should be minor. Let E be the expected cost for an individual for being in the database (for a formal definition see [oth14]). Then the individual's cost of being in the computation g is

$$(e^{\epsilon} - 1)E.$$

Let D_w^j be the j-th column vector of the matrix Z, i.e., the location data corresponding to cell tower j. Then, we define the mechanism as

$$g(D_w^j) := \frac{\#\text{ individuals in } j}{w},$$

resulting in sensitivity $1/w$. This setup satisfies the assumption that each individual can be modeled as a Bernoulli experiment. This can be done for every cell tower,

and thus covering the heatmap's area. The estimations of the expected baseline cost $E = \$0.01$ already cover the whole heatmap's area (all cell towers).

References

[ACC+19] Martin R Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin E Lauter, et al. "Homomorphic Encryption Standard." In: *IACR Cryptol. ePrint Arch.* (2019), p. 939.

[ACN+20] Micah Altman, Aloni Cohen, Kobbi Nissim, and Alexandra Wood. "What a Hybrid Legal-Technical Analysis Teaches Us About Privacy Regulation: The Case of Singling Out." In: *Available at SSRN* (2020).

[App20] Apple. *Differential Privacy*. https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf. 2020.

[APY20] Ittai Abraham, Benny Pinkas, and Avishay Yanai. "Blinder - Scalable, Robust Anonymous Committed Broadcast." In: *CCS*. ACM, 2020, pp. 1233–1252.

[Ash+12] Gilad Asharov et al. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE." In: *EUROCRYPT*. Vol. 7237. LNCS. Springer, 2012, pp. 483–501.

[BGL+15] Linus Bengtsson, Jean Gaudart, Xin Lu, Sandra Moore, Erik Wetter, Kankoe Sallah, Stanislas Rebaudet, and Renaud Piarroux. "Using mobile phone data to predict the spatial spread of cholera." In: *Scientific reports* 5 (2015), p. 8923.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." In: *ITCS*. ACM, 2012, pp. 309–325.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 868–886.

[Bün+18] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More." In: *IEEE S&P*. IEEE Computer Society, 2018, pp. 315–334.

[Bur20] U.S. Cencus Bureau. *Statistical Safeguards*. https://www.census.gov/about/policies/privacy/statistical_safeguards.html. 2020.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols." In: *FOCS*. IEEE, 2001, pp. 136–145.

[CFG+20] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. *PACT: Privacy Sensitive Protocols and Mechanisms for Mobile Contact Tracing*. 2020. arXiv: 2004.03544 [cs.CR].

[CGG+16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds." In: *ASIACRYPT (1)*. Vol. 10031. LNCS. 2016, pp. 3–33.

[CHL+18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. "Labeled PSI from Fully Homomorphic Encryption with Malicious Security." In: *ACM Conference on Computer and Communications Security*. ACM, 2018, pp. 1223–1237.

[CKK+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers." In: *ASIACRYPT (1)*. Vol. 10624. LNCS. Springer, 2017, pp. 409–437.

[CMG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. "Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication." In: *ACM Conference on Computer and Communications Security*. ACM, 2021, to appear.

[CML11] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. "Friendship and Mobility: User Movement in Location-Based Social Networks." In: *ACM SIGKDD*. KDD '11. New York, NY, USA: ACM, 2011, 1082–1090. ISBN: 9781450308137. DOI: 10.1145/2020408.2020579. URL: https://doi.org/10.1145/2020408.2020579.

[CN20] Aloni Cohen and Kobbi Nissim. "Towards formalizing the GDPR's notion of singling out." In: *Proc. Natl. Acad. Sci. USA* 117.15 (2020), pp. 8344–8352.

[CTV20] Ran Canetti, Ari Trachtenberg, and Mayank Varia. *Anonymous Collocation Discovery: Harnessing Privacy to Tame the Coronavirus*. 2020. arXiv: 2003.13670 [cs.CY].

[Dam+13] Ivan Damgård et al. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits." In: *ESORICS*. Vol. 8134. LNCS. Springer, 2013, pp. 1–18.

[Dit+20] Samuel Dittmer et al. "Function Secret Sharing for PSI-CA: With Applications to Private Contact Tracing." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1599.

[DLZ+20] Aaqib Bashir Dar, Auqib Hamid Lone, Saniya Zahoor, Afshan Amin Khan, and Roohie Naaz. "Applicability of mobile contact tracing in fighting pandemic (COVID-19): Issues, challenges and solutions." In: *Computer Science Review* 38 (2020), p. 100307. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100307. URL: https://www.sciencedirect.com/science/article/pii/S157401372030407X.

[DPS+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 643–662.

[DPT20] Thai Duong, Duong Hieu Phan, and Ni Trieu. "Catalic: Delegated PSI Cardinality with Applications to Contact Tracing." In: *ASIACRYPT (3)*. Vol. 12493. LNCS. Springer, 2020, pp. 870–899.

[DR14] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy." In: *Found. Trends Theor. Comput. Sci.* 9.3-4 (2014), pp. 211–407.

[DS16] Léo Ducas and Damien Stehlé. "Sanitization of FHE Ciphertexts." In: *EUROCRYPT (1)*. Vol. 9665. LNCS. Springer, 2016, pp. 294–310.

[Dwo06] Cynthia Dwork. "Differential Privacy." In: *ICALP (2)*. Vol. 4052. LNCS. Springer, 2006, pp. 1–12.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. "A Pragmatic Introduction to Secure Multi-Party Computation." In: *Found. Trends Priv. Secur.* 2.2-3 (2018), pp. 70–246.

[Eur18] European Commission. *General Data Protoction Regulation*. https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32016R0679. 2018.

[FCC+05] Neil M Ferguson, Derek AT Cummings, Simon Cauchemez, Christophe Fraser, Steven Riley, Aronrag Meeyai, Sopon Iamsirithaworn, and Donald S Burke. "Strategies for containing an emerging influenza pandemic in Southeast Asia." In: *Nature* 437.7056 (2005), pp. 209–214.

[FGM+16] Flavio Finger, Tina Genolet, Lorenzo Mari, Guillaume Constantin de Magny, Noël Magloire Manga, Andrea Rinaldo, and Enrico Bertuzzo. "Mobile phone data highlights the role of mass gatherings in the spreading of cholera outbreaks." In: *Proceedings of the National Academy of Sciences* 113.23 (2016), pp. 6421–6426.

[Fro17] Frost & Sullivan. *ASEAN Telecommunications Towers Market*. https://ww2.frost.com/wp-content/uploads/2017/01/ASEAN-Telecommunications-Towers-Market_-EDT_AG_Final.pdf. [Online; acc. 2021-03-30]. 2017.

[FV12] Junfeng Fan and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." In: *IACR Cryptology ePrint Archive* 2012 (2012), p. 144.

[GA20] Google and Apple. *Apple and Google's exposure notification system*. https://www.apple.com/covid19/contacttracing. 2020.

[GBK01] Bryan T Grenfell, Ottar N Bjørnstad, and Jens Kappey. "Travelling waves and spatial hierarchies in measles epidemics." In: *Nature* 414.6865 (2001), pp. 716–723.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *STOC*. ACM, 2009, pp. 169–178.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits." In: *CRYPTO*. Vol. 6223. LNCS. Springer, 2010, pp. 155–172.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority." In: *STOC*. ACM, 1987, pp. 218–229.

[Goo14] Google. *Learning Statistics with Privacy, aided by the Flip of a Coin*. https://ai.googleblog.com/2014/10/learning-statistics-with-privacy-aided.html. 2014.

[Heaa] *Microsoft SEAL (release 3.6)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Nov. 2020.

[Heab] *SCALE-MAMBA*. https://github.com/KULeuven-COSIC/SCALE-MAMBA. 2020.

[HJM+20] Marco Holz, Benjamin Judkewitz, Helen Möllering, Benny Pinkas, and Thomas Schneider. "PEM: Privacy-preserving Epidemiological Modeling." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1546.

[HL08] Carmit Hazay and Yehuda Lindell. "Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries." In: *TCC*. Vol. 4948. LNCS. Springer, 2008, pp. 155–175.

[HS14] Shai Halevi and Victor Shoup. "Algorithms in HElib." In: *CRYPTO (1)*. Vol. 8616. LNCS. Springer, 2014, pp. 554–571.

[HS15] Shai Halevi and Victor Shoup. "Bootstrapping for HElib." In: *EUROCRYPT (1)*. Vol. 9056. LNCS. Springer, 2015, pp. 641–670.

[HS18] Shai Halevi and Victor Shoup. "Faster Homomorphic Linear Transformations in HElib." In: *CRYPTO (1)*. Vol. 10991. LNCS. Springer, 2018, pp. 93–120.

[IMS15] Augustino Isdory, Eunice W Mureithi, and David JT Sumpter. "The impact of human mobility on HIV transmission in Kenya." In: *PloS one* 10.11 (2015), e0142805.

[Inf20] Informationszentrum Mobilfunk. *Zahl der Funkanlagenstandorte in Deutschland*. https://www.informationszentrum-mobilfunk.de/artikel/zahl-der-funkanlagenstandorte-in-deutschland. [Online; acc. 2021-03-30]. 2020.

[Ion+20] Mihaela Ion et al. “On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality.” In: *EuroS&P*. IEEE, 2020, pp. 370–389.

[Kel20] Marcel Keller. “MP-SPDZ: A Versatile Framework for Multi-Party Computation.” In: *CCS*. ACM, 2020, pp. 1575–1590.

[KL18] N. Kohli and P. Laskowski. “Epsilon Voting: Mechanism Design for Parameter Selection in Differential Privacy.” In: *2018 IEEE Symposium on Privacy-Aware Computing (PAC)*. 2018, pp. 19–30. DOI: 10.1109/PAC.2018.00009.

[Kru09] John Krumm. “A survey of computational location privacy.” In: *Pers. Ubiquitous Comput.* 13.6 (2009), pp. 391–399.

[LC11] Jaewoo Lee and Chris Clifton. “How Much Is Enough? Choosing ϵ for Differential Privacy.” In: *Information Security*. Ed. by Xuejia Lai, Jianying Zhou, and Hui Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 325–340. ISBN: 978-3-642-24861-0.

[LDP+14] Junzuo Lai, Robert H. Deng, HweeHwa Pang, and Jian Weng. “Verifiable Computation on Outsourced Encrypted Data.” In: *ESORICS (1)*. Vol. 8712. LNCS. Springer, 2014, pp. 273–291.

[LP12] Yehuda Lindell and Benny Pinkas. “Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer.” In: *J. Cryptol.* 25.4 (2012), pp. 680–722.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. “On Ideal Lattices and Learning with Errors over Rings.” In: *EUROCRYPT*. Vol. 6110. LNCS. Springer, 2010, pp. 1–23.

[LPR+20] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. “Private Join and Compute from PIR with Default.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1011.

[MPR+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. “Two-Sided Malicious Security for Private Intersection-Sum with Cardinality.” In: *CRYPTO (3)*. Vol. 12172. LNCS. Springer, 2020, pp. 3–33.

[MTB+21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. “Multiparty Homomorphic Encryption from Ring-Learning-with-Errors.” In: *Proc. Priv. Enhancing Technol.* 2021.4 (2021), pp. 291–311.

[NBW+17] Kobbi Nissim, Aaron Bembenek, Alexandra Wood, Mark Bun, Marco Gaboardi, Urs Gasser, David R O’Brien, Thomas Steinke, and Salil Vadhan. “Bridging the gap between computer science and legal approaches to privacy.” In: *Harv. JL & Tech.* 31 (2017), p. 687.

[NP01] Moni Naor and Benny Pinkas. “Efficient oblivious transfer protocols.” In: *SODA*. ACM/SIAM, 2001, pp. 448–457.

[oth14] Justin Hsu others. "Differential Privacy: An Economic Method for Choosing Epsilon." In: *CSF*. IEEE Computer Society, 2014, pp. 398–410.

[PR20] Benny Pinkas and Eyal Ronen. *Hashomer - A Proposal for a Privacy-Preserving Bluetooth Based Contact Tracing Scheme for Hamagen*. https://github.com/eyalr0/HashomerCryptoRef/blob/master/documents/hashomer.pdf. 2020.

[Rad20] Radiocells.org. *Cells and wifis in Austria*. https://www.radiocells.org/country/at. [Online; acc. 2021-03-30]. 2020.

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography." In: *STOC*. ACM, 2005, pp. 84–93.

[SS16] Gerald Spindler and Philipp Schmechel. "Personal data and encryption in the European general data protection regulation." In: *J. Intell. Prop. Info. Tech. & Elec. Com. L.* 7 (2016), p. 163.

[Sta21] Statista. *Austria: Total population from 2015 to 2025*. https://www.statista.com/statistics/263741/total-population-in-austria/. [Online; acc. 2021-03-30]. 2021.

[THN+14] Andrew J Tatem, Zhuojie Huang, Clothilde Narib, Udayan Kumar, Deepika Kandula, Deepa K Pindolia, David L Smith, Justin M Cohen, Bonita Graupe, Petrina Uusiku, et al. "Integrating rapid risk mapping and mobile phone call record data for strategic malaria elimination planning." In: *Malaria journal* 13.1 (2014), p. 52.

[TQS+09] Andrew J Tatem, Youliang Qiu, David L Smith, Oliver Sabot, Abdullah S Ali, and Bruno Moonen. "The use of mobile phone data for the estimation of the travel patterns and imported Plasmodium falciparum rates among Zanzibar residents." In: *Malaria journal* 8.1 (2009), p. 287.

[Tro+20] Carmela Troncoso et al. *Decentralized Privacy-Preserving Proximity Tracing*. 2020. arXiv: 2005.12273 [cs.CR].

[TSS+20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. *Epione: Lightweight Contact Tracing with Strong Privacy*. 2020. arXiv: 2004.13293 [cs.CR].

[WBEM+16] Amy Wesolowski, Caroline O Buckee, Kenth Engø-Monsen, and Charlotte Jessica Eland Metcalf. "Connecting mobility to infectious diseases: the promise and limits of mobile phone data." In: *The Journal of infectious diseases* 214.suppl_4 (2016), S414–S420.

[WEN+13] Amy Wesolowski, Nathan Eagle, Abdisalan M Noor, Robert W Snow, and Caroline O Buckee. "The impact of biases in mobile phone ownership on estimates of human mobility." In: *Journal of the Royal Society Interface* 10.81 (2013), p. 20120986.

[WET+12] Amy Wesolowski, Nathan Eagle, Andrew J Tatem, David L Smith, Abdisalan M Noor, Robert W Snow, and Caroline O Buckee. "Quantifying the impact of human mobility on malaria." In: *Science* 338.6104 (2012), pp. 267–270.

[Wik21] Wikipedia contributors. *Telecommunications in Singapore — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Telecommunications_in_Singapore&oldid=1002495194. [Online; acc. 2021-03-30]. 2021.

[WME+15] Amy Wesolowski, CJE Metcalf, Nathan Eagle, Janeth Kombich, Bryan T Grenfell, Ottar N Bjørnstad, Justin Lessler, Andrew J Tatem, and Caroline O Buckee. "Quantifying seasonal population fluxes driving rubella transmission dynamics using mobile phone data." In: *Proceedings of the National Academy of Sciences* 112.35 (2015), pp. 11114–11119.

[WQB+15] Amy Wesolowski, Taimur Qureshi, Maciej F Boni, Pål Roe Sundsøy, Michael A Johansson, Syed Basit Rasheed, Kenth Engø-Monsen, and Caroline O Buckee. "Impact of human mobility on the emergence of dengue epidemics in Pakistan." In: *Proceedings of the National Academy of Sciences* 112.38 (2015), pp. 11887–11892.

[Yao86] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets (Extended Abstract)." In: *FOCS*. IEEE Computer Society, 1986, pp. 162–167.

6

Pasta: A Case for Hybrid Homomorphic Encryption

Publication Data

Christoph Dobraunig, Lorenzo Grassi, Lukas Helminger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Pasta: A Case for Hybrid Homomorphic Encryption." In: *IACR TCHES* 2023.3 (2023), pp. 30–73

The appended paper is the author's version available at https://eprint.iacr.org/2021/731 which has been changed to use the design and formatting of this thesis.

Contributions

Main author.

Pasta: A Case for Hybrid Homomorphic Encryption

Christoph Dobraunig[4], Lorenzo Grassi[3], Lukas Helminger[1,2], Christian Rechberger[1], Markus Schofnegger[1], Roman Walch[1,2]

[1] IAIK, Graz University of Technology
[2] Know-Center GmbH, Graz, Austria
[3] Digital Security Group, Radboud University, Nijmegen
[4] Lamarr Security Research, Graz, Austria

Abstract

The idea of hybrid homomorphic encryption (HHE) is to drastically reduce bandwidth requirements when using homomorphic encryption (HE) at the cost of more expensive computations in the encrypted domain. To this end, various dedicated schemes for symmetric encryption have already been proposed. However, it is still unclear if those ideas are already practically useful, because *(1)* no cost-benefit analysis was done for use cases and *(2)* very few implementations are publicly available. We address this situation in several ways. We build an open-source benchmarking framework involving several use cases covering three popular libraries. Using this framework, we explore properties of the respective HHE proposals. It turns out that even medium-sized use cases are infeasible, especially when involving integer arithmetic. Next, we propose Pasta, a cipher thoroughly optimized for integer HHE use cases. Pasta is designed to minimize the multiplicative depth, while also leveraging the structure of two state-of-the-art integer HE schemes (BFV and BGV) to minimize the homomorphic evaluation latency. Using our new benchmarking environment, we extensively evaluate Pasta in SEAL and HElib and compare its properties to 8 existing ciphers in two use cases. Our evaluations show that Pasta outperforms its competitors for HHE both in terms of homomorphic evaluation time and noise consumption, showing its efficiency for applications in real-world HE use cases. Concretely, Pasta outperforms Agrasta by a factor of up to 82, Masta by a factor of up to 6 and Hera up to a factor of 11 when applied to the two use cases.

1 Introduction

In recent years, people have become increasingly concerned about the privacy of their data, and new regulations like the General Data Protection Regulation (GDPR)[1] forbid sharing and processing sensitive data. However, many applications, such as machine learning and statistics, require a vast amount of data to be as accurate as possible. With GDPR and similar regulations it is therefore difficult or even impossible to gather enough data to create useful and accurate

[1] https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32016R0679

models. One solution to this problem is employing privacy-preserving cryptographic protocols and primitives, such as secure multi-party computation (MPC) or homomorphic encryption (HE). Homomorphic encryption schemes allow performing computations on encrypted data without having access to the secret decryption key. Many privacy-preserving applications which employ homomorphic encryption use the following design principle: First, the data holder encrypts their dataset using a homomorphic encryption scheme and sends the ciphertexts to a server. The server then performs the computations on the ciphertexts and produces an encrypted result. Only the data holder knows the secret decryption key, so the server has to send the encrypted result to the data holder who can then decrypt it to get the final result of the computation. While this approach protects both the privacy of the input data and the secrecy of the applied computations, it comes with several drawbacks: First, applying homomorphic encryption results in a drastic performance penalty. Secondly, HE schemes suffer from ciphertext expansion. This means that the ciphertexts in HE schemes are several orders of magnitude larger than the corresponding plaintexts. This expansion negatively impacts the amount of data which has to be transferred from the data holder to the server. Especially when the client is an embedded device with limited bandwidth, memory, and computing power, this expansion can have a considerable impact on the overall performance of the application. The academic literature proposes two orthogonal solutions to this ciphertext expansion: Using symmetric ciphers in hybrid homomorphic encryption, or using LWE encryption and efficient conversion algorithms [CDK+21]. In this paper we focus on hybrid homomorphic encryption, its effect on integer HE use cases, and consequences of the chosen symmetric cipher.

1.1 Hybrid Homomorphic Encryption (HHE)

Hybrid homomorphic encryption was first mentioned in [NLV11]. The main idea behind HHE is the following: Instead of encrypting the data with HE schemes, encrypt the data with a symmetric cipher (expansion factor of 1) and send the symmetric ciphertexts to the server. The server then first homomorphically performs the symmetric decryption circuit to transform the symmetric ciphertext into a homomorphic ciphertext and then proceeds with performing the actual computations. This procedure trades bandwidth requirements with a more expensive computation on the server and requires that the data holder first sends the symmetric key encrypted under homomorphic encryption.

HE Schemes and HE–Friendly Symmetric Ciphers. Today, many HE schemes exist, such as BFV [Bra12; FV12] and BGV [BGV12] which allow for integer plaintexts in $\mathbb{Z}_q$ with $q \geq 2$, CKKS [CKK+17] which allows HE for real numbers, the original TFHE scheme [CGG+20] allowing only boolean plaintext, as well as the optimized TFHE version working over low-precision integers [CJP21]. These different schemes come with vastly different advantages and disadvantages and have diverging optimization criteria, such as minimizing

the multiplicative depth in BFV/BGV/CKKS and minimizing the total number of gates when using the gate-bootstrapping mode of TFHE.

At first, researchers tried to evaluate existing ciphers, like AES [DR00; DR02], with homomorphic encryption [GHS12; CCK+13; CLT14]. However, despite their plain efficiency, existing ciphers were not well-suited for HHE. Especially their large multiplicative depth deemed to be incompatible with modern HE schemes. As a consequence, researchers came up with symmetric cipher designs with different optimization criteria compared to, e.g., AES, mainly minimizing the multiplicative depth to be efficiently computable under HE. Many proposed HE–friendly symmetric ciphers, such as LowMC [ARS+15], Rasta [DEG+18], Agrasta [DEG+18], Dasta [HL20], Kreyvium [CCF+16], and FiLIP [MCJ+19], are defined over $\mathbb{Z}_2$, i.e., plaintexts are binary values. Consequently, they can be used to combat ciphertext expansion in the original TFHE scheme, as well as in BFV/BGV when instantiated with $q = 2$. Follow-up work then also introduced efficient ciphers for the requirements of the updated TFHE scheme (e.g., Elisabeth [CHM+22]), as well as ciphers tailored to CKKS, such as Rubato [HKL+22].

Open Problem. However, despite there being a vast number of symmetric ciphers proposed in the literature, the real ramifications of applying HHE to any use case are not yet understood so far. This is a direct consequence of a lack of benchmark comparisons of different symmetric ciphers in different HE libraries when applied to different use cases. As a result, the inefficiency of existing symmetric ciphers when applied to BFV/BGV with $q > 2$ (which is required for many use cases involving statistics or integer arithmetic in general, e.g., [JVC18; CMG+21; BBH+22]) was not yet realized so far: Once q is chosen for BGV/BFV, it cannot be changed without knowledge of the secret decryption key or without bootstrapping which is still not supported by many major HE libraries. Thus, if one wants to use one of the vast ciphers over $\mathbb{Z}_2$, one needs to instantiate BGV/BFV with $q = 2$ to be able to evaluate the boolean decryption circuit of these ciphers. This, however, results in also having to evaluate the use case in $\mathbb{Z}_2$ which requires to build binary circuits with significantly larger multiplicative depth to realize integer arithmetic. For this reason, using HHE in use cases over integers already implies a heavy performance loss compared to just implementing the use case with homomorphic encryption.

1.2 Contribution

Having said that, in this paper we tackle these problems and close the gap by implementing a benchmarking framework comparing multiple symmetric ciphers in three HE libraries and two use cases. We then also introduce the novel family of stream ciphers (dubbed Pasta) which are defined over $\mathbb{F}_p$. More specifically, our contributions are the following:

Extensive HHE Benchmarking Framework. To the best of our knowledge, we are the first to provide an extensive comparison of different symmetric ciphers in the context of hybrid homomorphic encryption spanning over several libraries. Notably, this increases the number of publicly implemented HHE schemes from only one to a total of 17, aiding public verifiability.[1] We come to the conclusion that most existing designs are not well-suited for large classes of use cases.

Designing an Efficient Cipher for HHE. Based on the conclusions of our benchmarking framework, we explore the design space for efficient ciphers for HHE over $\mathbb{F}_p$. Starting from the cost metrics in BFV/BGV and the Rasta design strategy, we compare several different proposal for efficient S-box implementations and show how to instantiate the slowest part of the cipher – the linear layer – in an efficient way by splitting the design in two parallel branches.

Pasta. Based on the analysis just described, we propose a new symmetric cipher, dubbed Pasta, optimized for integer HHE use cases. Pasta is defined to operate on plaintexts in $\mathbb{F}_p^t$, greatly increasing the performance compared to most previously proposed symmetric ciphers which are defined over $\mathbb{Z}_2$. Further, Pasta is designed to make use of the structure of two state-of-the-art integer HE cryptosystems (BFV and BGV) to minimize HHE decompression latency while still maintaining a small number of rounds and multiplicative depth. Our extensive benchmarks in our newly created framework confirm the advantage of Pasta compared to all other symmetric ciphers for HHE. Concretely, Pasta outperforms Agrasta [DEG+18], the currently fastest $\mathbb{Z}_2$ cipher for HHE, by a factor of 82 when applied to a small use case in HElib, and it outperforms Masta [HKC+20] and Hera [CHK+21], the two $\mathbb{F}_p^t$ contenders, by a factor of up to 6 and 11 respectively when applied to a larger use case in SEAL.

Follow-Up Works. Since we initially made our paper publicly available, our implementation framework has been used as a baseline for benchmarks in the followup designs proposed in [CIR22] and [CHM+22]. Furthermore, the Rubato cipher [HKL+22] directly uses the Feistel S-box (proposed in Section 6.4) as its non-linear layer.

1.3 Outline

The remaining paper is structured as follows. We first start with a small introduction to homomorphic encryption in Section 2, before we discuss related work to combat ciphertext expansion in different HE libraries in Section 3. Then we proceed by showing the effect of HHE on the server and client side when applied to a specific use case in Section 4. This section concludes with the statement, that the choice of symmetric cipher mostly effects the server side, which is why we proceed investigating the server side when using $\mathbb{Z}_2$ ciphers in Section 5. Since

[1] All our implementations are open source and available at https://github.com/IAIK/hybrid-HE-framework.git.

these ciphers are not suited for integer HE use cases, we design a new cipher in Section 6 and give the complete specification of the result, dubbed Pasta, in Section 7. We continue by analyzing the security of Pasta in Section 8 and finally benchmark it against its competitors in Section 9.

About Benchmarks. Throughout the paper, we run all benchmarks on a Linux server with an Intel Xeon E5-2699 v4 CPU (2.2 GHz, turboboost up to 3.6 GHz) and 512 GB RAM. Each individual benchmark only has access to one thread.

1.4 Notation

Let $t \geq 1$. For each vector $\vec{x} \in \mathbb{F}_p^{2t}$ we denote $\vec{x} := \vec{x}_L \| \vec{x}_R$ where $\vec{x}_L, \vec{x}_R \in \mathbb{F}_p^t$ are respectively the left and the right t words. Further, we write $\texttt{rot}_i(\vec{y})$ to indicate a rotation of the vector $\vec{y} \in \mathbb{F}_p^t$ by i steps to the left. With $\vec{y} \odot \vec{m}$ we denote the element-wise product (Hadamard product) between two vectors $\vec{y}, \vec{m} \in \mathbb{F}_p^t$.

2 Homomorphic Encryption

Homomorphic encryption has often been labeled the holy grail of cryptography, since it allows to perform any computation on encrypted data without knowledge of the secret decryption key. The concept of HE was introduced by Rivest et al. [RAD78], but the first schemes were only capable of performing one specific operation on encrypted data (e.g., multiplication with RSA [RSA78], addition with Paillier [Pai99]). The breakthrough came with Gentry's work from 2009 [Gen09], showing the first fully homomorphic encryption (FHE) scheme which in theory can perform any computation on encrypted data. Although deemed impractical, this work led the way for many improvements and follow-up publications [Bra12; FV12; BGV12; CGG+20; CKK+17].

Today's HE schemes base their security on the *learning with errors* (LWE) hardness assumption [Reg05], and its optimization over polynomial rings (Ring-LWE, or R-LWE) [LPR10]. In these schemes, random Gaussian noise is added during the encryption process. A homomorphic operation then increases this noise, negligible for homomorphic addition, but significant for homomorphic multiplication. Once the noise exceeds a specific threshold, the decryption will fail. The resulting schemes, therefore, allow the evaluation of arbitrary circuits over encrypted data up to a specific multiplicative depth which depends on the encryption parameters. Such a scheme is called a *somewhat homomorphic encryption* (SHE) scheme. In general, increasing the parameters to support a bigger circuit depth comes with a great performance penalty. In [Gen09], Gentry introduced the bootstrapping technique, a method to reset the noise in a homomorphic ciphertext. Bootstrapping allows to evaluate circuits of arbitrary depth on encrypted data and turns a (bootstrappable) SHE scheme into an FHE scheme. However, bootstrapping comes with a significant performance overhead,

which is why it is often faster to choose an SHE scheme with sufficiently large parameters.

2.1 Packing

Many modern HE schemes allow to encode a vector of n plaintexts into only one polynomial, and therefore, encrypt a vector into only one ciphertext [SV14]. Thereby, the size of the ciphertext does not depend on the exact number of slots ($\leq n$) of the vector filled during encryption. Homomorphic operations on the ciphertexts then correspond to element-wise operations on the encrypted vector. This packing is similar to single-instruction-multiple-data (SIMD) instructions on modern CPUs and can be used to massively increase the throughput and decrease the ciphertext expansion of HE applications. Operations supported by this packing include addition, subtraction, multiplication, and slot rotation. However, once encrypted, one cannot directly access individual slots of the encrypted vector. The available number of slots n depends on the parameters of the HE scheme and can range up to several thousand slots. Slot rotation is implemented by evaluating Galois automorphisms $\tau_i : a(X) \mapsto a(X^i)$ on encoded polynomials.

2.2 HE Schemes and Libraries

In this paper, we consider three HE schemes and their implementation in three libraries. We discuss the BFV [Bra12; FV12] scheme (and its implementation in SEAL [Pas]) in this section and for the sake of conciseness refer to Appendix A for a discussion of BGV [BGV12] in HElib [HS20] and TFHE [CGG+20] in the TFHE library [CGG+gu]. Furthermore, benchmarks in HElib and TFHE are later discussed in the appendix as well.

BFV [Bra12; FV12] in SEAL [Pas]. In BFV in SEAL plaintexts are elements in $\mathbb{Z}_q$. However, to support the packing described in the previous section, q has to be a prime p and packing is not supported for $q = 2$, i.e., one can not pack boolean plaintexts. We use SEAL version 3.6.2 in the paper. The runtime and added noise by homomorphic additions is negligible, which is why additions are considered free in the BFV cryptosystem. Therefore, the most relevant performance metric is the multiplicative depth of the evaluated circuit.

3 Related Work

In this paper we focus on HHE for the BFV and BGV HE schemes, and also discuss the application to the gate-bootstrapping mode of the original TFHE library. Hence, we include the boolean ciphers LowMC [ARS+15], Rasta [DEG+18], Agrasta [DEG+18] (which is the "aggressive" version of Rasta, recently broken in [LSM+21]), Dasta [HL20], Kreyvium [CCF+16], and FiLIP [MCJ+19], alongside the $\mathbb{F}_p$ competitors Masta [HKC+20] and Hera [CHK+21], in our comparison. However, other proposals for different HE schemes, such as CKKS

and the Concrete library, exist which we shortly discuss in Section 3.1 and Section 3.2. Finally, in Section 3.3, we discuss an alternative approach to reducing bandwidth requirements for HE applications which does not involve symmetric encryption schemes.

3.1 HHE for CKKS

The CKKS [CKK+17] homomorphic encryption scheme is another HE scheme which is relevant for private statistics and machine learning [VJH21; DSC+19; WSH+22; CGL+20]. However, the scheme includes approximation errors, which makes it incompatible with directly evaluating symmetric ciphers under a CKKS encryption. In [CHK+21] the authors mitigate this problem by proposing a framework (alongside the stream cipher Hera), where the symmetric cipher is first evaluated under a BFV encryption, before it gets translated to a CKKS ciphertext. The currently fastest symmetric cipher proposed for this framework is Rubato [HKL+22]. Similar to CKKS, this cipher includes approximation errors, which allows it to greatly reduces the number of rounds. Consequently, it is very fast when used with CKKS, but incompatible with BFV and BGV. To highlight the impact of Pasta we want to mention that the S-Box used in Rubato is directly taken from Pasta as proposed in Section 6.4.

3.2 HHE for Concrete

Recently, a new HE library, dubbed Concrete [CJL+20], has emerged, which implements a newer variant of TFHE as proposed in [CJP21]. This library is vastly different compared to SEAL/HElib: it allows to perform HE on plaintexts in the ring $\mathbb{Z}_{2^q}$ for small q, supports bootstrapping and evaluating lookup tables during bootstrapping. Packing, however, is not supported. In [CHM+22], the authors introduce Elisabeth-4, a $\mathbb{Z}_{2^q}$ variant of FiLIP which is optimized for HHE using Concrete, and evaluate its performance when classifying the FMNIST dataset using a deep neural network with HHE. Using Elisabeth-4 (and consequently Concrete) leads to different tradeoffs compared to Pasta: On one hand HE use cases are not bound by the depth due to bootstrapping, on the other hand, it only allows small precision integers ($q = 4$) potentially limiting its applicability to high-precision use cases. Directly comparing Elisabeth and Pasta is difficult due to their different design criteria and optimizations for vastly different HE libraries. Nonetheless, in [CHM+22] the authors compare Elisabeth to Pasta using our implementation framework, showing that a singlethreaded evaluation of Pasta-4 in HElib has a 1.26 times higher throughput than a multithreaded Elisabeth-4 in Concrete even though it is evaluated with 48 threads.

3.3 LWE-Native Encryption

In [CDK+21], the authors describe efficient algorithms to convert many LWE ciphertexts into a packed (see Section 2.1) R-LWE one. These algorithms can also be used to reduce ciphertext expansion of homomorphic encryption. Their

approach works as follows: First, they encrypt each plaintext $m_i \in \mathbb{F}_p$ under a secret key $\vec{s} \in \mathbb{Z}^N$ using basic LWE encryption by sampling a random vector $\vec{a}_i \xleftarrow{\$} \mathbb{Z}_q^N$ and calculating $b_i = -\langle \vec{a}_i, \vec{s} \rangle + \mu_i$, where $\mu_i \in \mathbb{Z}_q$ is a randomized encoding of m_i (with Gaussian noise). The LWE ciphertext then is $(b_i, \vec{a}_i) \in \mathbb{Z}_q^{N+1}$. To further reduce the size of the ciphertexts, one can use a random seed $\mathtt{se}$ and generate $\vec{a}_i$ using a pseudo-random number generator (PRNG) f. The seed can be reused to generate the random part of each ciphertext as $\vec{a}_i = f(\mathtt{se}; i)$. The resulting ciphertexts are semantically secure in the random oracle model. The client then transmits all b_i alongside the seed $\mathtt{se}$ to the server, which then transforms all LWE ciphertexts into a packed HE one using the algorithms described in [CDK+21]. The total communication cost for this approach is one $\mathbb{Z}_q$ element for each plaintext $m_i \in \mathbb{F}_p$, plus one seed $\mathtt{se}$ to generate the random part of the ciphertexts.

According to the benchmarks in [CDK+21], the LWE encryption approach has a smaller multiplicative depth, and thus, less noise consumption compared to HHE.[1] Depending on the actually evaluated use case, this smaller noise consumption can lead to requiring smaller HE parameters with less noise budget, and thus, a runtime advantage. However, their algorithms do not achieve a ciphertext expansion factor of 1, but a factor of $\frac{\log q}{\log p} + |\mathtt{se}|$. For many HE applications, the plaintext space defined by p is in the range of 16 to 60 bit and the size of q can easily exceed 800 bits, resulting in big expansion factors.

4 A first look at HHE

The performance, advantages, and disadvantages of HHE are not so well understood so far. Therefore, we start with an high-level investigation of the effects on both the client and server when applying HHE to a real use case before we investigate the choice of symmetric cipher in the next sections.

Benchmarking a Generic Use Case. Matrix multiplications over integers are a basic building block in many applications involving statistics or machine learning. Hence, for our first look we choose to apply HE and HHE to a use case involving three affine transformations to a secret vector $\vec{x}_0$. In other words, the layers have the form $\vec{x}_i' = M_i \cdot \vec{x}_i + \vec{b}_i$, where $\vec{x}_i, \vec{x}_i', \vec{b}_i \in \mathbb{F}_p^{200}$, $M_i \in \mathbb{F}_p^{200 \times 200}$, and p is a 60-bit prime. To make the use case more generic, we elementwise square the output vector after the first two affine transformations. The final use case has a multiplicative depth of 3 plaintext-ciphertext and 2 ciphertext-ciphertext multiplications and can be seen as, e.g., a small 3-layer neural network with squaring activation functions. We benchmark this use case after the initial setup phase, i.e., the server knows an HE encryption of the symmetric key and all HE evaluation keys. Further, we repeat this 1000 times, and the server aggregates

[1] The source code is not public at the time of writing and their benchmarks are missing the final transformation to get a packed HE ciphertext. However, our own implementation verifies the statements in this paragraph (see Section 4).

the final results before sending them back to the client. In a real-world scenario, this would be equivalent to, e.g., a sensor device sending measurements in fixed intervals to a server.

In Table 6.1, we give results for evaluating this use case in the SEAL library, first by just using HE, then by applying HHE with 3 different ciphers, and finally by applying the alternative approach using LWE-native encryption [CDK+21] (i.e., transmitted ciphertext are essentially seeded LWE ciphertexts). To better show the effects of HHE, we instantiate the HHE benchmark once with a generic symmetric cipher (AES), once with a fast boolean HHE optimized cipher (Agrasta), and once with a HHE optimized cipher defined over $\mathbb{F}_p$ (Pasta-3 as defined in Section 7 – since we aim to investigate the general effects of applying HHE to a use case in this section, details on Pasta-3 are not important at this point).

Table 6.1: Comparison of a use case with HHE to only using HE in SEAL.

	Client				Server		
	Random Words	Enc. *s*	RAM GB	Comm. kB	Runtime *s*	RAM GB	Response kB
HE	65 536 000	59	0.550	7 404 700	61 900	2.24	987.3
HHE **(Pasta-3)**	**0**	**16**	**0.005**	**1 500**	**669 400**	**23.0**	**2 097.3**
HHE (Agrasta)	0	2 200	0.004	1 500	?[a]	?[a]	$> 12\,000\,000$[b]
HHE (AES)	0	0.040	0.003	1 500	?[a]	?[a]	$> 12\,000\,000$[b]
LWE [CDK+21]	200 000	4 229	0.361	22 025	165 900	28.1	987.3

[a] Multiplicative depth of binary circuit (> 400) far too large for feasible HE parameters.
[b] No packing in SEAL for $\mathbb{Z}_2$ plaintexts, i.e., one HE ciphertext per bit.

HHE Results. As Table 6.1 shows, using HHE reduces the total client-to-server communication from 7.4 GB to 1.5 MB, the exact size of sending the input vector consisting of 200 60-bit field elements 1000 times. Furthermore, data encryption is also faster and requires less RAM, with the traditional cipher AES being the fastest option. However, to support the homomorphic evaluation of the HHE decompression circuit (i.e., homomorphically computing the symmetric decryption), the server-side requires larger HE parameters with higher noise budget, increasing the server-side runtime and RAM requirements. For HHE using the $\mathbb{F}_p$ cipher (i.e., Pasta-3), the server-side runtime increases by a factor of 10. However, using HHE with $\mathbb{Z}_2$ ciphers (e.g., Agrasta or AES) requires to implement binary circuits for the use case, resulting in a significant multiplicative depth requiring huge HE parameters, and thus in infeasibly long server runtimes.

Remark 5. *As discussed in Section 2, one can use bootstrapping to reset the noise in a homomorphic ciphertext to allow the evaluation of circuits with arbitrary multiplicative depth. However, SEAL does not support bootstrapping, and it is still very inefficient in HElib and does not result in faster runtimes for the $\mathbb{Z}_2$ ciphers compared to Pasta in* HHE*. Thus, we omit explicit bootstrapping benchmarks in this paper.*

LWE Results. As discussed in Section 3.3, LWE-native encryption [CDK+21] has larger ciphertext expansion than HHE (Concretely an expansion factor of $\frac{881}{60} = 14.68$ for the used parameter set). However, its smaller multiplicative depth allows it to use the same HE parameters as just using homomorphic encryption, resulting in a smaller runtime overhead. Both using HE and using LWE-native encryption require sampling Gaussian noise during encryption. Constrained devices, however, often do not have access to a reliable source of randomness. Therefore, we also list the number of random Gaussian words required on the client side to perform the encryption in Table 6.1. HHE does not require sampling random values during encryption, which is why using HHE is the preferable choice on constrained devices without a reliable source of randomness. Consequently, the first benchmarks show that HHE has the preferable effect on the client side due to not requiring sampling Gaussian randomness, having faster plain performance, and requiring less communication. The LWE-native encryption approach, however, leads to a faster server side evaluation due to having a smaller multiplicative depth.

Client Side Performance. Table 6.1 clearly shows that just using homomorphic encryption would result in unnecessarily large client-to-server communication. To further demonstrate the performance loss, we show the combined client timings (for encryption and client-to-server communication) for different network speeds in Figure 6.1. We depict timings for using only HE, HHE using Pasta-3, and for the LWE-native approach. We omit HHE using $\mathbb{Z}_2$ ciphers, since they result in infeasible server runtimes. Figure 6.1 shows that using HHE always results in the fastest client-side latencies, especially for network speeds below 1 Gbps (the average LTE upload speed in the USA is 5 Mbps[1]) where HE runtime is fully dominated by the data transmission.

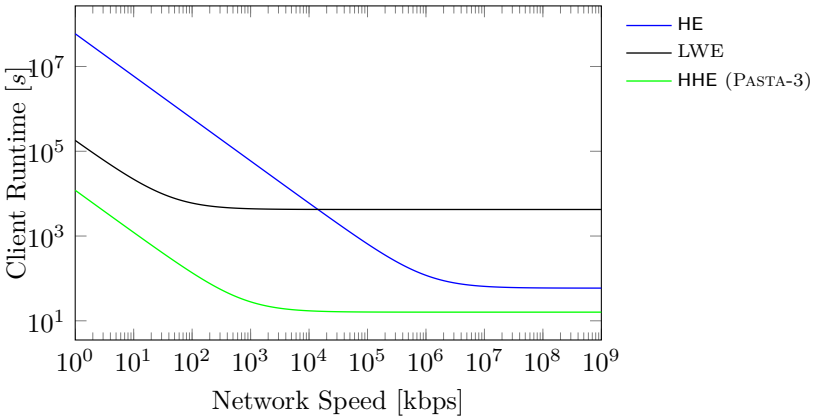


Figure 6.1: Encryption + upload time of HE, HHE with Pasta, and LWE-native encryption [CDK+21] depending on network speed.

[1] https://www.verizon.com/articles/4g-lte-speeds-vs-your-home-network/

Conclusion. To summarize, if the encryption time on a client is the bottleneck, then using HHE with an $\mathbb{F}_p$ cipher (in this case Pasta-3) is the preferred choice. Only HHE using traditional $\mathbb{Z}_2$ ciphers (e.g., AES) is faster, but using them results in infeasibly long server-side computations. Furthermore, if the client bandwidth is the bottleneck, then HHE has a considerable advantage. The concrete communication advantage depends on the HE parameters. For our example use case HE requires a factor of 4936× more communication than HHE, the LWE-native approach a factor of 14.86×. Since HHE has the largest server-side runtime overhead, using HHE has the best effect on constrained clients or in slow network settings. The choice of the symmetric cipher used in HHE has similar effects on the client side (all have ciphertext expansion of 1), but severely affects the server-side runtime. Consequently, we investigate the server-side computation using different symmetric ciphers in the remainder of the paper, starting with the inefficiency of $\mathbb{Z}_2$ ciphers.

We further want to note, that for sake of simplicity we assume plaintexts to have the exact size of the used prime p (i.e., 60 bit) in this first example of HHE. In practice, the exact plaintext space might be smaller to prevent overflows in $\mathbb{F}_p$ during homomorphic computations. Thus, while still instantiating the symmetric cipher and HE scheme with a 60 bit plaintext prime p, the actually used plaintexts might be significantly smaller. Since the size of HE and LWE ciphertexts in Table 6.1 do not depend on p but on a ciphertext modulus q, the size of the used plaintext being undetectable once encrypted, and the need to instantiate $\mathbb{F}_p$ ciphers with the same prime to allow decryption under HE, the values in Table 6.1 do not change for HE, LWE and HHE with the $\mathbb{F}_p$ cipher Pasta-3. Only the $\mathbb{Z}_2$ ciphers will benefit from the smaller plaintexts with smaller client to server communication. However, since the server side computation with its too large multiplicative depth is infeasibly long due to the need for binary circuits, this small advantage on the client side plays no role in practice.

5 Inefficiency of $\mathbb{Z}_2$ Ciphers

In this section, we evaluate the usability of proposed symmetric ciphers for HHE. We focus on boolean ciphers with plaintexts in $\mathbb{Z}_2$ since these are the majority of ciphers proposed for HHE. The main design criterion of all these ciphers is to reduce the AND depth of the decryption circuit.

Hybrid homomorphic encryption aims to reduce the communication overhead for outsourcing computations to a cloud. Therefore, we investigate not only the performance of the decryption circuit of each cipher under homomorphic encryption, but also the performance of the cipher in a complete HHE use case. The use case we benchmark in this section is very small, concretely a server which computes $\vec{r} = M \cdot \vec{x} + \vec{b}$, where $\vec{r}, \vec{x}, \vec{b} \in \mathbb{Z}_{2^{16}}^5$ and $M \in \mathbb{Z}_{2^{16}}^{5\times5}$, i.e., a 5×5 matrix-vector multiplication of 16-bit integers. The matrix M and the vector $\vec{b}$ are private and owned by the server, whereas $\vec{x}$ is a private vector owned by the client. The client uses HHE to send $\vec{x}$ in encrypted form to the server, and will get $\vec{r}$ in encrypted form as a result. As described above, the choice of a

cipher over $\mathbb{Z}_2$ also requires that we compute the integer matrix multiplication over $\mathbb{Z}_2$. This requires the implementation of binary circuits for addition[1] and multiplication, which have a much higher AND depth than performing the same operations over $\mathbb{F}_p$. Despite being only a very small matrix multiplication (5×5 with 16-bit integers), our benchmarks (given later in this section) show that the evaluation is already very slow, making it infeasible for $\mathbb{Z}_2$ ciphers to be applied to real-world statistics or machine learning use cases with multiple chained matrix multiplications of larger integers with matrices consisting of hundreds of entries.

5.1 A Zoo of $\mathbb{Z}_2$ Ciphers

In this paper, we benchmark 128-bit security instances of the ciphers LowMC [ARS+15], Rasta [DEG+18], Agrasta [DEG+18], Dasta [HL20], Kreyvium [CCF+16] (as stream cipher and in depth-bounded CTR mode), and FiLIP [MCJ+19]. In Table 6.2 we summarize the parameters of the ciphers in their respective modes of operation.[2]

Table 6.2: Parameters of the benchmarked $\mathbb{Z}_2$ ciphers in their respective modes of operations in bits.

Cipher	Blocksize	Keysize	Rounds	AND-depth
LowMC	256	128	14	14
Rasta-5	525	525	5	5
Rasta-6	351	351	6	6
Dasta-5	525	525	5	5
Dasta-6	351	351	6	6
Agrasta	129	129	4	4
Kreyvium	-	128	-	-
Kreyvium-12	46	128	-	12
Kreyvium-13	125	128	-	13
FiLIP-1216	-	16384	-	3
FiLIP-1280	-	4096	-	4

We start this section by first introducing Rasta, which is the baseline for many other proposals, before we discuss some followup-ciphers not included in our benchmark comparisons.

Rasta. Rasta is a family of stream ciphers, in which a permutation is applied to the secret key to produce the keystream. The permutation consists of several rounds of affine layers and an S-box instantiated with the

[1] We implemented depth-optimized carry-lookahead adders (CLA) in HElib and SEAL, and standard ripple carry adders (RCA) in TFHE.

[2] Kreyvium, FiLIP-1216, and FiLIP-1280 are stream ciphers without defined block size. In our benchmarks, we therefore define one block to be 46 bits for Kreyvium and 64 bits for both FiLIP instances since we believe just benchmarking one bit is not representitive.

χ-transformation [Dae95]. The main design criteria of Rasta is that each affine layer is pseudorandomly generated from an extendable-output function (XOF) [NIS15] seeded with a nonce N and the block counter i. This essentially prevents all attacks which require multiple plaintext/ciphertext pairs and allows to build a cipher with a low number of rounds. We depict the Rasta permutation in Figure 6.2.

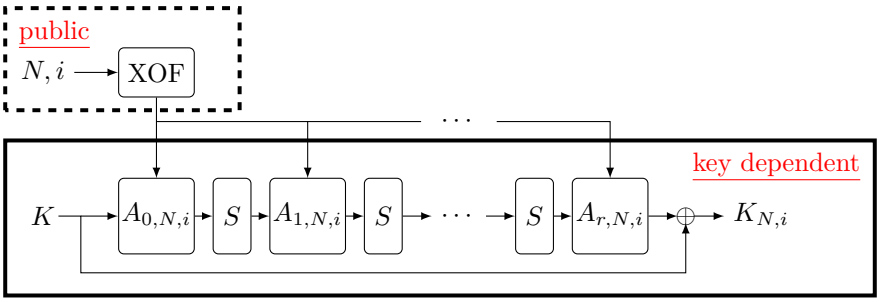


Figure 6.2: The r-round Rasta construction to generate the keystream $K_{N,i}$ for block i under nonce N with affine layers $A_{j,N,i}$. The picture is taken from [DEG+18].

Fasta. Shortly after first releasing our paper to the public the cipher Fasta [CIR22] was published. Fasta is an optimization of Rasta in which the linear layer is adapted for faster packed evaluation for specific HElib parameters. However, since not every HE library (such as SEAL) allows packing for $\mathbb{Z}_2$ ciphers, and Fasta's optimization directly benefits from very specific HElib parameters and does not translate to every use case or library, we do not include it in our comparisons. For benchmarks comparing Rasta to Fasta using our implementation framework we refer to [CIR22].

Chaghri. Very recently, another boolean ciphers, namely Chaghri [AMT22], was proposed in the literature. Following the Marvelous [AAB+20] design strategy, each round of Chaghri has a AND-depth of 2. Together with its comparably high number of rounds, Chaghri's total depth is 16, making it significantly deeper then any other symmetric cipher over $\mathbb{Z}_2$ discussed in our work. Furthermore, this design is heavily optimized for using a special type of packing, where each slot encodes polynomials in $\mathbb{F}_{2^{63}}$. While this allows them to use Frobenius automorphisms to evaluate x^{2^k} for free, it also has the disadvantage that no technique (to the best of our knowledge) is known to homomorphically extract bits from these polynomials. Consequently, one either has to pack only one bit into these polynomials severely limiting throughput, or Chaghri can only be applied to very specific use cases using this packing. Furthermore, this type of packing is not available in some libraries, such as SEAL. Finally, each Chaghri round consists of two multiplications with 3×3 MDS matrices, which have to be

implemented over polynomials with 63 elements, which is very expensive without this packing.

Besides, Chaghri was broken shortly after publication, which is also confirmed by the authors [AMT22]. The attack [LAW+22] works in practical time and increases the number of rounds from 8 to at least 14. Based on the benchmarks given in [AMT22], this increase by 75 % would result in a performance close to AES (i.e., the only other cipher they consider in their paper), which is severely outperformed by any other $\mathbb{Z}_2$ cipher proposed for HHE. However, the authors of [LAW+22] propose a modification of Chaghri, which allows to keep the 8 rounds while maintaining roughly the same efficiency, which was then later adopted by the authors of Chaghri [AMT22].

For all these reasons, Chaghri does *not* provide better performances than any other cipher considered in this paper, and we do not include it in our performance evaluation.

5.2 SEAL Benchmarks

In this section we discuss the benchmarks for the $\mathbb{Z}_2$ ciphers in SEAL, for benchmarks in HElib and TFHE we refer to Section B.2 and Section B.1 respectively. In SEAL, the available noise budget (i.e., how much further noise can be introduced until decryption will fail) depends on the ciphertext modulus q. However, big moduli q require a big degree N of the cyclotomic reduction polynomial for security. N, which is always a power of two, has a severe impact on the performance of the HE scheme. While a larger N allows for larger q to increase the noise budget, it significantly increases the runtime of homomorphic operations.

In Table 6.3 we present the benchmarks for the SEAL library, for homomorphically decrypting only one block, and for the small HHE use case, i.e., the 16-bit 5×5 affine transformation. For both benchmarks we give timings for homomorphically encrypting the symmetric key and homomorphically decrypting the symmetric ciphertexts (i.e., decompressing the HHE ciphertext) for the smallest N allowing enough noise budget for correct evaluation. We parameterize q such that the HE scheme has a security of 128 bits. For the HHE use case we additionally give the runtime for the affine transformation. Since SEAL does not allow to use packing with plaintexts in $\mathbb{Z}_2$, all implementations are bitsliced (i.e., one HE ciphertext per bit).

5.3 Discussion

Our benchmarks show that the runtime of the whole HHE use case (including cipher evaluation) using the $\mathbb{Z}_2$ ciphers is high, despite the tested use case being small. This emphasizes the requirement of $\mathbb{F}_p$ ciphers for HHE with integer use cases. In SEAL and HElib, the fastest ciphers are the ciphers based on the Rasta design strategy (Rasta, Dasta, Agrasta), with Agrasta being the fastest due to its small multiplicative depth. Only FiLIP has better noise propagation. However, due to its large symmetric key and long evaluation time, it is not competitive in the libraries we benchmarked. For figures comparing the

Table 6.3: Benchmarks of the $\mathbb{Z}_2$ ciphers in the SEAL library (security level $\lambda = 128$ bit).

Cipher	1 Block			Small HHE use case			
	N	Enc. Key s	Decomp. s	N	Enc. Key s	Decomp. s	Use Case s
LowMC	16384	1.75	613.9	32768	6.12	2 702.7	1 202.1
Rasta-5	8192	2.12	135.9	32768	25.4	2 618.5	1 201.8
Rasta-6	8192	1.42	88.5	32768	17.1	1 802.0	1 199.6
Dasta-5	8192	2.20	134.1	32768	25.4	2 594.0	1 209.2
Dasta-6	8192	1.49	88.7	32768	17.2	1 811.8	1 209.8
Agrasta	8192	**0.534**	**16.3**	16384	**1.76**	**76.2**	**241.0**
Kreyvium	16384	1.84	412.8	32768	6.17	2 028.5	1 210.7
Kreyvium-12	16384	1.75	414.8	32768	6.30	3 925.8	1 217.9
Kreyvium-13	16384	1.83	442.1	32768	6.18	1 999.0	1 199.3
FiLIP-1216	8192	66.1	1 064.7	16384	223.9	6 619.0	244.5
FiLIP-1280	8192	16.7	1 251.6	16384	56.0	7 783.2	242.0

runtime of HHE in SEAL and HElib and a comparison to $\mathbb{F}_p$ ciphers, we refer to Section 9.1.

6 Designing an Efficient Cipher for HHE over $\mathbb{F}_p$

Following the results from the previous section, we now want to design an efficient cipher for HHE for integer use cases. We will first have a look at existing related work (Section 6.1), before we identify the cost metric of the HE schemes in more detail (Section 6.2) and design a cipher accordingly.

6.1 Related Work

Masta. In an independent and concurrent work another symmetric cipher over $\mathbb{F}_p^t$ created for HHE use cases is introduced, namely Masta [HKC+20]. In their work, the $\mathbb{F}_p$ cipher Masta is proposed to increase throughput compared to boolean ciphers when evaluated under HE and its decryption runtime under HE is compared to Rasta when implemented in the HElib library.[1]

Masta can be seen as a direct translation of Rasta (Figure 6.2) to $\mathbb{F}_p^t$, with the exception of a different strategy in sampling random invertible matrices. Their approach involves sampling a random polynomial $m \in \mathbb{Z}_p[X]/(X^t - \alpha)$ and translating m into a matrix M. This matrix is then invertible by design and they only have to sample s field elements $\in \mathbb{F}_p$. Even though the S-box used in Rasta is in general not a permutation over $\mathbb{F}_p^t$, and therefore limits the possible outputs

[1] We suspect that in [HKC+20] the authors only benchmark a word-sliced HE implementation of Masta, which is why our packed implementation of Masta is significantly faster.

of the S-box layer in Masta,[1] the designers did not consider any additional changes to the baseline design and do not leverage any advantages of HE over fields $\mathbb{F}_p$. In this paper we consider the two 128-bit security instances of Masta with the lowest depth and use Shake128 to pseudorandomly generate all affine layers.

Since Masta does not consider any additional changes to Rasta based on the properties of BGV/BFV, and the S-box is not a permutation in $\mathbb{F}_p$, we aim to design a more optimized cipher in the next sections.

Hera. Another $\mathbb{F}_p$ cipher, namely Hera [CHK+21], was proposed in the literature alongside a framework for applying HHE to CKKS. Contrary to Rubato, Hera can also be applied to BFV and BGV which is why we also consider it in our comparisons.

The main design rationale behind Hera is to apply the Rasta design strategy in a different way to also benefit from the prevention of statistical attacks by randomizing the cipher, but with less preprocessing cost. They do this by fixing the affine layers and randomizing the key schedule by multiplying the key elements with pseudorandomly sampled $\mathbb{F}_p$ elements. They also fix a small statesize of just 16 words and a round number of 5 for 128 bit security and instantiate their linear layers with efficient AES-like matrices. As nonlinear layer they use the well-known cubing layer (see Section 6.4).

6.2 Cost Metrics

The goal is to design an efficient cipher for HHE over $\mathbb{F}_p^t$ with $2^{16} < p < 2^{60}$.[2] Since in both BGV and BFV (and their respective implementations in SEAL and HElib) the most significant performance metric is the multiplicative depth due to the absence of an efficient bootstrapping operation, our main goal is to reduce this metric. Since every round contributes to the multiplicative depth, and therefore to the overall noise consumption during a homomorphic evaluation of the cipher, we aim to design a secure cipher with a minimal number of rounds. Further, high-degree polynomials have a large multiplicative depth, and hence we consider low-degree S-boxes. Meeting both of these requirements usually requires a large state size for security. However, large state sizes lead to a high runtime of the cipher evaluation, especially in the linear layers. Therefore, our design will have to balance noise consumption and runtime to be efficiently usable in HHE. Furthermore, most HE applications leverage packing (Section 2.1) to increase performance, which is why we also aim to design a packing-friendly cipher which produces packed homomorphically encrypted ciphertexts. For a comparison of a word-sliced implementation of our final design to a packed implementation we

[1] While a concrete attack is not known, reducing the output entropy of an internal component may lead to unwanted effects in the final output of the function, and thus reduce the security against still unknown attacks. Therefore, most ciphers known in the literature rely on permutations and do not restrict the output entropy.

[2] SEAL does not allow larger field sizes.

refer to Appendix C. There we also compare a word-sliced implementation of Hera to Pasta.

Cost of HE Operations. In Table 6.4 we summarize the cost of each HE operation in SEAL and HElib. Note that the key switching operation is free in terms of noise in SEAL, whereas it adds noise to the ciphertext in HElib. Key switching is required after a ciphertext-ciphertext multiplication and after an homomorphic Galois automorphism (required for rotation), which is why these operations require more noise in HElib. For both libraries the noise consumption depends on the size of the prime p, with larger p implying higher noise consumption, especially in pt-ct and ct-ct multiplications. Therefore, one cannot consider plaintext-ciphertext multiplications as negligible when working over $\mathbb{F}_p$ and we also have to consider the plaintext-ciphertext multiplicative depth when designing an efficient cipher over $\mathbb{F}_p$.

Table 6.4: Cost of HE operations in SEAL and HElib.

	SEAL		HElib	
Operation	Noise	Runtime	Noise	Runtime
pt-ct Add	negligible	cheap	negligible	cheap
ct-ct Add	negligible	cheap	negligible	cheap
pt-ct Mul	moderate	cheap	moderate	cheap
ct-ct Mul	expensive	expensive	expensive	expensive
Automorphism	negligible	expensive	moderate	expensive

Remark 6. *In the future, more efficient bootstrapping implementations might become available e.g. due to efficient* HE *hardware accelerators which implement this feature. Depending on the concrete efficiency of bootstrapping, the optimization angle of* HE *might shift from minimizing the multiplicative depth to minimizing the most expensive* HE *operations, such as multiplications. In this case, symmetric ciphers optimized for* HHE *will be allowed to have more rounds with higher degree S-Boxes and will more closely look like some ciphers optimized for e.g. MPC where the total number of multiplications is the main bottleneck.*

6.3 Design Basis

Since our $\mathbb{Z}_2$ benchmarks indicate that designs based on Rasta are the preferred choice, we first consider an $\mathbb{F}_p^t$ version of Rasta with equal text/key size, and then modify it for security and efficiency. In the following, we analyze several candidates for each of the operations defining the cipher, and we also determine their implementation efficiency. Based on these results, we then design Pasta in Section 7.

6.4 S-Box

The original Rasta design uses the χ-transformation [Dae95] over $\mathbb{Z}_2^t$ as a single nonlinear layer. However, the χ-function is in general not a permutation when working over $\mathbb{F}_p^t$, which is why we consider alternative building blocks. Since the affine layers in a Rasta-based permutation are pseudorandomly generated for each new block, many attacks (mainly statistical attacks) are already prevented. Hence, the main goal of the S-box in this setting is to provide a sufficiently high degree to prevent algebraic attacks – the concrete structure of the S-box plays a comparably minor role. Consequently, we propose invertible low-degree S-boxes, describe how they can be efficiently implemented in a packed homomorphic evaluation, and compare their efficiency. Despite not being a permutation, Masta still uses the χ-function naturally defined over $\mathbb{F}_p^t$, which is why we include it in our comparison.

χ-S-box. The χ-S-box is defined as

$$[\chi(\vec{x})]_i = x_i + x_{i+2} + x_{i+1} \cdot x_{i+2} = x_i + x_{i+2} \cdot (1 + x_{i+1}).$$

The indices in the χ-S-box are taken modulo t, which is why χ can be efficiently evaluated using rotations, i.e.,

$$\chi(\vec{x}) = \vec{x} + \mathtt{rot}_2(\vec{x}) \odot (\vec{1} + \mathtt{rot}_1(\vec{x})).$$

This works if the rotation is cyclic for the vector of size t. However, once encrypted, homomorphic rotations are cyclic over a larger vector of size n. Hence, we need to simulate cyclic rotation by preprocessing the state first. However, the resulting vector has more than t elements, which can influence further homomorphic operations. Thus, one has to apply a masking multiplication afterwards with a mask $\vec{m} = \vec{1} \in \mathbb{F}_p^t$:

$$\begin{aligned} \vec{x}' &= \vec{x} + \mathtt{rot}_{(-t)}(\vec{x}) \\ \Rightarrow \chi(\vec{x}) &= \left(\vec{x}' + \mathtt{rot}_2(\vec{x}') \odot (\vec{1} + \mathtt{rot}_1(\vec{x}'))\right) \odot \vec{m}. \end{aligned}$$

Cube S-box. Given a prime p, $\gcd(p-1, 3) = 1$, let

$$[S(\vec{x})]_i = (x_i)^3.$$

We recall that the cube S-box is the invertible power map with the smallest degree, and it can be efficiently evaluated by simply applying two homomorphic multiplications which affect the state elementwise, i.e., $S(\vec{x}) = \vec{x} \odot \vec{x} \odot \vec{x}$.

Feistel-Like S-Box (via a Quadratic Function).

$$[S'(\vec{x})]_i = \begin{cases} x_i & \text{if } i = 0, \\ x_i + (x_{i-1})^2 & \text{otherwise,} \end{cases}$$

The Feistel-like S-box can also efficiently be implemented using rotations, i.e.,

$$S'(\vec{x}) = \vec{x} + \left(\mathtt{rot}_{(-1)}\left(\vec{x}\right) \odot \vec{m}\right)^2,$$

where $\vec{m} \in \mathbb{F}_p^t$ is a masking vector $\vec{m} = [0, 1, \ldots, 1]^T$.

Alternative Feistel-Like S-Box (via the χ-Function).

$$[S'(\vec{x})]_i = \begin{cases} x_i & \text{if } i \leq 1, \\ x_i + x_{i-1} \cdot x_{i-2} & \text{otherwise,} \end{cases}$$

The alternative Feistel-like S-box can also efficiently be implemented using rotations, i.e.,

$$S''(\vec{x}) = \mathtt{rot}_{(-1)}(\vec{x}) \odot \mathtt{rot}_{(-2)}(\vec{x}) \odot \vec{m} + \vec{x},$$

where $\vec{m} \in \mathbb{F}_p^s$ is a masking vector $\vec{m} = [0, 0, 1, \ldots, 1]^T$.

S-Box Cost Comparison

All S-box designs can efficiently be implemented on packed HE ciphertexts and require only a constant number of homomorphic operations independent of the state size. A summary of required homomorphic operations as well as the multiplicative depths of the different S-boxes is given in Table 6.5.

Table 6.5: HE operations and depth of different S-boxes.

S-box	pt-ct Add	ct-ct Add	pt-ct Mul	ct-ct Mul	Rot	pt-ct Depth	ct-ct Depth
χ	1	2	1	1	3	1	1
S	-	-	-	2	-	-	2
S'	-	1	1	1	1	1	1
S''	-	1	1	1	2	1	1

Based on Table 6.5, we decide to choose the Feistel S-box S' as the main S-box for our nonlinear layers, and to use the cube S-box S to increase the degree of our cipher to combat linearization attacks and reduce the state size of the cipher. We further explore the choice of the two different S-boxes in Section 8.4.

6.5 Linear Layer

In Rasta, the homomorphic runtime is dominated by the linear layer. In this section we discuss how to efficiently implement matrix-vector multiplications on packed homomorphic ciphertexts and introduce optimizations to reduce the homomorphic evaluation time.

Choice of Random Matrices

In the original Rasta design, each random $t \times t$ matrix is directly sampled and checked for invertibility. However, doing the invertibility check is expensive in $\mathbb{F}_p$ in terms of computational complexity. Therefore, in Pasta we choose a different approach and generate each matrix as a sequential matrix [GPP11; GPP+11] (Section 7). These matrices are invertible by design and only require to sample t field elements and performing $t \cdot (t-1)$ field multiplications and $(t-1)\cdot(t-1)$ field additions. Compared to sampling polynomials $m_i \in \mathbb{Z}_p[X]/(X^t - \alpha)$ and translating them to matrices M_i (like in Masta), sequential matrices require to sample equally many field elements, but need more field additions and multiplications. Sampling sequential matrices is thus slower with respect to the method used in Masta, but it comes with the cryptographic advantage of having less structure (see Section 8). Contrary to Hera, we do not fix the matrices and randomize key schedules due to the fact that in a packed implementation one can not leverage advantages of specially chosen matrices, such as implementation via only additions, and plain performance is insignificant compared to HE evaluation runtime.

Babystep-Giantstep Matrix-Vector Multiplication

The most efficient way of evaluating the product between a plain matrix and an encrypted packed vector in HE is using the babystep-giantstep optimized diagonal method [HS14; HS15; HS18]:

$$M\vec{x} = \sum_{k=0}^{t_2-1} \mathtt{rot}_{(kt_1)} \left(\sum_{j=0}^{t_1-1} \mathtt{diag}'_{(kt_1+j)}(M) \odot \mathtt{rot}_j(\vec{x}) \right), \tag{6.1}$$

where $t = t_1 \cdot t_2$, $\mathtt{diag}'_i(M) = \mathtt{rot}_{(-\lfloor i/t_1 \rfloor \cdot t_1)}(\mathtt{diag}_i(M))$, and $\mathtt{diag}_i(M)$ expresses the i-th diagonal of a matrix M in a vector of size t, with $i = 0$ being the main diagonal. Note that $\mathtt{rot}_j(\vec{x})$ only has to be computed once for each $j < t_1$. Therefore, a matrix multiplication requires $t_1 + t_2 - 2$ rotations, t plaintext-ciphertext multiplications, and $t - 1$ additions, and the total depth is 1 plaintext-ciphertext multiplication. Thus, we add words to the final state size of our design for efficiency if t does not nicely split into $t = t_1 \cdot t_2$. Compared to the number of homomorphic operations required to evaluate the S-boxes (Table 6.5), it is clear that the runtime of the homomorphic evaluation of our cipher is dominated by the linear layer.

Splitting the State

The babystep-giantstep algorithm dominates the runtime of the homomorphic Pasta evaluation and scales with the state size. Therefore, we propose to evaluate two individual instances of our cipher with state size t in parallel, with an efficient mixing step after each affine layer, allowing for an overall smaller state size. The final output of the design is then the output of the first half, and the second half

is discarded. The result is a cipher with the following properties: *(1)* The state size $s = 2 \cdot t$ is an even number and we truncate t words at the end. *(2)* Instead of evaluating one large $s \times s$ matrix multiplication we perform two smaller $t \times t$ matrix multiplications. *(3)* The S-box is applied on both branches individually. *(4)* The key has now double the size of the keystream. The latter has no effect on the HHE use case, since a packed homomorphic design still requires only one homomorphic ciphertext, with a size independent to the number of encoded words. However, we can use the inner structure of homomorphic ciphertexts to parallelize both cipher evaluations, cutting the runtime down to an evaluation of one cipher instance of state size t.

Inner Structure of HE ciphertexts. In R-LWE based homomorphic encryption schemes (like BFV and BGV) the plaintexts are polynomials $\in R_p = \mathbb{F}_p[X]/\Phi_m(X)$, with $\Phi_m(X)$ being the m-th cyclotomic polynomial. Using packing (Section 2.1) one can encode a vector of integers into one polynomial, homomorphic additions and multiplications then affect these vectors element-wise. Further, one can use Galois automorphisms to permute the encoded vector. Thus, the encoded vector can be seen as a hypercube [HS14] and an automorphism rotates the data along one dimension. The precise structure of this hypercube depends on the choice of $\Phi_m(X)$. In general, it is possible to use these automorphisms to create linear rotations over the encrypted vector, but this requires masking multiplications [HS14], which when evaluated homomorphically require noise budget. In terms of implementation efficiency, $\Phi_{2n}(X) = X^n + 1$, for n being a power of two, is a good choice. This polynomial is negacyclic and allows efficient polynomial multiplications via a negacyclic number theoretic transformation (NTT). For this reason, the homomorphic encryption standardization project[1] recommends using these power-of-two cyclotomic rings. Consequently, SEAL only implements HE with those rings and Masta is defined to use these rings as well [HKC+20]. The hypercube generated by such rings also has a nice structure: It corresponds to a matrix of two rows, each of size $\frac{\#\text{slots}}{2}$. Galois automorphisms can then directly be used to either linearly rotate both rows at once or rotate all columns simultaneously, i.e.,

$$\begin{bmatrix}\vec{x}_L \\ \vec{x}_R\end{bmatrix} \overset{\text{encode}}{\rightarrow} x \in R_p : \qquad \tau_{3^i}(x) \overset{\text{decode}}{\rightarrow} \begin{bmatrix}\texttt{rot}_i(\vec{x}_L) \\ \texttt{rot}_i(\vec{x}_R)\end{bmatrix}, \qquad \tau_{n-1}(x) \overset{\text{decode}}{\rightarrow} \begin{bmatrix}\vec{x}_R \\ \vec{x}_L\end{bmatrix},$$

for the Galois automorphism $\tau_i : a(X) \mapsto a(X^i)$.

Parallelizing Two Cipher Evaluations. In two state-of-the-art integer HE cryptosystems (BFV and BGV) we can use this inner structure of power-of-two homomorphic ciphertexts to parallelize both branches of our cipher. When encrypting the secret key and encoding vectors in the affine layer, one has to encode the vectors affecting the first branch of the cipher into the first row of the homomorphic ciphertext, and vectors affecting the second branch into the

[1] https://homomorphicencryption.org/

second row. As a result, all homomorphic operations are applied in parallel to both branches.

Efficient Linear Layer. For security, we have to mix both branches of our cipher after each affine transformation. An efficiently implementable linear layer, which is also invertible, is the following matrix multiplication:

$$\begin{bmatrix}\vec{y}_L\\ \vec{y}_R\end{bmatrix} = \begin{bmatrix}2\cdot I & I\\ I & 2\cdot I\end{bmatrix}\cdot\begin{bmatrix}\vec{x}_L\\ \vec{x}_R\end{bmatrix} = \begin{bmatrix}\vec{x}_L\\ \vec{x}_R\end{bmatrix} + \begin{bmatrix}\vec{x}_L\\ \vec{x}_R\end{bmatrix} + \begin{bmatrix}\vec{x}_R\\ \vec{x}_L\end{bmatrix},$$

where I is the $t\times t$ identity matrix. This can be implemented by two homomorphic additions and a homomorphic rotation.

In Table 6.6 we compare the cost of the new linear layer (two parallel instances of state size t) to the cost of one larger linear layer of size $s = 2\cdot t$. The new linear layer effectively requires half the homomorphic additions and multiplications, and choosing t such that it splits nicely into $t = t_1 \cdot t_2$ the number of rotations is also halved.

Table 6.6: Homomorphic operations and multiplicative depth of the linear layers, with $t = t_1 \cdot t_2$ and $2\cdot t = s_1 \cdot s_2$.

Linear Layer	pt-ct Add	ct-ct Add	pt-ct Mul	ct-ct Mul	Rot	pt-ct Depth	ct-ct Depth
Split and Mix	1	$t+2$	t	-	t_1+t_2	1	-
No Splitting	1	$2\cdot t$	$2\cdot t$	-	s_1+s_2-1	1	-

6.6 Total Homomorphic Operations and Multiplicative Depth

In Table 6.7 we summarize the number of homomorphic operations and the multiplicative depth of each individual part of our resulting new cipher, dubbed Pasta, as well as the total count for Pasta-3 (3 rounds) and Pasta-4 (4 rounds). The table also highlights that the multiplicative depth of Pasta, and therefore its noise consumption, only depends on the number of rounds. Further, the runtime of homomorphically evaluating Pasta is dominated by the affine layer and scales with the state size and the number of rounds.

Table 6.7: Homomorphic operations and multiplicative depth of Pasta, with $t = t_1 \cdot t_2$.

	pt-ct Add	ct-ct Add	pt-ct Mul	ct-ct Mul	Rot	pt-ct Depth	ct-ct Depth
Affine	1	t	t	-	t_1+t_2-1	1	-
Mix	-	2	-	-	1	-	-
S'	-	1	1	1	1	1	1
S	-	-	-	2	-	-	2
Round	1	$t+3$	$t+1$	1	t_1+t_2+1	2	1
Last Round	1	$t+2$	t	2	t_1+t_2	1	2
Pasta-3	4	$4t+10$	$4t+2$	4	$4(t_1+t_2)+2$	6	4
Pasta-4	5	$5t+13$	$5t+3$	5	$5(t_1+t_2)+3$	8	5

7 Pasta Specification

Here we provide the full PASTA specification. PASTA is a family of stream ciphers which applies the PASTA-π permutation under a nonce N and a block counter i to the secret key, followed by a truncation, to produce the final keystream. Keystream generation is shown in Figure 6.3. For a prime p s.t. $\gcd(p-1,3)=1$,[1] a PASTA encryption is defined as

- $\mathsf{KGen}()$: $\mathsf{sk} \xleftarrow{\$} \mathbb{F}_p^{2t}$
- $\mathsf{Enc}_{\mathsf{sk}}(\vec{m}, N)$: To encrypt the message $\vec{m} \in \mathbb{F}_p^l$ under the secret key sk and nonce N, parse $\vec{m} = \vec{m}_0||\vec{m}_1||...||\vec{m}_j$ with $\vec{m}_i \in \mathbb{F}_p^t$ and return $\vec{c} = \vec{c}_0||\vec{c}_1||...||\vec{c}_j$, where $\vec{c}_i = \vec{m}_i + \text{left}_t(\text{PASTA-}\pi(\mathsf{sk}, N, i))$, where $\text{left}_t(\cdot)$ returns the first t words.
- $\mathsf{Dec}_{\mathsf{sk}}(\vec{c}, N)$: To decrypt the ciphertext $\vec{c} \in \mathbb{F}_p^l$ using the secret key sk and nonce N, parse $\vec{c} = \vec{c}_0||\vec{c}_1||...||\vec{c}_j$ with $\vec{c}_i \in \mathbb{F}_p^t$ and return $\vec{m} = \vec{m}_0||\vec{m}_1||...||\vec{m}_j$, where $\vec{m}_i = \vec{c}_i - \text{left}_t(\text{PASTA-}\pi(\mathsf{sk}, N, i))$, where $\text{left}_t(\cdot)$ returns the first t words.

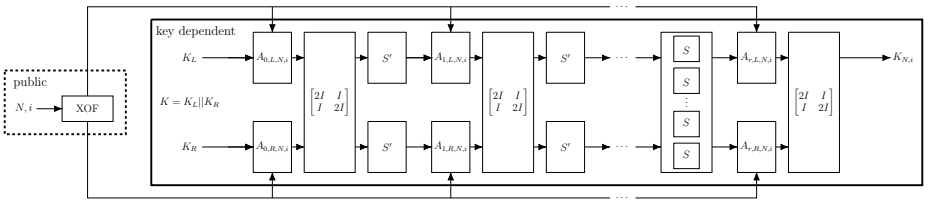


Figure 6.3: The truncated r-round PASTA-π permutation to generate the keystream $K_{N,i}$ for block i under nonce N with affine layers $A_{j,k,N,i}$.

The permutation PASTA-$\pi(\vec{x}, N, i)$ on a vector $\vec{x} \in \mathbb{F}_p^{2t}$, thereby, is defined as

$$\begin{aligned}\text{PASTA-}\pi(\vec{x}, N, i) = A_{r,N,i} \circ S_{\text{cube}} \circ A_{r-1,N,i} \circ S_{\text{feistel}} \\ \circ A_{r-2,N,i} \ldots \circ A_{1,N,i} \circ S_{\text{feistel}} \circ A_{0,N,i}(\vec{x}),\end{aligned} \tag{6.2}$$

where $r \geq 1$ is the number of rounds and where

- S_{feistel} is an S-box layer defined as $S_{\text{feistel}}(\vec{x}) = S'(\vec{x}_L)\|S'(\vec{x}_R)$, where S' over $\mathbb{F}_p^t$ is a Feistel structure defined as

$$\forall l \in \{0, 1, \ldots, t-1\}: \qquad [S'(\vec{y})]_l = \begin{cases} y_l & \text{if } l = 0, \\ y_l + (y_{l-1})^2 & \text{otherwise,} \end{cases}$$

  where $\vec{y} = y_0\|y_1\|\cdots\|y_{t-1} \in \mathbb{F}_p^t$,

[1] We recall that the S-box $S(x) = x^d$ for $d \geq 2$ is invertible over $\mathbb{F}_p$ if and only if $\gcd(p-1, d) = 1$.

- S_{cube} is an S-box defined as $S_{\text{cube}}(\vec{x}) = x_0^3 \| x_1^3 \| \cdots \| x_{s-1}^3$,
- for each $j \in \{0, \ldots, r\}$, $A_{j,N,i}$ is an affine layer

$$A_{j,N,i}(\vec{x}) = \begin{bmatrix} 2 \cdot I & I \\ I & 2 \cdot I \end{bmatrix} \begin{bmatrix} M_{j,L,N,i}(\vec{x}_L) + \vec{c}_{j,L,N,i} \\ M_{i,R,N,i}(\vec{x}_R) + \vec{c}_{j,R,N,i} \end{bmatrix},$$

  where $I \in \mathbb{F}_p^{t \times t}$ is the identity matrix and where $M_{j,L,N,i}, M_{j,R,N,i} \in \mathbb{F}_p^{t \times t}$ and $\vec{c}_{j,L,N,i}, \vec{c}_{j,R,N,i} \in \mathbb{F}_p^t$ are generated for each round from an XOF seeded with a nonce N and a counter i.

To efficiently sample each invertible matrix $M_{j,k,N,i} \in \mathbb{F}_p^{t \times t}$, we sample sequential matrices following [GPP11; GPP+11]. For each $k \in \{L, R\}$, we define $M_{j,k,N,i} := \left(\tilde{M}_{j,k,N,i}\right)^t$, where $\tilde{M}_{j,k,N,i} \in \mathbb{F}_p^{t \times t}$ is defined as

$$\tilde{M}_{j,k,N,i} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_t \end{bmatrix}$$

for $\alpha_1, \ldots, \alpha_t \in \mathbb{F}_p \setminus \{0\}$. $M_{j,k,N,i}$ is an invertible matrix which can be built by sampling t random elements and performing $t \cdot (t-1)$ multiplications and $(t-1) \cdot (t-1)$ additions.

7.1 Concrete Instances

We propose a 3-round instance Pasta-3 as well as a 4-round instance Pasta-4 using Shake128 [NIS15] as XOF. These instances provide at least 128 bits of security for the prime fields $\mathbb{F}_p$ with $\log_2(p) > 16$ and $\gcd(p-1, 3) = 1$. Table 6.8 shows the block and key sizes and compares them to Masta and Hera.

Security Margin. In all cases, we add a security margin to our construction. Concretely, we take the largest number of words s needed for security, we multiply this number by 1.2 for a 20% security margin, and we then take the smallest even integer larger than or equal to that.

Table 6.8: 128 bit security instances of Pasta, Masta, and Hera.

Instance	Rounds	# Key Words	# Plain Words	# Cipher Words	XOF
Pasta-3	3	256	128	128	Shake128
Pasta-4	4	64	32	32	Shake128
Masta-4	4	128	128	128	Shake128
Masta-5	5	64	64	64	Shake128
Hera	5	16	16	16	Shake128

7.2 Comparison to Previous Designs

In this section we summarize Pasta by comparing it to previous designs. Furthermore, in Section 9.3 we discuss $\mathbb{F}_p$ primitives for different use cases and compare them to Pasta.

S-box. Rasta and Dasta use the χ-transformation as single nonlinear layer. Masta uses a translation of χ to $\mathbb{F}_p^t$ as nonlinear layer, despite it being no permutation, and Hera uses the cubing layer. In Pasta we introduce and use two different, bijective S-boxes. This is motivated by the desire of reducing the number of rounds while maintaining a reasonable state size. Having $r-1$ Feistel S-boxes and a final cube S-box with higher degree and depth allows us to build Pasta instances with comparable number of plain/cipher words as Masta with one round less. This implies both, a faster homomorphic evaluation time, as well as less noise consumption compared to Masta. We further explore the choice of two different S-boxes in Section 8.4.

Linear-Layer. Pasta, Rasta, Dasta, and Masta use randomly generated linear layers to mitigate statistical attacks, and Hera has a randomized key schedule for the same reason. While Rasta just samples random invertible matrices, Dasta uses random permutations of the same fixed matrix. Masta on the other hand samples random polynomials and translates them to matrices (which have lots of structure). These methods, however, all just differ in how the matrices are generated and do not effect the homomorphic evaluation time. Contrary, Pasta's linear layer is thoroughly optimized for efficient evaluation in HE. Indeed, instead of generating a $2t \times 2t$ random invertible matrix directly, we pick up $2t$ random elements and construct two sequential matrices $M_i \in \mathbb{F}_p^{t \times t}$ as given in [GPP11; GPP+11]. These two matrices are then combined into one $2t \times 2t$ matrix via a cheap mixing operation, effectively cutting HE runtime in half.

Truncation vs. Feed-Forward. Pasta discards the feed-forward addition of the secret key (as done in Rasta, Dasta, and Masta) in favor of a truncation. This allows to prevent MITM attack in a more efficient way, at the cost of using a larger state. In the packed HE evaluation the truncated words, however, do not influence the runtime since they can be evaluated simultaneously to the non-truncated part of the state.

8 Pasta Security Analysis

Given a certain number of rounds (fixed in advance), our goal is to find the minimum number of key words $s = 2t$ for which we can guarantee security of at least κ bits. If not specified otherwise, $\kappa \approx \log_2(p^s)$. This is slightly different from what is usually done in traditional symmetric cryptanalysis. Indeed, in general, given a state $\mathbb{F}_p^s$ and a security level κ, one looks for the minimum number of

rounds which provide a security level of at least κ bits. Here we modify the approach since one of our main goals is to keep the depth as low as possible, focusing on 3 and 4 rounds.

Remark 7. *The design approach of Pasta is analogous to the one originally proposed for Rasta. For this reason, in many cases we limit ourselves to adapt the security argument proposed for Rasta to Pasta.*

8.1 Truncation versus Feed-Forward

Consider a permutation $F : \mathbb{F}_p^s \to \mathbb{F}_p^s$, and assume it can be split as $F(\cdot) = F_2 \circ F_1(\cdot)$. The advantage of a truncation with respect to a feed-forward operation is that it prevents attacks using the backward direction without requiring a high degree of the inverse round function. Indeed, in the feed-forward case, given $y = F(x)+x$, one can set up a system of equations of the form $F_1(x) = F_2^{-1}(y-x)$. In order to prevent the possibility to solve it using algebraic techniques (e.g., Gröbner bases), we need that both F_1 and F_2^{-1} have a high degree. In the case of truncation, given $y = \text{left}_t(F(x))$, the system of equations becomes $F_1(x) = F_2^{-1}(y \,||\, y')$ for a certain unknown $y' \in \mathbb{F}_p^t$. If t is large enough, the cost of solving it exceeds the security level. However, the overall size of the state must be larger than in the feed-forward case due to losing part of the state.

8.2 Security against Statistical Attacks: Properties of the Linear Layer

As in Rasta, the security against statistical attacks as differential [BS90] and linear [Mat93] ones (besides all their variants, as the truncated differential [Knu94], zero–correlation linear [BW12], impossible differential [BBS99], and so on) is achieved by changing the linear layers at every encryption. In a statistical attack, the attacker makes a statistical analysis of the ciphertexts generated by a set of chosen/known plaintexts in order to break the scheme. This strategy works under the assumption that the ciphertexts are generated via the same encryption scheme. By construction, this is not the case for Rasta–like designs as Pasta, which implies that statistical attacks are not a threat for our design.

Having said that, it is important that the linear layers that instantiate Pasta do not have any weakness that could be exploited for an attack, and that full diffusion is achieved over the entire scheme. For this reason, we study the linear branch number of the random matrices that instantiate Pasta, and we show that it is sufficiently high in general. We recall that the branch number of a matrix is defined as the minimum number of non-zero entries that two t-element mask vectors α and β that satisfy $\alpha = M^T \times \beta$ could have – we refer to [DGG+21] for an overview of correlation analysis in $\mathbb{F}_p$.

Since Pasta's linear layer is defined as

$$\begin{bmatrix} \vec{x}_L \\ \vec{x}_R \end{bmatrix} \mapsto \begin{bmatrix} 2 \cdot I & I \\ I & 2 \cdot I \end{bmatrix} \times \begin{bmatrix} M_{j,L,N,i} \times \vec{x}_L \\ M_{i,R,N,i} \times \vec{x}_R \end{bmatrix},$$

we have the following scenario:

- the fixed matrix $\text{circ}(2,1) \in \mathbb{F}_p^{2\times 2}$ is MDS, which implies that full diffusion among the l-th element of $\vec{x}_L$ and the l-th element of $\vec{x}_R$ is achieved for each $l \in \{0, 1, \ldots, t-1\}$;
- the invertible matrices $M_{j,L,N,i}$ and $M_{j,R,N,i}$ are randomly generated for each new encryption, hence, we cannot guarantee a certain branch number a priori.

For this reason, we estimate a lower bound of the probability that a randomly picked matrix $M \in \mathbb{F}_p^{t\times t}$ allows for transitions on the t-element mask vectors α to β, $\alpha = M^T \times \beta$, where α and β have many zeros (which corresponds to the best scenario for an attacker).

Proposition 1. *Let $M \in \mathbb{F}_p^{t\times t}$ be a random invertible matrix. Its branch number satisfies the following for $p > t \geq 6$:*

$$\Pr[\textit{branch number} \geq t/2] \geq 1 - \frac{2}{p^{t/2-2}} .$$

Proof 1. *By definition:*

$$\begin{aligned}\Pr[\textit{branch number} \geq z] &= 1 - \Pr[\textit{branch number} < z] \\ &= 1 - \frac{\sum_{\alpha,\beta \ s.t. \ \#(\alpha)+\#(\beta)<z} |\mathfrak{X}_{\alpha,\beta}|}{|\mathfrak{I}|}\end{aligned}$$

where

- *$\#(\gamma)$ denotes the number of non-zero entries of the vector γ;*
- *$\mathfrak{X}_{\alpha,\beta}$ denotes the set of invertible matrices that satisfy $\alpha = M^T\beta$;*
- *$\mathfrak{I}$ denotes the set of invertible matrices.*

First of all, we are interested in the number $|\mathfrak{I}|$ of all possible bijective matrices M. A matrix M is bijective, if all its row vectors are linearly independent and different from the all 0 vector. So, for the first row, we have $p^t - 1$ possibilities to choose a row vector. For the second row, we have p^t possibilities to choose the coefficients minus p choices that is just are linear combination of the first row. In the third row, we now have $p^t - p^2$ choices, and so on. So we finally end up with

$$|\mathfrak{I}| = \prod_{i=0}^{t-1} \left(p^t - p^i\right) .$$

Next, we consider the number of matrices M, that allow a transition $\alpha = M^T\beta$ for fixed non-zero α and β. For our goal, we are interested in an upper bound of such a number. Hence, we limit ourselves to consider a weaker condition, namely, that (i) β maps to the first coordinate of α and that (ii) the matrix

is invertible. It is simple to observe that the first condition is satisfied by at most p^{t-1} choices of the coefficients of the first row (note that if $\alpha_0 = 0$, then we exclude the zero-vector as first row of M). By combining this fact with the requirement that M is bijective, we get the number of matrices M that map α to β is upper bounded by

$$|\mathfrak{X}_{\alpha,\beta}| \leq \underbrace{p^{t-1}}_{due\ to\ \alpha = M^T\beta} \cdot \underbrace{\prod_{i=1}^{t-2}\left(p^{t-1} - p^i\right)}_{for\ invertibility} = p \cdot \prod_{i=2}^{t-1}\left(p^t - p^i\right) .$$

Finally, we have a look at how many different masks α and β exist, which have together i non-zero entries. This number is simply given by $(p-1)^i \cdot \binom{2t}{i}$.

Now we have all ingredients we need to bound the probability that a randomly selected matrix M has a branch number smaller than z

$$\begin{aligned}\Pr[branch\ number < z] &\leq \frac{\overbrace{p \cdot \prod_{i=2}^{t-1}\left(p^t - p^i\right)}^{\geq |\mathfrak{X}_{\alpha,\beta}|} \cdot \overbrace{\sum_{i=1}^{z}\left((p-1)^i\binom{2t}{i}\right)}^{=\sum_{\alpha,\beta\ s.t.\ \#(\alpha)+\#(\beta)<z} 1}}{\underbrace{\prod_{i=0}^{t-1}\left(p^t - p^i\right)}_{=|\mathfrak{I}|}} \\ &\leq \frac{\sum_{i=1}^{z}\left((p-1)^i\binom{2t}{i}\right)}{\left(p^t-1\right)\left(p^{t-1}-1\right)} \leq \frac{z(p-1)^z\binom{2t}{t}}{\left(p^t-1\right)\left(p^{t-1}-1\right)} \\ &\leq \frac{2zp^z \cdot t^t}{\left(p^t-1\right)\left(p^{t-1}-1\right)} ,\end{aligned}$$

where $\binom{2t}{t} = \frac{2t\cdot(2t-1)\cdot\ldots\cdot(t+1)}{t!} \leq \frac{(2t)^t}{2^{t-1}} = 2 \cdot t^t$ since $t! \geq 2^{t-1}$. We now set $z = t/2$ and assume that $p > t \geq 6$, we get

$$\Pr[branch\ number < t/2] \leq \frac{p \cdot p^{t/2} \cdot p^t}{\left(p^t-1\right)\left(p^{t-1}-1\right)} \leq \frac{16 \cdot p^{3t/2+1}}{9 \cdot p^{2t-1}} \leq \frac{2p^2}{p^{t/2}} \leq 1/2 ,$$

where $(x^t - 1) \geq 3/4 \cdot x^t$ for $x \geq 3$ and $t \geq 2$, which means that

$$\Pr[branch\ number \geq t/2] \geq 1 - \frac{2}{p^{t/2-2}}$$

for $p \gg t \geq 6$. □

Thus, $\Pr[\text{branch number} \geq t/2] \approx 1$ for $p \gg t \geq 6$.

However, in our case, the total number of sequential matrices that we can generate is limited by the t elements α_i we can choose. Hence, in total we can generate $\hat{\kappa} = (p-1)^t$ invertible matrices. Considering this special case, we get that

$$\Pr[\text{branch number} \geq z] \geq 1 - \frac{2 \cdot z \cdot p^z \cdot t^t}{(p-1)^t} \geq 1 - \frac{2^z \cdot z \cdot t^t}{(p-1)^{t-z}} ,$$

where $2(p-1) \geq p$. It follows that

$$\Pr[\text{branch number} \geq t/2] \geq 1 - \frac{t}{2} \cdot \left(\frac{2t^2}{p-1}\right)^{t/2},$$

i.e., $\Pr[\text{branch number} \geq t/2] \approx 1$ for $p \gg 2t^2$, as in our case.

8.3 Security against Algebraic Attacks

To describe our analysis, we focus on Pasta-3. Our input $\vec{x}$ consists of $s = 2t$ unknown key elements and the output $\vec{y}$ consists of t elements (after truncation). Hence, for a known nonce N and block counter i we have

$$\vec{x} = k_1 \,||\, k_2 \,||\, \cdots \,||\, k_s\,,$$
$$\vec{y} = \text{left}_t(\text{Pasta-}\pi(\vec{x}, N, i)) = y_1 \,||\, y_2 \,||\, \cdots \,||\, y_t\,.$$

Linearization

In a linearization approach, the attacker replaces all monomials of degrees greater than 1 by new variables, and finally tries to solve the resulting system of linear equations. Assuming n_v variables and a maximum degree of d, the number of possible monomials is

$$n_m = \sum_{i=1}^{d} \binom{n_v + i - 1}{i}. \tag{6.3}$$

For Pasta-3 we have $d = 12$, and hence s input words with degree 12 after one function call. Further, we obtain t equations with each call. In order to get as many equations n_e as variables n_v for our equation system, we can simply request more data, which eventually results in $n_e = n_v$ after $s/t = 2$ blocks (this has no effect on the efficiency of the linearization). Due to the complexity of solving a linear equation system in n_m variables, we target $\log_2(n_m) > 64$. Hence, $s \geq 207$ input words for a security of 128 bits. Following the same analysis, we need $s \geq 51$ for Pasta-4 and $s \geq 101$ for a Masta-like 4-round instance using only degree-2 Feistel-like S-boxes.

In this analysis, we assume that almost all monomials appear in the final representations, since our design provides strong diffusion in half of the state by using dense invertible matrices, and full diffusion after two full linear layers. In order to get more confidence in our design, we also did some practical tests and show the results in Figure 6.4. To avoid the effect of cancellations, we used prime numbers of sizes larger than 2^{16}. We observe that for the state sizes we tested, the actual number of monomials in the output word with the smallest number of monomials is always very close to the upper bound for the number of monomials given in Eq. (6.3).

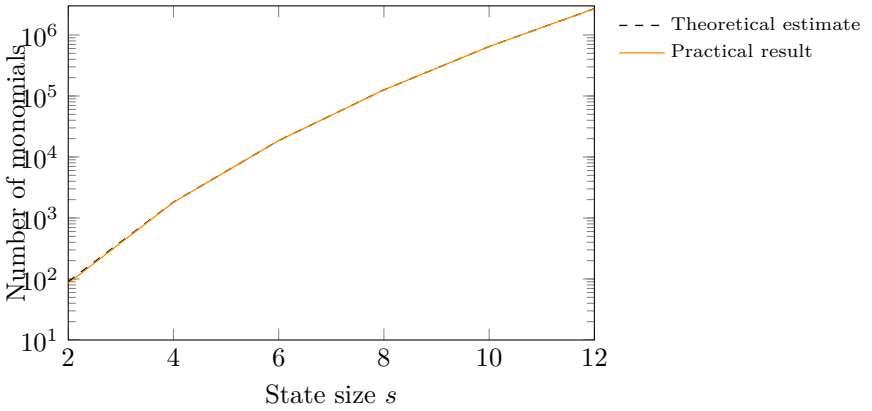


Figure 6.4: Comparison of the estimated number of monomials in each of the output words according to Eq. (6.3) and the lowest number of monomials found in a practical evaluation.

Gröbner Basis Attacks

Here we determine how large our key s has to be in order to provide security with respect to Gröbner basis attacks up to a complexity of 2^{128} function calls. As was the case above, we can simply generate sufficiently many equations by requesting at least $s/t = 2$ blocks. Hence, $n_v = n_e$, and we can estimate the complexity of solving such a system of equations by using theoretical bounds. However, these bounds assume a regular system of equations, and in practical tests we quickly observed that this is not the case for Pasta. Indeed, when building more full-round equations and hence an overdetermined system, we can force the degree of regularity to reach a minimum of 12. By reusing the estimate for the complexity of computing a Gröbner basis [BFS+05] we need $s \geq 207$. Similar results can be obtained by assuming $d = 24$ for Pasta-4.

There is also a different way to argue the number of words to use. From the linearization analysis we know that there will be roughly 2^{64} different monomials in each of the resulting equations. Due to the internals of Gröbner basis algorithms, this results in around $\left(2^{64}\right)^{\omega}$ operations being necessary to compute a basis.[1] We pessimistically (from a designer's point of view) set $\omega = 2$ and thus have $\left(2^{64}\right)^{\omega} = 2^{128}$.

Additional Strategies. The strategy presented above is only one way to attack the system using Gröbner bases. It is common to also consider approaches which introduce new variables in each state. The main idea of this technique is to reduce the degrees of the equations at the expense of more variables, which is particularly useful when trying to represent high-degree equations in a more

[1] For example, the F5 algorithm [Fau99] uses Gaussian elimination on a Macaulay matrix, whose rows indicate the equations in the system and whose columns are indexed by the monomials in these equations.

efficient way. In more detail, we may introduce a new variable after each nonlinear operation. Considering a total state size of $s = 2t$ words, we need to introduce $2s(r-1)$ new variables for an r-round construction (note that no new variables are needed after the final round, since the stream output added to a plaintext is a degree-3 combination of the previous variables). Using this many variables and equations of a degree larger than or equal to 2 results in a high solving complexity when assuming nontrivial (i.e., dense) equations (we refer to [JV17; NNY18], in which degree-2 equation systems over $\mathbb{Z}_2$ are considered). We therefore conjecture that introducing intermediate variables will only increase the complexity needed to solve the final system when compared to using full-round equations.

Other Algebraic Attacks

Many other known attacks (including e.g. higher-order differential attack [Lai94; Knu94], interpolation attack [JK97], and so on) are prevented by our random linear layers which are different in each Pasta-π evaluation. This is the same strategy as used by e.g. Rasta and Masta. We shortly discuss these attacks in this section. Furthermore, the recent attack proposed on Agrasta [LSM+21] does not apply to Pasta, since it directly exploits the χ-layer which is not present in Pasta, and it works differently over large prime fields.

Higher-Order Differential Attacks. Higher-order differential attacks [Lai94; Knu94] are essentially prevented by the fact that the attacker is only allowed to evaluate a single instance once due to the different linear layers. Moreover, the only subspaces of a finite field $\mathbb{F}_p$ with prime characteristic are $\{0\}$ and $\mathbb{F}_p$ itself, which makes higher-order differential attacks even harder (however, there have been variations of this attack vector which also work over $\mathbb{F}_p$ [BCD+20]). This also includes higher-order differential distinguishers and attacks based on higher-order differential properties (e.g., cube attacks [Vie07; DS09]).

Interpolation Attacks. In an interpolation attack [JK97], the attacker tries to build an interpolation polynomial mapping an input to the corresponding output. This polynomial can then be used to recover the secret key. However, interpolation attacks need multiple evaluations of a fixed permutation, which is not possible when considering Pasta and its varying linear layers.

Guessing Attacks. Guessing (or guess-and-determine) attacks combine the guessing of one or more variables with other attack strategies, potentially decreasing their complexities by fixing parts of the secret. However, due to the large number of state words and a minimum size of 17 bits for each of them, it is unlikely that guessing any of the state words (or even multiple of them) leads to an advantage. Indeed, using our analysis, guessing from 1 to $\lfloor 127/17 \rfloor$ words does not lead to any improvement, but even makes the attacks worse. In more detail, we would need to improve the attack itself by a factor of at least 2^{17w} when guessing w input words, which for all configurations we tested

$(1, \ldots, \lfloor 128/17 \rfloor$ guesses) is not possible with our analysis. For example, in the 17-bit case with $s = 51$ and when considering the linearization approach, the complexity is reduced by less than one bit when guessing a single variable. When assuming $s = 44$ (guessing the maximum feasible number of variables), the complexity of the attack is still around 120 bits, which is much more than the allowed $128 - 7 \cdot 17 = 9$ bits. We remark that this is the "weakest" instance from the attacker's perspective, and for all larger primes we would need an even higher performance increase for the actual attack. Further, given the density of the algebraic representation, we do not expect that the equation systems get significantly easier to solve by guessing any small number of variables.

8.4 On Using Two Different S-Boxes

To be optimized for HHE, we designed Pasta to have a small number of rounds (implying less noise consumption) and a small state size (implying fast homomorphic evaluation time). Therefore, we make use of a Feistel S-box of degree 2 and a cube S-box of degree 3. Using only Feistel S-boxes would result in a design with worse performance: A 3-round design using only Feistel S-boxes would require $t \approx 500$ plain/cipher words (based on the security analysis in Section 8), which results in significantly longer homomorphic evaluation times. A 4-round design would have the same multiplicative depth as Masta-4, leading to the same HE parameters and noise consumption as Masta-4. Therefore, this design would be faster then Masta due to the smaller size t ($t = 55$ as shown in Section 8) in one evaluation branch. However, it would not have a noise advantage. Pasta-3, on the other hand, has both a runtime and a noise advantage due to requiring fewer rounds by having the same size t as Masta-4.

The diffusion of a 2-round cipher based only on cube S-boxes would largely rely only on the single layer between the matrix multiplication. Thus the resulting diffusion is likely bad potentially allowing to separate the cipher [CDK+18]. Therefore, we chose to instantiate Pasta-3 by using the smallest depth which allows a 3-round cipher with approximately the same number of plain/cipher words t as Masta-4, which is using two Feistel S-boxes and one cube S-box.

9 Pasta Benchmarks

In this section, we benchmark a packed implementation of our Pasta design in both SEAL and HElib. We also reimplemented a packed version of Masta and Hera, using the same algorithms to generate random field elements and homomorphic matrix multiplications as in Pasta to compare these ciphers in a fair setting. Similar as in Section 5, we also benchmark the ciphers in a real HHE use case.

9.1 Comparing Pasta to $\mathbb{Z}_2$ Ciphers

We first compare Pasta, Masta, and Hera to the $\mathbb{Z}_2$ benchmarks from Section 5. Therefore, we instantiate these ciphers with a 17-bit prime and benchmark their performance for the small use case from Section 5.[1] The resulting benchmarks can be seen in Table 6.9 where we depict both runtime and remaining noise budget after each step of the HHE use case for SEAL. For benchmarks in HElib we refer to Section B.2.

Table 6.9: Runtime and noise budget of the small HHE use case in the SEAL library (security level $\lambda = 128$ bit).

Cipher	N	Enc. Key		Decomp.		Small Use Case	
		runtime s	noise bit	runtime s	noise bit	runtime s	noise bit
$p = 65537$ (17 bit):							
Pasta-3	16384	**0.017**	364	**9.28**	95	**0.197**	51
Pasta-4	32768	0.059	800	21.0	451	1.11	406
Masta-4	32768	0.058	800	54.2	460	1.11	415
Masta-5	32768	0.057	800	39.2	386	1.13	341
Hera	32768	0.051	800	16.6	333	1.12	287

Discussion

In the following, we compare the runtime and noise consumption of all $\mathbb{Z}_2$ and $\mathbb{F}_p$ (with $p = 65537$) ciphers, namely in Figure 6.5 for homomorphically decrypting one block in SEAL ($\mathbb{F}_p$ values from Section 9.2), and in Figure 6.6 for the HHE use case (including HHE decompression) in SEAL. For HElib benchmarks we refer to Section B.2.

Our figures indicate that Pasta is always the fastest cipher – mainly Pasta-4 due to the small number of encrypted words. However, Pasta-3 is faster when evaluating the whole HHE use case in SEAL due to the small multiplicative depth requiring smaller HE parameters for security. Comparing Pasta to the $\mathbb{Z}_2$ ciphers, one can observe that homomorphically decrypting one block requires less noise budget for the $\mathbb{Z}_2$ ciphers. However, Pasta has (besides the runtime advantage) a noise advantage over the $\mathbb{Z}_2$ ciphers when considering the HHE use case due to the significantly larger multiplicative depth of the binary circuits for integer arithmetic. Concretely, decompression and use case evaluation is 33× faster in SEAL using Pasta-3 and 82× faster in HElib using Pasta-4 compared to Agrasta. Using TFHE in gate-bootstrapping mode for $\mathbb{Z}_2$ ciphers instead of e.g. SEAL does not help the $\mathbb{Z}_2$ ciphers either, since Pasta-3 in SEAL is 47× faster than using Kreyvium in TFHE for the small HHE use case. Increasing

[1] For sake of simplicity we do not consider plaintext overflows in this example, so no mod p operation is part of the binary circuits and no mod 2^{16} is part of the $\mathbb{F}_p$ benchmarks.

the bitsize of the encrypted integers or chaining multiple matrix multiplications would further demonstrate the advantage of Pasta over $\mathbb{Z}_2$ ciphers, since the drastic increase in the multiplicative depth of the use case would make using the $\mathbb{Z}_2$ ciphers infeasible.

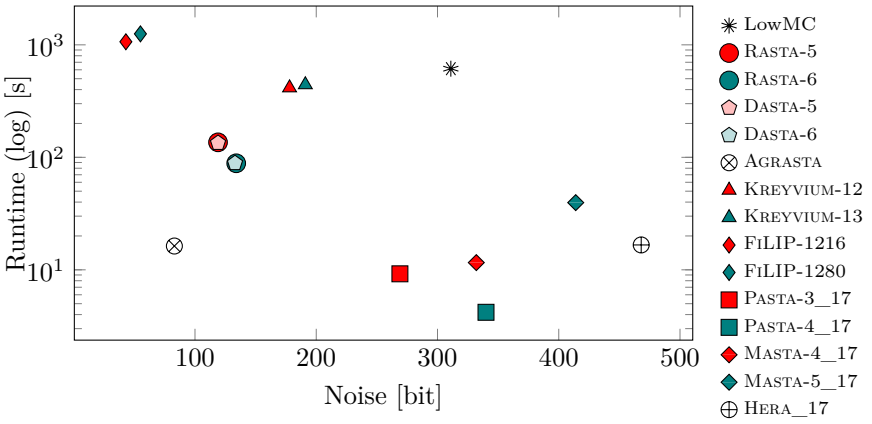


Figure 6.5: Runtime and noise comparison of $\mathbb{Z}_2$ ciphers for homomorphically decrypting 1 Block in SEAL (security level $\lambda = 128$ bit).

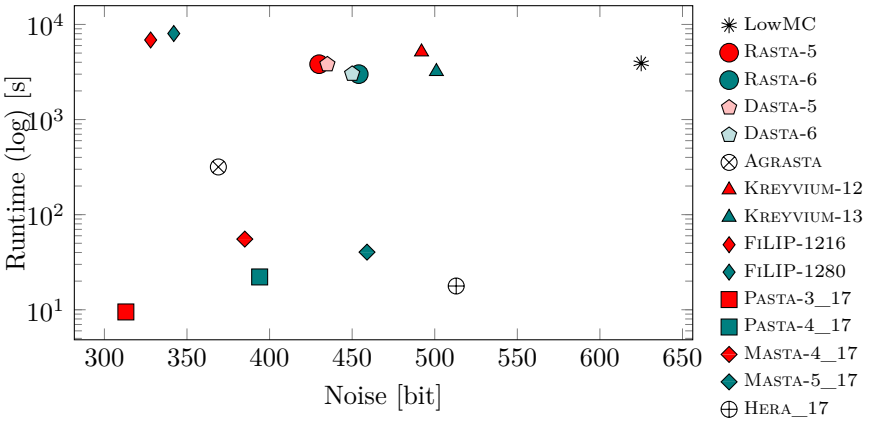


Figure 6.6: Runtime and noise comparison for the small HHE use case in SEAL (security level $\lambda = 128$ bit).

9.2 Pasta versus Masta and Hera

Since all $\mathbb{F}_p$ ciphers outperform the $\mathbb{Z}_2$ ciphers for HHE, we continue with comparing these ciphers. Similar to the $\mathbb{Z}_2$ benchmarks, we also compare Pasta, Masta, and Hera in a real HHE use case. However, to further demonstrate the

advantage of the $\mathbb{F}_p$ ciphers in HHE, we benchmark a more extensive use case with a significantly higher multiplicative depth. We reuse the same use case as in Section 4, i.e., three affine layers interleaved with squarings on a vector $\vec{x} \in \mathbb{F}_p^{200}$. We benchmark the use case for 3 different primes p.

SEAL Benchmarks

In this section we discuss the benchmarks for the $\mathbb{F}_p$ ciphers in SEAL, for benchmarks in HElib we refer to Section B.2. Furthermore, we provide CPU cycle counts for plain encryption with Pasta, Masta, and Hera in Section B.3. In Table 6.10 we present the benchmarks for the packed implementation of Pasta, Masta, and Hera in the SEAL library. We give timings for homomorphically decrypting one block and additionally timings for the bigger HHE use case. We parameterize SEAL to provide 128 bits of security and use the smallest N allowing enough noise budget for correct evaluation.

Table 6.10: $\mathbb{F}_p$ benchmarks for the SEAL library (security level $\lambda = 128$ bit).

Cipher	1 Block			Bigger HHE use case			
	N	Enc. Key s	Decomp. s	N	Enc. Key s	Decomp. s	Use Case s
$p = 65537$ (17 bit):							
Pasta-3	16384	0.016	9.22	32768	0.056	**86.2**	43.9
Pasta-4	16384	**0.016**	**4.19**	32768	0.057	147.8	**43.8**
Masta-4	16384	0.016	11.6	32768	0.058	108.7	43.9
Masta-5	32768	0.062	39.6	32768	0.056	157.0	43.9
Hera	32768	0.052	16.6	32768	**0.051**	215.4	43.9
$p = 8088322049$ (33 bit):							
Pasta-3	32768	0.057	43.1	32768	**0.055**	**86.3**	**43.9**
Pasta-4	32768	0.057	21.2	65536	0.216	833.4	220.8
Masta-4	32768	0.058	54.4	65536	0.215	568.5	221.3
Masta-5	32768	0.055	39.3	65536	0.215	852.6	220.7
Hera	32768	**0.051**	**16.6**	65536	0.196	1227.7	220.7
$p = 1096486890805657601$ (60 bit):							
Pasta-3	32768	**0.055**	**58.3**	65536	0.212	**448.6**	**220.8**
Pasta-4	65536	0.220	119.2	65536	0.212	833.6	221.0
Masta-4	65536	0.220	284.3	65536	0.212	571.9	223.1
Masta-5	65536	0.219	213.3	65536	0.212	853.3	220.9
Hera	65536	0.200	94.6	65536[a]	**0.193**	1228.3	221.0

[a] Noise budget did not suffice and bigger parameters are not available in SEAL. Thus, bootstrapping is required.

Discussion

In the following figures we compare the runtime and noise consumption of Pasta, Masta, and Hera for 3 different prime fields $\mathbb{F}_p$, in Figure 6.7 for

homomorphically decrypting one block in SEAL, and in Figure 6.8 for the HHE use case (including HHE decompression) in SEAL. For HElib benchmarks we refer to Section B.2.

The figures show the advantage of Pasta compared to its competitors. In all figures, Pasta-3 has a smaller runtime and noise consumption then Masta, especially when the smaller multiplicative depth allows for smaller HE parameters (compare, e.g., 33-bit prime fields in Figure 6.8, where Pasta-3 is 6× faster than Masta-4). Pasta-3 is only outperformed by Pasta-4 and Hera for a small number of encrypted words (e.g., only encrypting one block as for the 33-bit prime for SEAL where Hera is slightly faster then Pasta-4, or the small HHE use case from Section 9.1 in HElib where Pasta-4 is 2.7× faster then Masta-4) if the overall multiplicative depth allows Pasta-4 or Hera to use the same HE parameters as Pasta-3. Hence, we propose using Pasta-4 for HHE use cases with a small number of encrypted words, and Pasta-3 everywhere else.

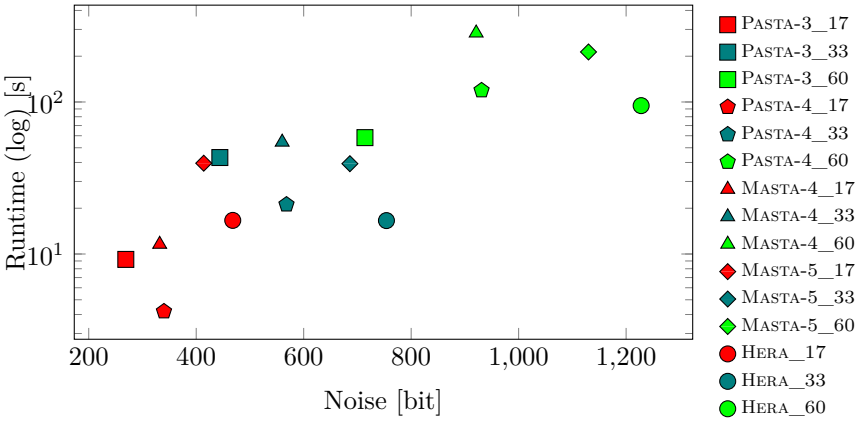


Figure 6.7: Runtime and noise comparison of $\mathbb{F}_p$ ciphers for homomorphically decrypting 1 Block in SEAL (security level $\lambda = 128$ bit).

9.3 Pasta in Different Use Cases

In recent years, many symmetric primitives defined over $\mathbb{F}_p^t$, such as GMiMC [AGP+19], HadesMiMC [GLR+20], Poseidon [GKR+21], Rescue [AAB+20], Ciminion [DGG+21], Griffin [GHR+22], Reinforced Concrete [GKL+22], and Hydra [GØS+22], have been proposed in the literature. However, contrary to Pasta, these primitives were not designed for HHE, but for MPC and zk-SNARK/STARK use cases, which is why they were optimized for different metrics. While having a low multiplicative depth is the most important design criterion for use cases involving homomorphic encryption, the other use cases usually just require a small total number of multiplications. Therefore, these afromentioned symmetric primitives have a significant larger number of rounds and, consequently, a large multiplicative depth which makes them infeasible

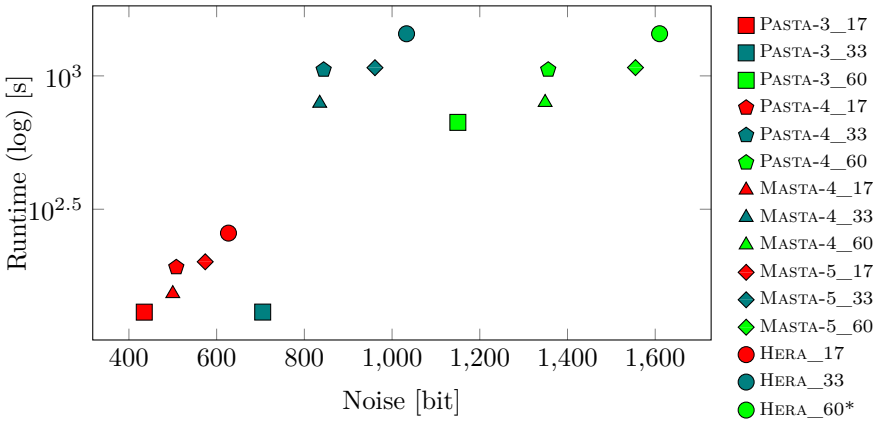


Figure 6.8: Runtime and noise comparison for the bigger HHE use case in SEAL (security level $\lambda = 128$ bit). Ciphers marked with a * did not have enough noise budget.

for HE use cases. Pasta on the other hand has a very small depth, but the significantly larger statesize results in a large total number of multiplications. In HE use cases many of these multiplications are performed in parallel using packing, but this large number of multiplications makes Pasta worse for MPC and zk-SNARK/STARK applications. In some MPC scenarios (e.g., scenarios with a very high-delay, low bandwidth WAN connection between the parties), the low multiplicative depth of Pasta may, however, give it an advantage over the other constructions.

Conclusion

In this paper, we investigated hybrid homomorphic encryption, a technique to combat ciphertext expansion in homomorphic encryption applications at the cost of more expensive computations in the encrypted domain. Since HHE was first mentioned in [NLV11], many symmetric ciphers for HHE have been proposed in the literature. However, the effects of applying HHE to any use case were not really understood so far. In our work, we tackled this issue in several ways: First, we for the first time investigate the high-level impact on the server and client when applying HHE to a practical use case in Section 4. Secondly, we implement a framework which for the first time compares many different symmetric ciphers when used with HHE in three popular HE libraries. Finally, to improve the performance of HHE, we propose a new symmetric cipher, dubbed Pasta, which outperforms the state-of-the-art for integer use cases over $\mathbb{F}_p$.

The main take-aways of this paper are the following: Our investigations show, that HHE achieves the best results when the clients are embedded devices with limited computational power and bandwidth. Furthermore, many state-of-the-art

ciphers are not well suited for many HHE applications due to being defined over $\mathbb{Z}_2$. Finally, while HHE is very beneficial for clients, the actual computation in the encrypted domain suffers. This is due to first having to decrypt the symmetric ciphertexts under homomorphic encryption before computing the actual use case. While this extra work naturally contributes to the computation runtime, it also contributes to the multiplicative depth of the whole HE computation. Since an efficient bootstrapping operation is still missing from most state-of-the-art HE libraries (such as the ones considered in this paper), this additional multiplicative depth significantly contributes to the whole computaiton runtime. As a consequence, we show that only evaluating a cipher under HE is not enough to estimate its performance in HHE, one has to consider the whole HHE use case instead.

Acknowledgments

This work was supported by EU's Horizon 2020 project Safe-DEED under grant agreement n°825225, and 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. Lorenzo Grassi is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA.

A HE Schemes and Libraries (cont.)

BGV [BGV12] in HElib [HS20]. The BGV scheme, and its implementation in HElib, allows plaintexts in $\mathbb{Z}_{p^r}$ and offers more flexibility for choosing HE parameters than SEAL. It allows arbitrary cyclotomic reduction polynomials and it is possible to find parameters which allow packing for $\mathbb{Z}_2$ plaintexts. However, this flexibility comes with the drawback that parameterizing for HElib is more difficult than finding parameters in SEAL, and the limited parameter sets in SEAL allow for more optimized implementations. In this paper we use the HElib version 2.1.0. Similar to BFV in SEAL, additions are considered free in BGV, and the multiplicative depth of the circuit is the most relevant performance metric.

TFHE [CGG+20] in TFHE [CGG+gu]. The TFHE library, more concretely the gate-bootstrapping version of the original TFHE library which we use in this paper, is vastly different from SEAL and HElib. It only allows the encryption of boolean values (i.e., plaintexts are in $\mathbb{Z}_2$), but it is optimized for fast gate bootstrapping. This basically means that after the evaluation of a homomorphic gate the noise in the ciphertext is reset. As a consequence, contrary to most other modern homomorphic encryption schemes, the multiplicative depth of a circuit is no relevant metric there. However, each homomorphically evaluated gate requires the same computational effort, thus additions are not considered to be free as in the BFV or BGV cryptosystems. The most relevant metric for TFHE in gate-bootstrapping mode is, therefore, the total number of gates. Furthermore, SIMD style packing is not supported in TFHE. Since TFHE only allows to encrypt boolean values, we do not implement and consider $\mathbb{F}_p$ ciphers in this library.

B Additional Benchmarks

In this section we give the benchmarks of all the ciphers in the original TFHE library (Section B.1), and in HElib (Section B.2). Finally, we compare the plain performance of Pasta and Masta in Section B.3.

B.1 TFHE Benchmarks of $\mathbb{Z}_2$ Ciphers

Since the noise in the ciphertexts is reset after every homomorphic operation due to gate-bootstrapping in TFHE, we do not have to choose any parameters for the benchmarks (except the security level, which we set to 128 bits). In Table 6.11 we present the benchmarks for the TFHE library for homomorphically decrypting only one block, and for the small HHE use case from Section 5. We give timings for homomorphically encrypting the symmetric key, homomorphically decrypting one block, and for the small HHE use case. Since TFHE does not support packing all implementations are bitsliced (i.e., one HE ciphertext per bit).

Table 6.11: Benchmarks for the TFHE library (security level $\lambda = 128$ bit).

Cipher	Enc. Key	1 Block	Small HHE use case	
		Decomp.	Decomp.	Use Case
	s	s	s	s
LowMC	0.003	6 120.5	6 310.6	175.6
Rasta-5	0.013	5 728.8	5 807.8	164.0
Rasta-6	0.009	3 275.0	3 293.6	162.2
Dasta-5	0.013	5 642.6	5 664.7	165.2
Dasta-6	0.009	3 272.7	3 293.0	**162.0**
Agrasta	0.003	407.1	408.4	164.4
Kreyvium	**0.003**	**284.1**	**290.4**	162.6
Kreyvium-12	0.003	284.1	559.7	162.6
Kreyvium-13	0.003	310.1	290.4	162.6
FiLIP-1216	0.442	1 504.6	1 886.9	164.3
FiLIP-1280	0.107	1 594.5	1 981.8	162.8

Discussion. In Figure 6.9 we compare the runtime of homomorphically decrypting one block and the whole HHE use case (including homomorphic decryption) of the $\mathbb{Z}_2$ ciphers in TFHE. In the gate-bootstrapping version of TFHE the main performance metric is the total gate count, which is why Kreyvium is the fastest choice. Since the TFHE library only allows plaintexts in $\mathbb{Z}_2$ we do not implement and compare $\mathbb{F}_p$ ciphers in TFHE.

B.2 HElib Benchmarks

In this section, we give all the benchmarks in the HElib. First, we benchmark the $\mathbb{Z}_2$ ciphers, before we compare them to the $\mathbb{F}_p$ ciphers. Finally, we benchmark Pasta, Masta, and Hera in a more extensive use case.

HElib Benchmarks of $\mathbb{Z}_2$ Ciphers

In HElib, the security and available noise budget mainly depend on the choice of the cyclotomic reduction polynomial, as well as the size of the ciphertext modulus. A bigger modulus provides a bigger noise budget at the cost of less security. A bigger cyclotomic polynomial provides more security, but is bad for performance. In our benchmarks, we use the tool provided by HElib to find suitable parameters given a target security level of 128 bits and a target noise budget which we gathered from the experiments. The resulting parameter sets provide $\lambda' \approx 128$ bits of security with the majority of sets providing slightly less.

In Table 6.12 we present the benchmarks for the HElib library, for homomorphically decrypting only one block, and for the small HHE use case from Section 5. For both benchmarks we give timings alongside the chosen m-th cyclotomic reduction polynomial (chosen by HElib) and the estimated security λ' (estimated by HElib). For the HHE use case we additionally give the runtime

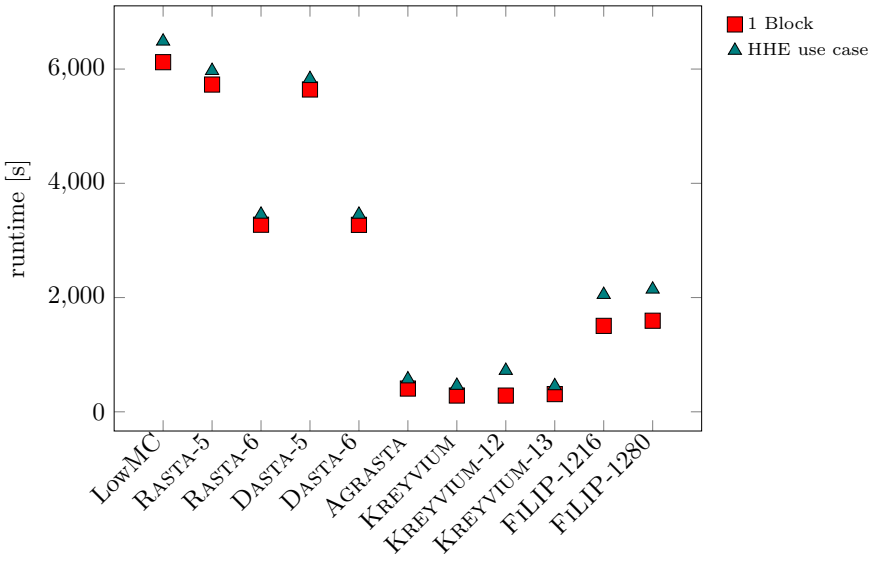


Figure 6.9: Runtime comparison of homomorphically decrypting one block and the small HHE use case (including HHE decompression) of $\mathbb{Z}_2$ ciphers in TFHE (security level $\lambda = 128$ bit).

for the affine transformation use case. To compare the benchmarks to SEAL and TFHE, all implementations are bitsliced (i.e., one HE ciphertext per bit).

Remark 8. *HElib supports packing for $\mathbb{Z}_2$ plaintexts. Even though a packed implementation of the symmetric ciphers will increase their overall performance, it complicates the evaluation of an integer matrix-vector multiplication based on binary circuits. Therefore, packed implementations do not fix the main issue of $\mathbb{Z}_2$ ciphers for* HHE*, which is supporting integer arithmetic over $\mathbb{F}_p$. For this reason, we do not provide explicit packed benchmarks for the ciphers in the paper.*

Comparing Pasta to $\mathbb{Z}_2$ Ciphers in HElib

The benchmarks for HElib can be seen in Table 6.13 where we depict both runtime and remaining noise budget after each step of the HHE use case from Section 5. In the following, we compare the runtime and noise consumption of all $\mathbb{Z}_2$ and $\mathbb{F}_p$ (with $p = 65537$) ciphers, namely in Figure 6.10 for homomorphically decrypting one block in HElib ($\mathbb{F}_p$ values from Section B.2), and in Figure 6.11 for the HHE use case (including HHE decompression) in HElib. Since Masta and Pasta require to use the m-th cyclotomic reduction polynomial $(X^{m/2} + 1)$, where m is a power-of-two, we chose parameters differently compared to Section B.2: We parameterize q to provide enough noise budget to evaluate the benchmark and chose the m to be the smallest power-of-two such that the parameters provide ≥ 128 bits security. Thereby, for a fixed m, a smaller q provides both, larger

Table 6.12: Benchmarks of the $\mathbb{Z}_2$ ciphers in the HElib library.

Cipher	1 Block				Small HHE use case				
	m	λ' bit	Enc. Key s	Decomp. s	m	λ' bit	Enc. Key s	Decomp. s	Use Case s
LowMC	23377	110	9.22	1 132.4	43691	108	27.5	3 708.8	1 618.8
Rasta-5	11441	111	11.7	284.2	31609	118	57.7	1 666.9	922.4
Rasta-6	11441	111	7.79	207.7	31609	108	41.8	1 401.0	1 037.2
Dasta-5	11441	111	11.8	276.7	31609	118	57.9	1 608.4	922.0
Dasta-6	11441	111	7.87	201.7	31609	108	41.6	1 357.3	1 042.6
Agrasta	10261	117	**2.38**	**38.3**	32767	108	**13.7**	**276.9**	853.6
Kreyvium	14351	108	3.97	497.0	43691	144	22.0	3 392.6	1 431.9
Kreyvium-12	14351	108	4.06	498.3	43691	147	22.0	6 657.1	1 392.6
Kreyvium-13	15709	113	4.38	577.1	43691	144	21.7	3 407.3	1 420.9
FiLIP-1216	5461	113	131.4	1 357.5	23311	108	1 010.0	17 919.7	**566.6**
FiLIP-1280	8435	119	47.3	2 197.4	24929	105	337.2	27 613.9	745.2

security and faster performance. Consequently, greater λ' in Table 6.13 also lead to faster runtimes compared to instantiating the same benchmark with exactly 128 bits of security.

Table 6.13: Runtime and noise budget of the small HHE use case in the HElib library.

Cipher	m	λ' bit	Enc. Key		Decomp.		Small Use Case	
			runtime s	noise bit	runtime s	noise bit	runtime s	noise bit
$p = 65537$ (17 bit):								
Pasta-3	65536	173	0.054	410	26.0	74	0.754	23
Pasta-4	65536	142	**0.054**	475	**13.0**	56	**0.737**	6
Masta-4	65536	133	0.054	502	36.7	86	0.740	36
Masta-5	131072	254	0.116	566	55.4	56	1.71	5
Hera	131072	234	0.124	632	20.9	69	1.74	19

HElib Benchmarks of $\mathbb{F}_p$ Ciphers

In Table 6.14 we present the benchmarks for the packed implementation of Pasta, Masta, and Hera in the HElib library. We give timings for homomorphically decrypting one block and additionally timings for the bigger HHE use case (Section 9.2). We chose parameters in the same fashion as in Section B.2, i.e., choosing q to provide enough noise budget to evaluate the benchmark, and choose the m-th cyclotomic reduction polynomial, with m being a power of two, such that the HE scheme provides ≥ 128 bits security.

Remark 9. *In Table 6.14, some benchmarks were run with $\lambda < 128$ bits security. The reason for that is that $m = 262144$ unfortunately lead to infeasible runtimes. Consequently, $m = 131072$ seems to be an upper limit for feasible runtimes in HElib, and use cases requiring larger amounts of noise than can be provided*

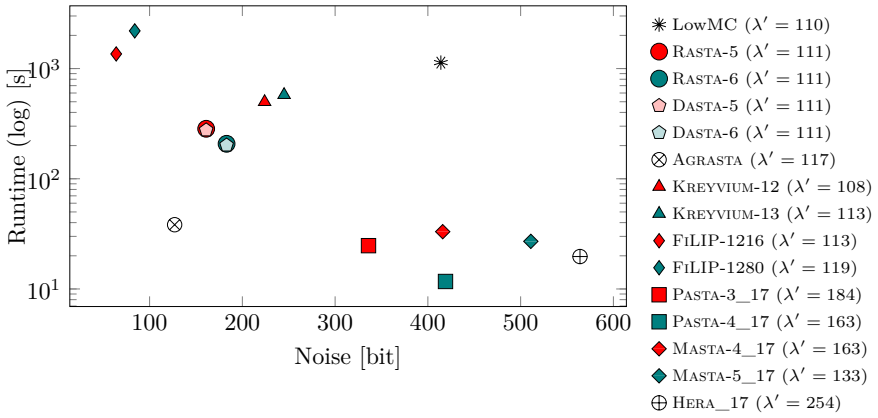


Figure 6.10: Runtime and noise comparison of $\mathbb{Z}_2$ ciphers for homomorphically decrypting 1 Block in HElib (HE security level λ').

by $m = 131072$ *and* $\lambda \geq 128$ *would inevitably require an efficient bootstrapping operation.*

In the following figures we compare the runtime and noise consumption of the ciphers for 3 different prime fields $\mathbb{F}_p$, in Figure 6.12 for homomorphically decrypting one block in HElib, and in Figure 6.13 for the HHE use case (including HHE decompression) in HElib.

B.3 Plain Benchmarks of Pasta, Masta and Hera

In Table 6.15 we compare the number of CPU cycles of the encryption circuit of Pasta to the encryption circuit of Masta and Hera. Since these ciphers generate random matrices and/or round constants independent of the secret key, which can be precomputed before encryption, we additionally give CPU cycles for generating these affine layers and keys schedules and the encryption circuit with precomputed randomness. Table 6.15 shows that Hera, with its small block size and fixed matrices which can be evaluated purely by additions, is the fastest cipher in plain. However, this advantage comes at the cost of higher number of rounds, which worsenes homomorphic performance. Comparing Pasta to Masta, one can observe that Pasta-4, due to its small state size, requires the smallest number of cycles to encrypt one block. Pasta-3, on the other hand, due to sampling sequential matrices instead of polynomials $m \in \mathbb{Z}_p[X]/(X^t - \alpha)$ (as in Masta) and requiring twice as many matrices per round, is the slowest cipher to encrypt one block in plain. However, the difference to Masta-4 is only a factor of 3, which in practice corresponds to latencies in the order of milliseconds.

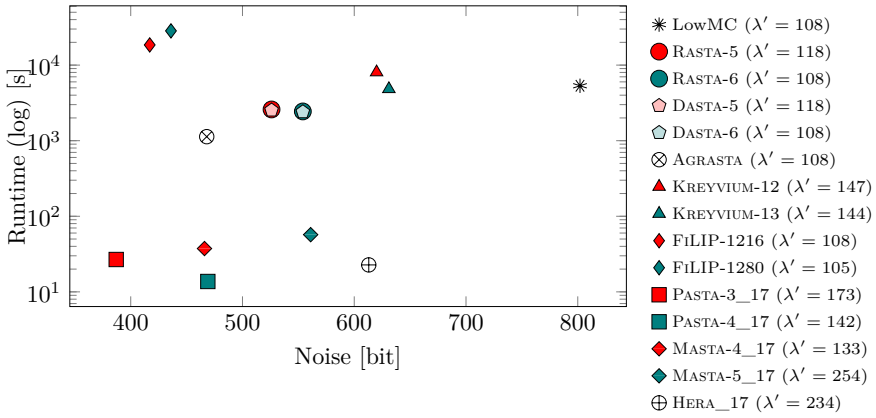


Figure 6.11: Runtime and noise comparison for the small HHE use case in HElib (HE security level λ').

C Packed vs. Word-Sliced Implementation of Pasta

In Section 6, we describe efficient SIMD algorithms to evaluate Pasta on a packed HE ciphertext. In this section, we want to compare them to a word-sliced implementation where one would encrypt only one field element $\in \mathbb{F}_p$ into one HE ciphertext. A word-sliced implementation has several disadvantages. First, the homomorphic evaluation time of Pasta would be much slower. In a packed implementation, the S-boxes can be evaluated with $\mathcal{O}(1)$ homomorphic operations, and with $\mathcal{O}(t)$ HE operations in a word-sliced implementation. The word-sliced affine layer requires $\mathcal{O}(t^2)$ HE operations compared to $\mathcal{O}(t)$ operations when using packing. Secondly, the initial setup in the HHE use case requires the transmission of the HE encrypted symmetric key. In a packed implementation, this is always only one HE ciphertext. However, in a word-sliced implementation, one has to transmit $2 \cdot t$ HE ciphertexts, drastically increasing the communication cost of this setup phase. Finally, if the HHE use case leverages packing, one has to reconstruct a packed ciphertext from its word-sliced state using many rotations on the server.

However, word-sliced implementations have an advantage as well. They do not require homomorphic rotations (and, therefore, no Galois keys) and one can access each word of the state individually. This is why one can implement the S-boxes from Section 6.4 without requiring masking multiplications. As a consequence, word-sliced implementations have less noise consumption. Splitting the state in our Pasta design is also beneficial for word-sliced implementations, since it reduces the number of homomorphic multiplications from $(2 \cdot t)^2$ to $2 \cdot t^2$ per affine layer, reducing the runtime.

Table 6.14: $\mathbb{F}_p$ benchmarks for the HElib library.

Cipher	1 Block				Bigger HHE use case				
	m	λ' bit	Enc. Key s	Decomp. s	m	λ' bit	Enc. Key s	Decomp. s	Use Case s
$p = 65537$ (17 bit):									
Pasta-3	65536	184	0.052	24.7	65536	128	**0.064**	**57.6**	**19.9**
Pasta-4	65536	163	**0.052**	**11.7**	131072	229	0.124	210.8	38.6
Masta-4	65536	163	0.062	33.1	131072	229	0.131	157.3	45.4
Masta-5	65536	133	0.064	27.1	131072	199	0.135	252.8	48.4
Hera	131072	254	0.116	19.7	131072	189	0.121	315.1	48.0
$p = 8088322049$ (33 bit):									
Pasta-3	65536	128	**0.057**	**28.7**	131072	162	**0.166**	**187.7**	60.5
Pasta-4	131072	204	0.166	35.3	131072	144	0.190	320.5	**57.8**
Masta-4	131072	196	0.165	101.3	131072	144	0.166	256.2	69.5
Masta-5	131072	166	0.168	82.4	131072[a]	117	0.242	427.8	80.0
Hera	131072	150	0.179	29.6	131072[a]	110	0.239	526.1	82.4
$p = 1096486890805657601$ (60 bit):									
Pasta-3	131072	162	0.185	94.1	131072[a]	97	**0.285**	**268.8**	84.4
Pasta-4	131072	129	**0.183**	50.5	131072[a]	83	0.310	486.7	**84.2**
Masta-4	131072	129	0.208	144.7	131072[a]	83	0.289	387.4	101.6
Masta-5	131072[a]	99	0.233	122.1	131072[a]	70	0.300	635.5	111.9
Hera	131072[a]	89	0.249	**44.2**	131072[a]	60	0.318	816.3	124.1

[a] Further increasing m for security resulted in infeasibly long runtimes.

C.1 About a Word-Sliced Hera Implementation

Since Hera has a small statesize (16) and a larger round number (5), it might be beneficial to have a word-sliced implementation instead of a packed one. Indeed, since the linear layers can purely be implemented by additions, the multiplicative depth gets reduced from 10 ct-ct multipliations and 7 pt-ct multiplications to a depth of 10 and 1 multiplications respectively. Comparing a word-sliced implementation to a packed Pasta-3 implementation, one can observe that Pasta-3 still is preferable. On one hand, Pasta-3 has a smaller depth (4 ct-ct and 6 pt-ct Multiplications) implying less noise consumption and smaller HE parameters. On the other hand, comparing the number of HE operations involved (96 pt-ct and 160 ct-ct multiplications for Hera and 514 pt-ct multiplications, 4 ct-ct multiplications and 98 rotations for Pasta-3) one can see that Pasta-3 requires significantly less of the more expensive rotations and ct-ct multiplications at the cost of more pt-ct multiplications. Thus, if there is a small advantage of Hera when evaluating one block with 16 output words (Pasta-3 has 128), then this advantage is already gone when evaluating two or more blocks. Consequently, we conjecture that Pasta-3 is still more beneficial in most use cases than a wordsliced Hera implementation.

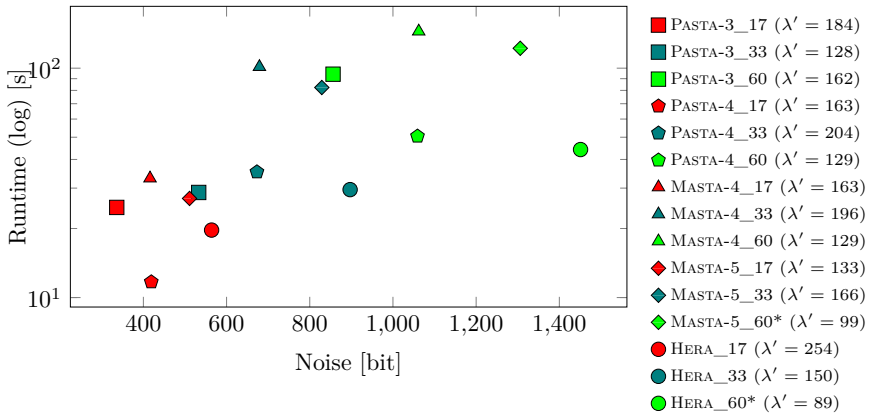


Figure 6.12: Runtime and noise comparison of $\mathbb{F}_p$ ciphers for homomorphically decrypting 1 Block in HElib (HE security level λ'). Ciphers marked with a * were evaluated with less than 128 bit HE security.

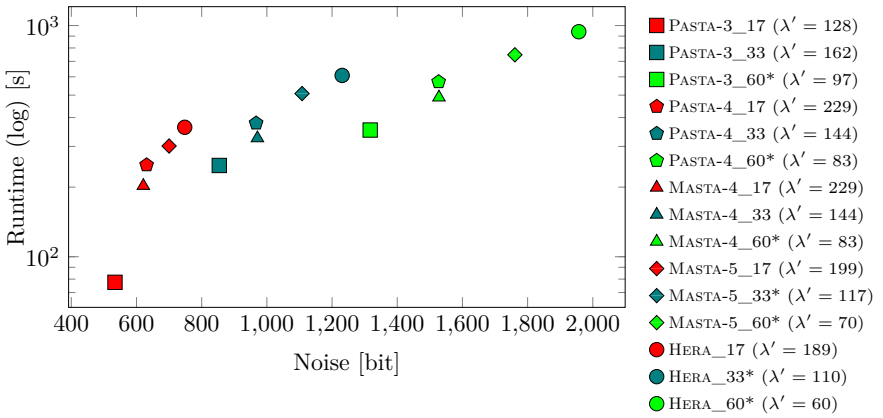


Figure 6.13: Runtime and noise comparison for the bigger HHE use case in HElib (HE security level λ'). Ciphers marked with a * were evaluated with less than 128 bit HE security.

Table 6.15: Cycles for encrypting one block in plain, averaged over 1000 executions.

Cipher	Total	Affine Generation	Encrypting
$p = 65537$ (17 bit):			
Pasta-3	17 041 380	9 196 314	7 845 066
Pasta-4	1 363 339	825 067	538 272
Masta-4	6 535 937	2 164 002	4 371 935
Masta-5	2 105 628	752 374	1 353 254
Hera	**60 391**	30 615	29 776
$p = 8088322049$ (33 bit):			
Pasta-3	22 429 444	11 637 800	10 791 644
Pasta-4	1 750 420	973 205	777 215
Masta-4	8 427 384	1 975 522	6 451 862
Masta-5	2 690 636	674 201	2 016 435
Hera	**54 567**	17 350	37 217
$p = 1096486890805657601$ (60 bit):			
Pasta-3	31 053 515	16 067 138	14 986 377
Pasta-4	2 458 680	1 315 770	1 142 910
Masta-4	11 405 862	1 968 100	9 437 762
Masta-5	3 542 410	669 220	2 873 190
Hera	**61 360**	16 873	44 487

References

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols." In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 1–45.

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. "Feistel Structures for MPC, and More." In: *ESORICS*. Vol. 11736. LNCS. Springer, 2019, pp. 151–171.

[AMT22] Tomer Ashur, Mohammad Mahzoun, and Dilara Toprakhisar. "Chaghri - an FHE-friendly Block Cipher." In: *IACR Cryptol. ePrint Arch.* (2022). accepted at ACM CCS 2022, p. 592.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. "Ciphers for MPC and FHE." In: *EUROCRYPT*. Vol. 9056. LNCS. Springer, 2015, pp. 430–454.

[BBH+22] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. "Privately Connecting Mobility to Infectious Diseases via Applied Cryptography." In: *Proc. Priv. Enhancing Technol.* 2022.4 (2022), pp. 768–788.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials." In: *EUROCRYPT*. Vol. 1592. LNCS. Springer, 1999, pp. 12–23.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. "Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems." In: *CRYPTO*. Vol. 12172. LNCS. Springer, 2020, pp. 299–328.

[BFS+05] Magali Bardet, Jean-Charles Faugere, Bruno Salvy, and Bo-Yin Yang. "Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems." In: *Proc. of MEGA*. Vol. 5. 2005, pp. 2–2.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping." In: *ITCS*. ACM, 2012, pp. 309–325.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 868–886.

[BS90] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems." In: *CRYPTO*. Vol. 537. LNCE. Springer, 1990, pp. 2–21.

[BW12] Andrey Bogdanov and Meiqin Wang. "Zero Correlation Linear Cryptanalysis with Reduced Data Complexity." In: *FSE*. Vol. 7549. LNCS. Springer, 2012, pp. 29–48.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. "Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression." In: *FSE*. Vol. 9783. LNCS. Springer, 2016, pp. 313–333.

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. "Batch Fully Homomorphic Encryption over the Integers." In: *EUROCRYPT*. Vol. 7881. LNCS. Springer, 2013, pp. 315–335.

[CDK+18] Benoît Cogliati, Yevgeniy Dodis, Jonathan Katz, Jooyoung Lee, John P. Steinberger, Aishwarya Thiruvengadam, and Zhe Zhang. "Provable Security of (Tweakable) Block Ciphers Based on Substitution-Permutation Networks." In: *CRYPTO*. Vol. 10991. LNCS. Springer, 2018, pp. 722–753.

[CDK+21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. "Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts." In: *ACNS*. Vol. 12726. LNCS. Springer, 2021, pp. 460–479.

[CGG+20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "TFHE: Fast Fully Homomorphic Encryption Over the Torus." In: *J. Cryptol.* 33.1 (2020), pp. 34–91.

[CGG+gu] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library*. https://tfhe.github.io/tfhe/. August 2016.

[CGL+20] Edward J. Chou, Arun Gururajan, Kim Laine, Nitin Kumar Goel, Anna Bertiger, and Jack W. Stokes. "Privacy-Preserving Phishing Web Page Classification Via Fully Homomorphic Encryption." In: *ICASSP*. IEEE, 2020, pp. 2792–2796.

[CHK+21] Jihoon Cho, Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. "Transciphering Framework for Approximate Homomorphic Encryption." In: *ASIACRYPT*. Vol. 13092. LNCS. Springer, 2021, pp. 640–669.

[CHM+22] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. "Towards Globally Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher." In: *IACR Cryptol. ePrint Arch.* (2022), p. 180.

[CIR22] Carlos Cid, John Petter Indrøy, and Håvard Raddum. "FASTA - A Stream Cipher for Fast FHE Evaluation." In: *CT-RSA*. Vol. 13161. LNCS. Springer, 2022, pp. 451–483.

[CJL+20] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. "CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE." In: *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. Vol. 15. 2020.

[CJP21] Ilaria Chillotti, Marc Joye, and Pascal Paillier. "Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks." In: *CSCML*. Vol. 12716. LNCS. Springer, 2021, pp. 1–19.

[CKK+17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. "Homomorphic Encryption for Arithmetic of Approximate Numbers." In: *ASIACRYPT*. Vol. 10624. LNCS. Springer, 2017, pp. 409–437.

[CLT14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. "Scale-Invariant Fully Homomorphic Encryption over the Integers." In: *Public Key Cryptography*. Vol. 8383. LNCS. Springer, 2014, pp. 311–328.

[CMG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. "Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication." In: *CCS*. ACM, 2021, pp. 1135–1150.

[Dae95] Joan Daemen. *Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis*. http://jda.noekeon.org/. K.U.Leuven, 1995.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. "Rasta: A Cipher with Low ANDdepth and Few ANDs per Bit." In: *CRYPTO*. Vol. 10991. LNCS. Springer, 2018, pp. 662–692.

[DGG+21] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. "Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields." In: *EUROCRYPT*. Vol. 12697. LNCS. Springer, 2021, pp. 3–34.

[DR00] Joan Daemen and Vincent Rijmen. "Rijndael for AES." In: *AES Candidate Conference*. National Institute of Standards and Technology, 2000, pp. 343–348.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[DS09] Itai Dinur and Adi Shamir. "Cube Attacks on Tweakable Black Box Polynomials." In: *EUROCRYPT*. Vol. 5479. LNCS. 2009, pp. 278–299.

[DSC+19] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. "CHET: an optimizing compiler for fully-homomorphic neural-network inferencing." In: *PLDI*. ACM, 2019, pp. 142–156.

[Fau99] Jean-Charles Faugére. "A new efficient algorithm for computing Gröbner bases (F4)." In: *Journal of Pure and Applied Algebra* 139.1 (1999), pp. 61–88. ISSN: 0022-4049. DOI: https://doi.org/10.1016/S0022-4049(99)00005-5. URL: https://www.sciencedirect.com/science/article/pii/S0022404999000055.

[FV12] Junfeng Fan and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption." In: *IACR Cryptol. ePrint Arch.* 2012 (2012), p. 144.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices." In: *STOC*. ACM, 2009, pp. 169–178.

[GHR+22] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "A New Feistel Approach Meets Fluid-SPN: Griffin for Zero-Knowledge Applications." In: *IACR Cryptol. ePrint Arch.* (2022), p. 403.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Homomorphic Evaluation of the AES Circuit." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 850–867.

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation." In: *CCS*. ACM, 2022, pp. 1323–1335.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems." In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. "On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy." In: *EUROCRYPT*. Vol. 12106. LNCS. Springer, 2020, pp. 674–704.

[GØS+22] Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. "From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications." In: *IACR Cryptol. ePrint Arch.* (2022), p. 342.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. "The PHOTON Family of Lightweight Hash Functions." In: *CRYPTO*. Vol. 6841. LNCS. Springer, 2011, pp. 222–239.

[GPP+11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. "The LED Block Cipher." In: *CHES*. Vol. 6917. LNCS. Springer, 2011, pp. 326–341.

[HKC+20] Jincheol Ha, Seongkwang Kim, Wonseok Choi, Jooyoung Lee, Dukjae Moon, Hyojin Yoon, and Jihoon Cho. "Masta: An HE-Friendly Cipher Using Modular Arithmetic." In: *IEEE Access* 8 (2020), pp. 194741–194751.

[HKL+22] Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Jooyoung Lee, and Mincheol Son. "Rubato: Noisy Ciphers for Approximate Homomorphic Encryption." In: *EUROCRYPT*. Vol. 13275. LNCS. Springer, 2022, pp. 581–610.

[HL20] Phil Hebborn and Gregor Leander. "Dasta - Alternative Linear Layer for Rasta." In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 46–86.

[HS14] Shai Halevi and Victor Shoup. "Algorithms in HElib." In: *CRYPTO*. Vol. 8616. LNCS. Springer, 2014, pp. 554–571.

[HS15] Shai Halevi and Victor Shoup. "Bootstrapping for HElib." In: *EUROCRYPT*. Vol. 9056. LNCS. Springer, 2015, pp. 641–670.

[HS18] Shai Halevi and Victor Shoup. "Faster Homomorphic Linear Transformations in HElib." In: *CRYPTO*. Vol. 10991. LNCS. Springer, 2018, pp. 93–120.

[HS20] Shai Halevi and Victor Shoup. "Design and implementation of HElib: a homomorphic encryption library." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1481.

[JK97] Thomas Jakobsen and Lars R. Knudsen. "The Interpolation Attack on Block Ciphers." In: *FSE*. Vol. 1267. LNCS. Springer, 1997, pp. 28–40.

[JV17] Antoine Joux and Vanessa Vitse. "A Crossbred Algorithm for Solving Boolean Polynomial Systems." In: *NuTMiC*. Vol. 10737. LNCS. Springer, 2017, pp. 3–21.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. "GAZELLE: A Low Latency Framework for Secure Neural Network Inference." In: *USENIX Security Symposium*. USENIX Association, 2018, pp. 1651–1669.

[Knu94] Lars R. Knudsen. "Truncated and Higher Order Differentials." In: *FSE 1994*. Vol. 1008. LNCS. Springer, 1994, pp. 196–211.

[Lai94] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis." In: *Communications and Cryptography: Two Sides of One Tapestry*. 1994, pp. 227–233.

[LAW+22] Fukang Liu, Ravi Anand, Libo Wang, Willi Meier, and Takanori Isobe. "Coefficient Grouping: Breaking Chaghri and More." In: *IACR Cryptol. ePrint Arch.* (2022), p. 991.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings." In: *EUROCRYPT*. Vol. 6110. LNCS. Springer, 2010, pp. 1–23.

[LSM+21] Fukang Liu, Santanu Sarkar, Willi Meier, and Takanori Isobe. "Algebraic Attacks on Rasta and Dasta Using Low-Degree Equations." In: *ASIACRYPT*. Vol. 13090. LNCS. Springer, 2021, pp. 214–240.

[Mat93] Mitsuru Matsui. "Linear Cryptanalysis Method for DES Cipher." In: *EUROCRYPT*. Vol. 765. LNCS. Springer, 1993, pp. 386–397.

[MCJ+19] Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. "Improved Filter Permutators for Efficient FHE: Better Instances and Implementations." In: *INDOCRYPT*. Vol. 11898. LNCS. Springer, 2019, pp. 68–91.

[NIS15] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. National Institute of Standards and Technology (NIST), FIPS PUB 202, U.S. Department of Commerce. 2015.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. "Can homomorphic encryption be practical?" In: *CCSW*. ACM, 2011, pp. 113–124.

[NNY18] Ruben Niederhagen, Kai-Chun Ning, and Bo-Yin Yang. "Implementing Joux-Vitse's Crossbred Algorithm for Solving MQ Systems over GF(2) on GPUs." In: *PQCrypto*. Vol. 10786. LNCS. Springer, 2018, pp. 121–141.

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes." In: *EUROCRYPT*. Vol. 1592. LNCS. Springer, 1999, pp. 223–238.

[Pas] *Microsoft SEAL (release 3.6)*. https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA. Nov. 2020.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. "On Data Banks and Privacy Homomorphisms." In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179.

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography." In: *STOC*. ACM, 2005, pp. 84–93.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." In: *Commun. ACM* 21.2 (1978), pp. 120–126.

[SV14] Nigel P. Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations." In: *Des. Codes Cryptogr.* 71.1 (2014), pp. 57–81.

[Vie07] Michael Vielhaber. "Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack." In: *IACR Cryptology ePrint Archive* 2007 (2007), p. 413.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. “SoK: Fully Homomorphic Encryption Compilers.” In: *IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 1092–1108.

[WSH+22] Roman Walch, Samuel Sousa, Lukas Helminger, Stefanie N. Lindstaedt, Christian Rechberger, and Andreas Trügler. “CryptoTL: Private, efficient and secure transfer learning.” In: *CoRR* abs/2205.11935 (2022).

7

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

Publication Data

Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. "From Farfalle to Megafono via Ciminion: The PRF Hydra for MPC Applications." In: *EUROCRYPT (4)*. Vol. 14007. Lecture Notes in Computer Science. Springer, 2023, pp. 255–286

The appended paper is the author's full version available at https://eprint.iacr.org/2022/342 which has been changed to use the design and formatting of this thesis. The full version includes the missing appendices from the conference version which consist of the full cryptanalysis of Hydra, missing algorithms, and further benchmarks.

Contributions

Main author.

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

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

[1] Radboud University Nijmegen (Nijmegen, The Netherlands)
[2] Simula UiB (Bergen, Norway)
[3] Graz University of Technology (Graz, Austria)
[4] Know-Center GmbH (Graz, Austria)
[4] TACEO GmbH (Graz, Austria)

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.

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 [DPS+12; DKL+13] or protocols based on Shamir's secret sharing [Sha79]. 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.

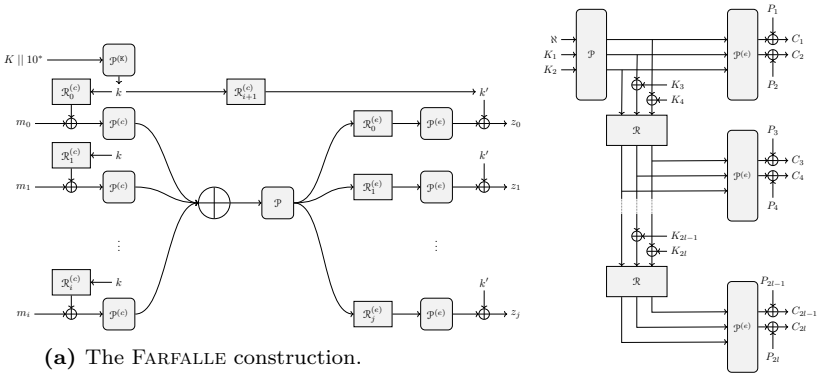
MPC has been applied to many use cases, including privacy-preserving machine learning [MZ17], private set intersection [KRS+19], truthful auctions [BDJ+06], and revocation in credential systems [HKR+21]. In the literature describing these use cases, data is often directly entered from and delivered to 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 [GRR+16], 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 [LTW13], distributed data storage [GRR+16], virtual hardware security modules[1], MPC-in-the-head based zero-knowledge proofs [IKO+07] and signatures [CDG+17], oblivious TLS [ADS+21], 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 [DGG+21] is one of the most competitive schemes for PRF applications. Proposed at Eurocrypt'21, it is based on the Farfalle mode of operation [BDH+17]. 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 [BDP+08] 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 Farfalle and Ciminion.[2] It is designed to be competitive in all

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

[2] "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) The Farfalle construction.

(b) The Ciminion PRF.

Figure 7.1: Farfalle and Ciminion (notation adapted to the one used in this paper).

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 [ARS+15], MiMC [AGR+16], GMiMC [AGP+19], HadesMiMC [GLR+20], and Rescue [AABS+20]. 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 [DGG+21]. In the following, we briefly discuss the Farfalle construction and the MPC-friendly primitive Ciminion based on it.

Farfalle.

Farfalle [BDH+17] is an efficiently parallelizable permutation-based construction of arbitrary input and output length, taking as input a key. As shown in Fig. 7.1a and recalled in Section 3, the Farfalle construction consists of a *compression layer* followed by an *expansion layer*. The compression layer produces 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 [DGG+21], 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. 7.1b 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 [CFG+18; CG20]. 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 [GOP+22]. 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 [GLR+20], which mixes rounds with full S-box layers and rounds with partial S-box layers. Similar to Neptune [GOP+22], we use two different round functions, one for

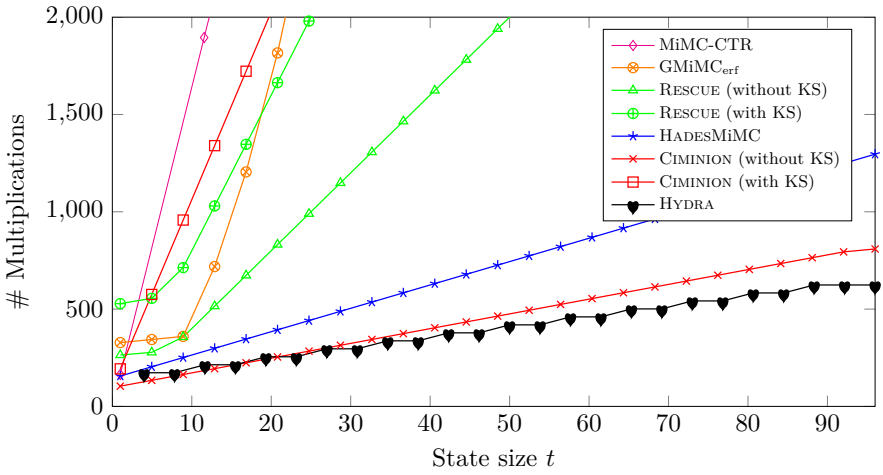


Figure 7.2: Number of MPC multiplications of several designs over $\mathbb{F}_p^t$, with $p \approx 2^{128}$ and $t \geq 2$ (security level of 128 bits).

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.

1.4 MPC Performance and Comparison

The performance of any MPC calculation scales with the number of nonlinear operations. In Figure 7.2 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 [Kel20] library and confirm the results expected from Figure 7.2. 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 $\cdot \,||\, \cdot$ 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 [DGG+21] and [GRR+16] 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 [GRR+16], 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 10. *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 [ARS+15], Rain [DKR+21], or weakPRF [DGH+21].*

2.2 Cost Metric for MPC Applications

Modern MPC protocols such as SPDZ [DPS+12; DKL+13] 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 [Bea91] 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 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

$$\#\texttt{triples} = \texttt{cost}_d := \mathrm{hw}(d) + \lfloor \log_2(d) \rfloor - 1\,, \qquad \texttt{depth}_{\text{online}} = \texttt{cost}_d\,. \quad (7.1)$$

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 Algorithm 2 and Algorithm 3).

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 [GRR+16] 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 Algorithm 4) 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 [AABS+20], the authors propose two different algorithms to evaluate R (see Algorithm 5 and Algorithm 6), 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. (7.1), 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 [BDH+17] with inputs and outputs of arbitrary length. As shown in Fig. 7.1a, 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 $\mathcal{R} : \mathbb{F}_q^n \to \mathbb{F}_q^n$.[1] 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 $x \in \mathbb{F}_q$, $^{1/2\times}\textsc{Farfalle} : \mathbb{F}_q^n \to (\mathbb{F}_q^n)^\star$ operates as $^{1/2\times}\textsc{Farfalle}(x) = y_0 \,||\, y_1 \,||\, y_2 \,||\, \cdots \,||\, y_j \,||\, \cdots$, 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 [DGG+21] 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. 7.1b, 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 [BDP+08] instantiated via the permutation $\mathcal{P}$. We refer to [DGG+21, 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

[1] We mention that in [BDH+17], 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.

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

(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

$$\mathcal{P}(x) = \mathcal{P}_{r-1} \circ \ldots \circ \mathcal{P}_1 \circ \mathcal{P}_0(x) \tag{7.2}$$

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

$$\mathcal{S}(x) := \sum_{i=0}^{r-1} \mathcal{P}_i \circ \ldots \circ \mathcal{P}_1 \circ \mathcal{P}_0(x), \tag{7.3}$$

(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 $\mathcal{R}_0(y, z) = (y, z)$.

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

$$\text{MEGAFONO}_\mathtt{K}(x) \coloneqq \mathcal{F}_{k_2}(y, z) \,||\, \mathcal{F}_{k_3}(\mathcal{R}_1(y, z)) \,||\, \cdots \,||\, \mathcal{F}_{k_{i+2}}(\mathcal{R}_i(y, z)) \,||\, \cdots$$

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 11. *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 12. *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 [BDH+17, 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 $w_i = \mathcal{F}_k(\mathcal{R}_i(y, z))$ 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 [CFG+18; CG20]. In this scenario, one attack consists of solving the system of equations $\{w_i - \mathcal{F}_k(\mathcal{R}_i(y, z)) = 0\}_{i\geq 0}$ 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 [EM91] of the form $x \mapsto \mathtt{K} + \mathcal{P}(x + \mathtt{K})$. As proven in [Dae91; DKS12], 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

$$y_j - y_h = \mathcal{P}^{(e)}(\mathcal{R}_j(x)) - \mathcal{P}^{(e)}(\mathcal{R}_h(x)) \tag{7.4}$$

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 [BBL+22] 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. (7.4)) – 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 $v_j = \mathcal{C}_k(\mathcal{R}_j(y,z))$ and $v_l = \mathcal{C}_k(\mathcal{R}_l(y,z))$ 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 $w_j = \mathcal{F}_k(\mathcal{R}_j(y,z))$ and $w_l = \mathcal{F}_k(\mathcal{R}_l(y,z))$,

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

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 [BDH+17, 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 [CGG+22] and/or unaligned SPN schemes [BDK+21], 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 nontrivial, 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. (7.3). 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 $z = \mathcal{S}(\mathcal{P}^{-1}(y - \mathtt{K}))$.[1] 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 in the case of the keys $k_0, k_1, k_2, \ldots$ 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 [MN17], 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 [ALP+19], 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

[1] 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.

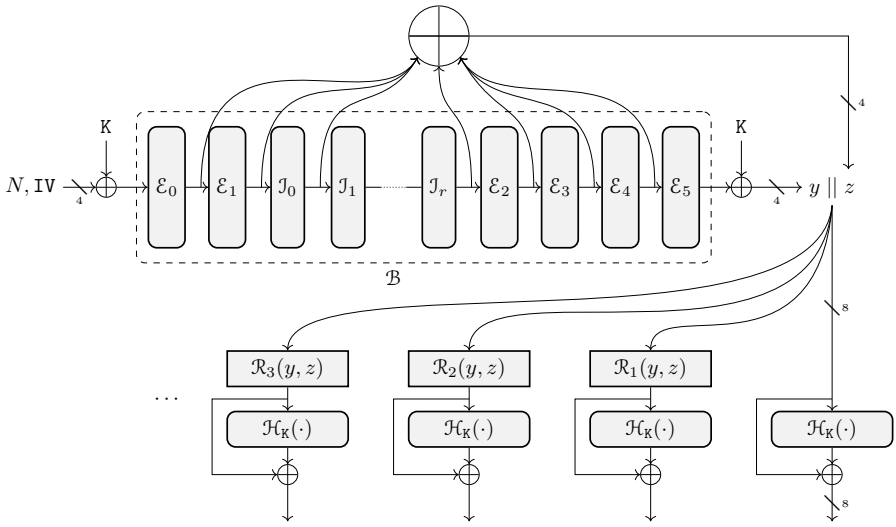


Figure 7.3: The Hydra PRF (where $r := R_{\mathcal{I}} - 1$ for aesthetic reasons).

applications were also proposed for Farfalle, we do not describe them here, but refer to [BDH+17, 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 [BDP+11], defined over $\mathbb{F}_2^n$ for $n = 25 \cdot 2^l$ for $l \in \{0, 1, \ldots, 6\}$. 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 [BDH+17, 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 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

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

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

Hydra.

An overview of HYDRA[1] is given in Fig. 7.3, where

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

(3) $\mathcal{H}_{\mathtt{K}} : \mathbb{F}_p^8 \to \mathbb{F}_p^8$ 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

$$\forall i \geq 1 : \qquad \mathcal{R}_i(y, z) := \varphi_i + \mathcal{R} \circ \mathcal{R}_{i-1}(y, z)\,, \tag{7.5}$$

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 Algorithm 7.

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

$$\mathcal{B}(x) = \underbrace{\mathcal{E}_5 \circ \cdots \circ \mathcal{E}_2}_{4 \text{ times}} \circ \underbrace{\mathcal{I}_{R_\mathcal{I}-1} \circ \cdots \circ \mathcal{I}_0}_{R_\mathcal{I} \text{ times}} \circ \underbrace{\mathcal{E}_1 \circ \mathcal{E}_0}_{2 \text{ times}}(M_\mathcal{E} \times x), \tag{7.6}$$

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 Appendix E for details on how we generate the pseudo-random constants).

[1] 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 $\mathcal{H}_{\mathtt{K}}$ as its multiple heads.

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

$$y_l = x_l + \left(\left(\sum_{j=0}^{3}(-1)^j \cdot x_j\right)^2 + \left(\sum_{j=0}^{3}(-1)^{\lfloor j/2\rfloor} \cdot x_j\right)\right)^2 \text{ for } 0 \leq l \leq 3\,. \quad (7.7)$$

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 (7.7), 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 Section 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 Appendix H for details).

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

$$M_{\mathcal{I}} = \begin{pmatrix} \mu_{0,0}^{(\mathcal{I})} & 1 & 1 & 1 \\ \mu_{1,0}^{(\mathcal{I})} & \mu_{1,1}^{(\mathcal{I})} & 1 & 1 \\ \mu_{2,0}^{(\mathcal{I})} & 1 & \mu_{2,2}^{(\mathcal{I})} & 1 \\ \mu_{3,0}^{(\mathcal{I})} & 1 & 1 & \mu_{3,3}^{(\mathcal{I})} \end{pmatrix}, \quad (7.8)$$

for which the conditions *(a)*, *(b)*, and *(c)* are satisfied (we suggest to use the tool given in Section 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. (7.5)) 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

$$v = \left(\sum_{i=0}^{3}(-1)^i \cdot y_i\right) \cdot \left(\sum_{i=0}^{3}(-1)^{\lfloor \frac{i}{2} \rfloor} \cdot z_i\right), \; w = \left(\sum_{i=0}^{3}(-1)^i \cdot z_i\right) \cdot \left(\sum_{i=0}^{3}(-1)^{\lfloor \frac{i}{2} \rfloor} \cdot y_i\right), \tag{7.9}$$

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

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

where $M_{\mathcal{J}} \in \mathbb{F}_p^{4\times 4}$ 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 (7.7), 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 Appendix H). We recommend that $M_{\mathcal{J}}$ has a similar form to the matrix in Eq. (7.8) 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 $\mathcal{H}_{\mathtt{K}}$ must be at least

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

where $\widehat{R_{\mathcal{J}}}$ and $R^*_{\mathcal{H}}$ are the minimum positive integers that satisfy Eq. (7.15) and Eq. (7.12), respectively. Note that we have added a security margin of 12.5% for $\mathcal{B}$ and 25% for $\mathcal{H}_{\mathtt{K}}$. In Appendix A, we provide a script that returns the number of rounds $R_{\mathcal{J}}$ and $R_{\mathcal{H}}$ for given p and κ. For instance, with $\kappa = 128$, we get $R_{\mathcal{J}} = 42$ and $R_{\mathcal{H}} = 39$. A concrete instantiation of Hydra's matrices for $p = 2^{127} + 45$ is given in in 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 [DPS+12; DKL+13]), 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 [GLR+20; DGG+21].

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

6.1 The Body $\mathcal{B}$

The Hades Design Strategy.

For $\mathcal{B}$, we aim to retain the advantages of Hades [GLR+20], 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 [GOP+22], a modified version of the sponge hash function Poseidon [GKR+21]. 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. [GOP+22] 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 Section 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 [DR01] (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 [LM90]. 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 [GOP+22], 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 $\mathcal{F}(x_0, \ldots, x_{n-1}) = (y_0, \ldots, y_{n-1})$ is invertible when*

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

We provide the proof in Section F.1. No conditions are imposed on F. Even if not strictly necessary, we choose $\{\lambda_j^{(0)}\}_{j=0}^{n-1}, \ldots, \{\lambda_j^{(l-1)}\}_{j=0}^{n-1}$ such that they are linearly independent. Since we require $\sum_{j=0}^{n-1} \lambda_j^{(i)} = 0$ for $i \in \{0, \ldots, l-1\}$, 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 [Vau99]. Hence, it is crucial to choose the matrix $M_\mathcal{J}$ in order to break them. For this goal, in Appendix H, we show how to adapt the analysis/tool proposed in [GRS21; GSW+21] for breaking arbitrarily long subspace trails for P-SPN schemes to the case of the generalized Lai–Massey constructions. In particular, based on [GRS21, 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 $i \in \{0, 1, \ldots, l-1\}$: $\sum_{j=0}^{n-1} \lambda_j^{(i)} \cdot \left(\sum_{k=0}^{n-1} M_\mathcal{J}[j,k]\right) \neq 0$,

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

We give further details on these two conditions in Section 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 $\mathcal{H}_\mathtt{K}$ 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 $M_\mathcal{J}$). 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 $\mathcal{H}_\mathtt{K}$ 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 [DHA+18] and Ciminion. This has multiple advantages, such as frustrating the meet-in-the-middle attacks on the expansion phase described in [CFG+18; CG20] 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 and e.g. avoid certain approaches such as slide attacks [BW99]. 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 $\mathcal{F}(x_0, \ldots, x_{n-1}) = (y_0, \ldots, y_{n-1})$ is invertible for*

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

The proof is given in Section F.2. We impose that $(\lambda_0, \ldots, \lambda_{n'-1})$, $(\varphi'_0, \ldots, \varphi'_{n'-1}) \in \mathbb{F}_p^{n'}$ and $(\varphi_0, \ldots, \varphi_{n'-1})$, $(\lambda'_0, \ldots, \lambda'_{n'-1}) \in \mathbb{F}_p^{n'}$ are pairwise linearly independent, in order to guarantee that the variables v and w in Eq. (7.9) 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 [GLR+20, 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 [GLR+20, Sect. 4.2 – 5.2] regarding algebraic attacks can be easily adapted to the case of $\mathcal{B}$.

We refer to 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 Appendix G, keeping in mind that the Even–Mansour construction cannot guarantee more than $2 \cdot \log_2(p) \geq \kappa$ bits of security [Dae91; DKS12] (this value is reached when $\mathcal{B}$ resembles a PRP).

Finally, in 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 [GRS21] 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 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 Section 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 [CLO13] 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 [Fau99]. There is, however, an important class of polynomial systems, known as semi-regular (see [BFS+05] 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

$$H(z) = \frac{\prod_{i=1}^{n_e}(1-z^{d_i})}{(1-z)^{n_v}}, \tag{7.10}$$

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

$$\mathcal{O}\left(\binom{D_{\text{reg}} + n_v}{n_v}^{\omega}\right), \tag{7.11}$$

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 [BFS+05].

Gröbner Basis Attacks on $\mathcal{H}_{\mathtt{K}}$.

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 Section 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 Section 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. (7.10). 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. (7.11) and define $R^*_{\mathcal{H}} = R^*_{\mathcal{H}}(\kappa)$ to be the minimum positive integer such that

$$\binom{2R^*_{\mathcal{H}} - 2 + D_{\text{reg}}}{2R^*_{\mathcal{H}} - 2}^2 \geq 2^{\kappa}, \tag{7.12}$$

where D_{reg} is computed from Eq. (7.10) 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.

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. (7.10), 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.

Table 7.1: 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

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 [Kel20][1] (version 0.2.8, files can be found in Appendix A) and benchmark it using SPDZ [DPS+12; DKL+13] with the MASCOT [KOS16] 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 instantiated with $\kappa = 128$. Hydra requires $4 \cdot R_{\mathcal{E}} \cdot (\mathrm{hw}(d) + \lfloor \log_2(d) \rfloor - 1) + 2 \cdot R_{\mathcal{Y}} + (R_{\mathcal{H}} + 2) \cdot \lceil \frac{t}{8} \rceil - 2$ multiplications, hence $130 + 41 \cdot \lceil \frac{t}{8} \rceil$ in this setting.

We implemented all x^3 evaluations using the technique from [GRR+16] (see Algorithm 4), 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 [AABS+20] (see Algorithm 6). MP-SPDZ allows to precompute squares and inverses from Beaver triples in an additional communication round in the offline phase (see Section 2).

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

Table 7.2: 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.

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 (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.

In Table 7.1, we compare the performance of Hydra to some competitors when encrypting t plaintext words,[1] for a comparison with more PRFs we refer to 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 7.1 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 Appendix K) and Rescue requires expensive $x^{1/d}$ evaluations.

For the sake of completeness, in Table 7.2 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

[1] 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.

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 triples 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.

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 7.1 and Table 7.2). 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.

A Supplementary Files

In the repository

`https://extgit.iaik.tugraz.at/krypto/hydra`

we provide the following files as supplementary material.

- `MP-SPDZ`: This folder contains the MP-SPDZ framework and cipher implementations used for benchmarking.
- `calc_round_numbers.sage`: This script calculates the number of rounds for a given security level κ.
- `hydra.sage`: This script contains the reference implementation of Hydra, written in sage.

We refer to the `Readme.md` file in the repository for more information about the supplied files.

B Megafono and Hydra Version 0

In the first version of this paper, Megafono and Hydra were different from the ones proposed in this current version. Here we list the main differences.

About Megafono Version 0:

- in the first version, a final truncation-summation techniques was applied to the outputs of the expanding parts. We decided to replace it with a feed-forward operation both in order to avoid wasting encryption material and in order to increase the security with respect to guessing attacks;
- we apply the rolling function on the entire state $y\|z$, as opposed to only z. We also suggest to make use of a nonlinear rolling function;
- we re-formulated the security assumption of Megafono regarding the independence between y and z;
- we also added a discussion regarding the possibility to use Megafono for more traditional confidentiality/authenticity goals as a possible replacement of Farfalle.

About Hydra Version 0:

- the main change regards the fact that we replaced the `Amaryllises` construction with power maps. We noticed that for the proposed design, the construction instantiated with the power maps has almost the same security and efficiency as the one instantiated with `Amaryllises`. Using power maps helps simplify the construction, and allows us to devote more space to discuss the design rationale. We point out that `Amaryllises` is discussed in [Gra22] instead;

- due to this change, the body of Hydra is almost equivalent to the cipher HadesMiMC. As a result, we can simplify the security analysis by re-using results from the HadesMiMC paper;
- we decided to instantiate $\mathcal{B}$ with only 6 external rounds, and show that this is indeed sufficient for achieving security;
- we replaced the linear rolling function with a nonlinear one.

We emphasize that we are currently not aware of any attack on Hydra Version 0.

C Hydra Matrices for $p = 2^{127} + 45$

In this section we give concrete instantiations of the matrices used in the Hydra PRF which fulfill all conditions of Section 5 for $\mathbb{F}_p$ with $p = 2^{127} + 45$. More specifically, for this prime we instantiate the body of Hydra with the MDS matrix $M_{\mathcal{E}}$ in the external rounds, where

$$M_{\mathcal{E}} = \begin{pmatrix} 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \\ 2 & 1 & 1 & 3 \end{pmatrix},$$

the matrix $M_{\mathcal{I}}$ in the internal rounds, where

$$M_{\mathcal{I}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 1 & 1 \\ 3 & 1 & 3 & 1 \\ 4 & 1 & 1 & 2 \end{pmatrix},$$

and finally, the matrix $M_{\mathcal{J}}$ in the heads of Hydra, where

$$M_{\mathcal{J}} = \begin{pmatrix} 3 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 7 & 3 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4 & 1 & 4 & 1 & 1 & 1 & 1 & 1 \\ 3 & 1 & 1 & 8 & 1 & 1 & 1 & 1 \\ 7 & 1 & 1 & 1 & 7 & 1 & 1 & 1 \\ 8 & 1 & 1 & 1 & 1 & 5 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 1 & 2 & 1 \\ 4 & 1 & 1 & 1 & 1 & 1 & 1 & 6 \end{pmatrix}.$$

D MPC Subprotocols

In this section, we restate well known MPC subprotocols, such as beaver multiplication (Algorithm 1), generating shared squares (Algorithm 2) and shared inverses (Algorithm 3) from Beaver triples, calculating $[x]^3$ (Algorithm 4), as well as two methods for calculating $[x]^d$ (Algorithm 5 and Algorithm 6).

Algorithm 1: Beaver Multiplications [Bea91].

Data: $[x], [y]$
Result: $[xy]$
`// Offline Phase`
$[a], [b], [ab] \leftarrow$ `GenTriple`()
`// Online Phase`
$s \leftarrow \text{Open}([x] - [a])$
$t \leftarrow \text{Open}([x] - [b])$
$[xy] \leftarrow [ab] + [x] \cdot t + [y] \cdot s - s \cdot t$
return $[xy]$

Algorithm 2: Random square from Beaver triple.

Data: $[a], [b], [ab]$
Result: $[a], [a^2]$
$y \leftarrow \text{Open}([a] + [b])$
$[a^2] \leftarrow y \cdot [a] - [ab]$
return $[a], [a^2]$

Algorithm 3: Random inverse from Beaver triple.

Data: $[a], [b], [ab]$
Result: $[a], [a^{-1}]$
$y \leftarrow \text{Open}([ab])$
if $y = 0$ **then**
    `Abort.`
$[a^{-1}] \leftarrow [b] \cdot y^{-1}$
return $[a], [a^{-1}]$

Algorithm 4: Optimized $[x]^3$ evaluation [GRR+16].

Data: $[x]$
Result: $[x^3]$
`// Offline Phase`
$[a], [b], [ab] \leftarrow$ `GenTriple`()
$[r], [r^2] \leftarrow$ `GenSquare`()
$[r^3] \leftarrow [r] \cdot [r^2]$ `// Using` $[a], [b], [ab]$
`// Online Phase`
$y \leftarrow \text{Open}([x] - [r])$
$[x^3] = 3 \cdot y \cdot [r^2] + 3 \cdot y^2 \cdot [r] + y^3 + [r^3]$
return $[xy]$

Algorithm 5: Forward method to calculate $[x]^d$ [AABS+20].

Data: $[x]$
Result: $[x^d]$
// Offline Phase
1 $[a], [b], [ab] \leftarrow$ GenTriple()
2 $[r], [r^{-1}] \leftarrow$ GenInv()
3 $[r^{-d}] \leftarrow$ Pow$([r^{-1}], d)$ // Using multiple Beaver triples or squares
// Online Phase
4 $y \leftarrow \text{Open}([x] \cdot [r])$ // Using $[a], [b], [ab]$
5 $[x^d] \leftarrow y^d \cdot [r^{-d}]$
6 **return** $[x^d]$

Algorithm 6: Backwards method to calculate $[x]^d$ [AABS+20].

Data: $[x]$
Result: $[x^d]$
// Offline Phase
1 $[a], [b], [ab] \leftarrow$ GenTriple()
2 $[r], [r^{-1}] \leftarrow$ GenInv()
3 $[r^{1/d}] \leftarrow$ Pow$([r], 1/d)$ // Using multiple Beaver triples or squares
// Online Phase
4 $y \leftarrow \text{Open}([x] \cdot [r^{1/d}])$ // Using $[a], [b], [ab]$
5 $[x^d] \leftarrow y^d \cdot [r^{-1}]$
6 **return** $[x^d]$

E The Hydra PRF – Specification Details

Pseudo Code.

A pseudo code of Hydra is proposed in Algorithm 7.

Algorithm 7: The Hydra PRF.

Data: Prime integer $p \geq 2^{63}$, $4 \leq t \leq 2^{\kappa/2}$, $N \in \mathbb{F}_p$, $\mathtt{K} \in \mathbb{F}_p^4$.
Result: $h \in \mathbb{F}_p^t$.
Let $t = 8 \cdot t' + t''$ for $t', t'' \in \mathbb{N}$, where $t'' = t \mod 8$.
Let $x, y, z \leftarrow 0 \in \mathbb{F}_p^4$.
`// First step (computing` $\mathcal{B}$`)`
$x \leftarrow M_{\mathcal{E}} \times (\mathtt{K} + [N \mid\mid \mathtt{IV}])$.
for $i \leftarrow 0$ **to** 1 **do**
    $x \leftarrow \mathcal{E}_i(x)$.
    $z \leftarrow z + x$.
for $i \leftarrow 0$ **to** $R_{\mathcal{J}} - 1$ **do**
    $x \leftarrow \mathcal{J}_i(x)$.
    $z \leftarrow z + x$.
for $i \leftarrow 2$ **to** 5 **do**
    $x \leftarrow \mathcal{E}_i(x)$.
    $z \leftarrow z + x$.
$y \leftarrow \mathcal{E}_7(x) + \mathtt{K}$.
`// Expansion step (using` $\mathcal{H}_{\mathtt{K}}$`)`
for $i \leftarrow 0$ **to** t' **do**
    $h_i \leftarrow \mathcal{H}_{\mathtt{K}}(\mathcal{R}_i(y, z))$.
$h_{t'} \leftarrow T_{8,t''}(h_{t'})$.
return $h := h_0 \mid\mid h_1 \mid\mid \cdots \mid\mid h_{t'-1} \mid\mid h_{t'} \in \mathbb{F}_p^t$.

Generation of Matrices and Constants.

We pseudo-randomly generate all matrices and constants using Shake-128 [BDP+11] seeded with the string *Hydra* and the used prime. Thereby, we use rejection sampling to sample field elements, and we reject and resample matrices and constants if they do not meet the requirements specified in this paper. We refer to our source code present in the supplementary material (Appendix A) for more details.

F Proofs

F.1 Proof of Proposition 1

Proof 1. *As in a Lai–Massey construction, the invertibility follows from the fact that for* $i \in \{0, 1, \ldots, l-1\}$

$$\sum_{h=0}^{n-1} \lambda_h^{(i)} \cdot y_h = \sum_{h=0}^{n-1} \lambda_h^{(i)} \cdot \left(x_h + F\left(\sum_{j=0}^{n-1} \lambda_j^{(1)} x_j, \sum_{j=0}^{n-1} \lambda_j^{(2)} x_j, \ldots, \sum_{j=0}^{n-1} \lambda_j^{(t)} x_j \right) \right)$$
$$= \sum_{h=0}^{n-1} \lambda_h^{(i)} x_h + \underbrace{\sum_{h=0}^{n-1} \lambda_h^{(i)}}_{=0} \cdot F\left(\sum_{j=0}^{n-1} \lambda_j^{(1)} x_j, \sum_{j=0}^{n-1} \lambda_j^{(2)} x_j, \ldots, \sum_{j=0}^{n-1} \lambda_j^{(t)} x_j \right) = \sum_{h=0}^{n-1} \lambda_h^{(i)} x_h .$$

It follows that $x_h = y_h - F\left(\sum_{j=0}^{n-1} \lambda_j^{(0)} \cdot y_j, \sum_{j=0}^{n-1} \lambda_j^{(1)} \cdot y_j, \ldots, \sum_{j=0}^{n-1} \lambda_j^{(l-1)} \cdot y_j \right)$ *for each* $h \in \{0, 1, \ldots, n-1\}$. □

F.2 Proof of Proposition 2

Proof 2. *We will show the invertibility of* $\mathcal{F}$ *by constructing a pre-image for any given* $(y_0, \ldots, y_{n-1}) \in \mathbb{F}_q^n$. *To this end, define*

$$a = \sum_{i=0}^{n'-1} \lambda_i \cdot y_i \,, \quad a' = G(a) \qquad \text{and} \qquad b = \sum_{i=n'}^{n-1} \lambda'_{i-n'} \cdot y_i \,, \quad b' = H(b) \,,$$

and consider the $n \times n$*-matrix* $M_{a,b} = \begin{bmatrix} I & A \\ B & I \end{bmatrix}$, *where*

$$A = a' \cdot \begin{bmatrix} \varphi_0 & \varphi_1 & \cdots & \varphi_{n'-1} \\ \varphi_0 & \varphi_1 & \cdots & \varphi_{n'-1} \\ \vdots & & \ddots & \vdots \\ \varphi_0 & \varphi_1 & \cdots & \varphi_{n'-1} \end{bmatrix}, \qquad B = b' \cdot \begin{bmatrix} \varphi'_0 & \varphi'_1 & \cdots & \varphi'_{n'-1} \\ \varphi'_0 & \varphi'_1 & \cdots & \varphi'_{n'-1} \\ \vdots & & \ddots & \vdots \\ \varphi'_0 & \varphi'_1 & \cdots & \varphi'_{n'-1} \end{bmatrix},$$

and I *is the* $n' \times n'$*-identity matrix. By Theorem 3 in [Sil00], we have*

$$\det(M_{a,b}) = \det(I - B \times A) \,,$$

and we can furthermore write

$$B \times A = a' \cdot b' \cdot \left(\sum_{j=0}^{n'-1} \varphi'_j \right) \cdot \begin{bmatrix} \varphi_0 & \varphi_1 & \cdots & \varphi_{n'-1} \\ \varphi_0 & \varphi_1 & \cdots & \varphi_{n'-1} \\ \vdots & & \ddots & \vdots \\ \varphi_0 & \varphi_1 & \cdots & \varphi_{n'-1} \end{bmatrix} .$$

The assumption $\left(\sum_{j=0}^{n'-1} \varphi'_j \right) = 0$ *ensures that* $B \times A$ *is the zero matrix, and hence* $\det(M_{a,b}) = 1$. *In particular,* $M_{a,b}$ *is invertible and there is a tuple*

$(x_0, \ldots, x_{n-1}) \in \mathbb{F}_q^n$ *satisfying*

$$M_{a,b} \times (x_0, \ldots, x_{n-1})^\top = (y_0, \ldots, y_{n-1})^\top .$$

In particular, we have that $(x_0, \ldots, x_{n-1})$ *is the pre-image of* $(y_0, \ldots, y_{n-1})$ *under* $\mathcal{F}$*. Indeed, by construction of* $M_{a,b}$*, we only need to show that* $a = \sum_{i=0}^{n'-1} \lambda_i \cdot x_i$ *and that* $b = \sum_{i=n'}^{n-1} \lambda'_{i-n'} \cdot x_i$*. In a manner similar to that of Section F.1, we get*

$$a = \sum_{i=0}^{n'-1} \lambda_i \cdot y_i = \sum_{i=0}^{n'-1} \lambda_i \cdot x_i + a' \cdot \left(\sum_{j=n'}^{n-1} \varphi_{j-n'} \cdot x_j \right) \underbrace{\left(\sum_{h=0}^{n'-1} \lambda_h \right)}_{=0} = \sum_{i=0}^{n'-1} \lambda_i \cdot x_i .$$

The case of b *is identical, which completes the proof.* □

G Details about the Security Analysis of $\mathcal{B}$

In this section, we assume the attacker knows both the inputs and outputs of $x \mapsto \mathtt{K} + \mathcal{B}(x + \mathtt{K})$. The aim is to determine a sufficient number of rounds to guarantee 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 data complexity at most $2^{\kappa/2}$.

G.1 Statistical Attacks

The security of $\mathcal{B}$ against statistical attacks as the differential one [BS90; BS93], truncated differentials [Knu94], and others, follows the security argument proposed for HadesMiMC and Poseidon. Indeed, the external rounds of $\mathcal{B}$ are the same as in HadesMiMC and Poseidon, and the security argument of HadesMiMC and Poseidon against statistical attacks depends on these external rounds only. In [GLR+20, Sect. 4 – 5], it is shown that the permutation

$$x \mapsto \underbrace{\mathcal{E}_3 \circ \mathcal{E}_2}_{2 \text{ times}} \circ \mathcal{L} \circ \underbrace{\mathcal{E}_1 \circ \mathcal{E}_0}_{2 \text{ times}}(\cdot),$$

where $\mathcal{E}_i$ are the external rounds and $\mathcal{L} : \mathbb{F}_p^{4\times 4} \to \mathbb{F}_p^4$ is an invertible linear layer that is secure against statistical attacks. Moreover, the security against statistical attacks does not decrease when replacing $\mathcal{L}$ with an invertible nonlinear function (i.e., the internal rounds). Hence we refer to the analysis proposed in [GLR+20], from which we deduce that statistical attacks do not pose any threat to $\mathcal{B}$.

G.2 Algebraic Attacks

We mainly use the internal rounds of $\mathcal{B}$ in order to gain security against algebraic attacks. First, we point out that no subspace trail [LAA+11; LMR15; GRR17] can cover an arbitrary number of rounds $\mathcal{I}$, due to the choice of the matrix $M_{\mathcal{I}}$, as explained in Appendix H. In there, we show how to adapt the analysis/tool proposed in [GRS21; GSW+21] for breaking arbitrarily long subspace trail for P-SPN schemes to the case of the generalized Lai–Massey constructions.

Density of the Algebraic Representation.

Algebraic attacks are especially efficient against schemes that have a simple algebraic structure in larger fields. It is well-known that many attacks, such as interpolation and Gröbner basis attacks, can exploit a sparse polynomial representation of the scheme. Consequently, it is important to study the density of these polynomials. We recall (see, e.g, Theorem 2.4 in [DPS20]) that the number of possible monomials in a polynomial of degree D in t variables is

$$N(d,t) \coloneqq \binom{t+D}{D}. \tag{7.13}$$

The conditions *(a)* and *(b)* on $M_{\mathcal{I}}$, described in Section 5.2, are introduced to ensure that the density of the polynomials representing the scheme are close to this upper bound. We justify them in the following.

Conditions on the Linear Layer. Let $x \in \mathbb{F}_p^t$ be the input of one round. By simple computation, the j-th $\mathbb{F}_p$-output of the next round is

$$y_j = \sum_{l=0}^{t-1} M_{\mathcal{I}}[j,l] \cdot x_l + \left(\sum_{l=0}^{t-1} M_{\mathcal{I}}[j,l]\right) \cdot G\left(\sum_{h=0}^{t-1} \lambda_h \cdot x_h\right)$$

for a certain (simplified) quadratic function G, and where we omitted constant additions for simplicity. Let $\hat{y} = G\left(\sum_{h=0}^{t-1} \lambda_h \cdot x_h\right)$. By applying the nonlinear $S_{\mathcal{I}}$, the i-th $\mathbb{F}_p$-output is

$$\begin{aligned} &y_i + G\left(\sum_{h=0}^{t-1} \lambda_h \cdot \left(\sum_{l=0}^{t-1} M_{\mathcal{I}}[h,l] \cdot x_l + \left(\sum_{l=0}^{t-1} M_{\mathcal{I}}[h,l]\right) \cdot \hat{y}\right)\right) \\ =&y_i + G\left(\sum_{l=0}^{t-1} x_l \cdot \left(\sum_{h=0}^{t-1} \lambda_h \cdot M_{\mathcal{I}}[h,l]\right) + \hat{y} \cdot \sum_{h=0}^{t-1} \lambda_h \cdot \left(\sum_{l=0}^{t-1} M_{\mathcal{I}}[h,l]\right)\right). \end{aligned}$$

Hence, the condition $\sum_{h=0}^{t-1} \lambda_h \cdot \left(\sum_{l=0}^{t-1} M_{\mathcal{I}}[h,l]\right) \neq 0$ is crucial in order to ensure the growth of the degree, while the conditions $\sum_{h=0}^{t-1} \lambda_h \cdot M_{\mathcal{I}}[h,l] \neq 0$ for each $l \in \{0,1,\ldots,t-1\}$ ensure that the polynomial is not sparse.

Forward Direction Density. For the external rounds $\mathcal{E}$, our practical tests (Fig. 7.4) indicate that the maximum number of possible monomials is not reached at the i-th round. The growth in number of monomials is nevertheless strong, and the results seem to suggest that the discrepancy can be more than accounted for by including an additional round $\mathcal{E}$. For the internal rounds $\mathcal{I}$, the growth of the number of monomials is closer to the optimum, reaching its maximum in some of our tests (Fig. 7.5).

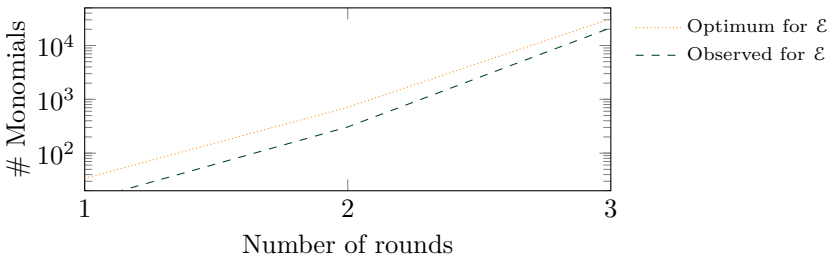


Figure 7.4: Comparison of the maximum number of monomials and the observed number of monomials in $\mathcal{B}$ for the $\mathcal{E}$ rounds.

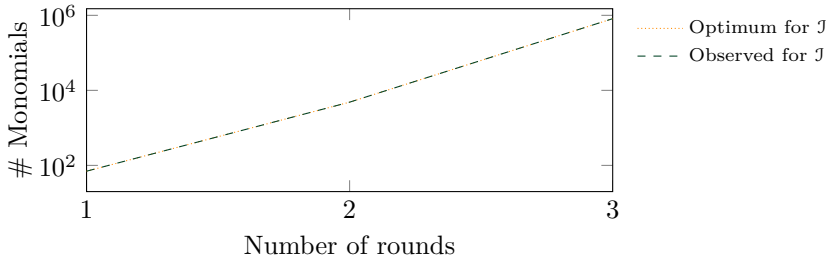


Figure 7.5: Comparison of the maximum number of monomials and the observed number of monomials in $\mathcal{B}$ for the $\mathcal{I}$ rounds.

Backward Direction Density. We note that the inverse of each $\mathcal{E}$ round has a larger degree than the forward direction for small d, since $x \mapsto x^{1/d}$ is used instead of $x \mapsto x^d$. The polynomial is also more dense, and hence we conclude that an $\mathcal{E}^{-1}$ round provides at least the degree and density of an $\mathcal{E}$ round.

Interpolation Attack.

The goal of the interpolation attack [JK97] is to construct an interpolation polynomial that describes the function. The cost of setting up such an attack depends on the number of monomials in the interpolation polynomial, which can be estimated with the degree of the function. If the number of unknown monomials is sufficiently large, then it is not possible to construct the interpolation polynomial faster than using a brute-force attack. Roughly speaking, if the interpolation polynomial is dense and has a sufficiently high degree, this attack does not work.

Let us consider a scenario in which only one input component is active and the others are fixed, that is, the polynomial depends only on a single variable. We make the following three conservative assumptions when estimating the degree growth for a MitM interpolation attack. Firstly, we only count one out of the two initial external rounds. This is to account for the fact that the external rounds do not produce dense polynomials of maximal degree, as observed in Fig. 7.4. Secondly, we arbitrarily discount 3 internal rounds in order to destroy possible

relations existing between the coefficients of the monomials (due to the fact that the degree-4 function that defines the nonlinear layer is not generic, but has a particular structure). Thirdly, we expect that three external rounds are needed to prevent an attacker going in the backwards direction. Thus only one of the four final rounds will be counted toward our degree estimate. Since the data available for constructing the polynomial is restricted by $2^{\kappa/2}$, the number of internal rounds $R_{\mathcal{J}}$ must satisfy

$$4^{R_{\mathcal{J}}-3} \cdot d^2 \geq 2^{\kappa/2} \implies R_{\mathcal{J}} \geq \frac{\kappa}{4} - \log_2(d) + 3,$$

where 4 and $d \geq 3$ are the degrees of the internal and the external rounds respectively. Finally, we add an extra 3 rounds in order to destroy invariant subspaces of the Lai–Massey scheme, and hence estimate that

$$R_{\mathcal{J}} \geq \frac{\kappa}{4} - \log_2(d) + 6 \tag{7.14}$$

rounds are necessary for preventing MitM interpolation attacks.

Higher-Order Differential Attack.

Given a vectorial Boolean function $\mathcal{F}$ over $\mathbb{F}_2^n$ of degree d, the higher-order differential attack [Lai94; Knu94] exploits that

$$\sum_{x \in \mathfrak{V}+v} x = \sum_{x \in \mathfrak{V}+v} \mathcal{F}(x) = 0$$

for each affine subspace $\mathfrak{V} + v \subseteq \mathbb{F}_2^n$ of dimension strictly larger than d (i.e., $\dim(\mathfrak{V}) \geq d + 1$). The corresponding attack in the case of a prime field $\mathbb{F}_p$ has recently been proposed in [BCD+20]. Since this result is related to the degree of the polynomial that describes the permutation, we claim that the number of rounds necessary to guarantee security against the interpolation attack provides security against this attack as well.

Linearization Attack.

Many well-known techniques for solving multivariate polynomial systems of equations use linearization (see, e.g., [CKP+00]). Given a system of polynomial equations, the idea is to turn it into a system of linear equations by adding new variables that replace all the monomials in the system of degree larger than 1. The resulting linear system of equations can be solved using linear algebra if there are sufficiently many equations.

Consider a system in t unknowns of degree limited by D, where the number of monomials, $N(D, t)$, is given by Eq. (7.13). The attack has a computational cost of $\mathcal{O}(N(D, t)^\omega)$ operations (for $2 < \omega \leq 3$) and a memory cost of $\mathcal{O}(N(D, t)^2)$ to store the linear equations. Depending on parameter choices, the hybrid approach which combines exhaustive search with this approach may lead to a reduced cost. Guessing $x < t$ variables leads to a complexity of

$$\mathcal{O}\left(p^x \cdot N(D, t - x)^\omega\right).$$

For simplicity, we start by assuming that the attacker can collect enough (input, output) pairs to directly linearize a polynomial system representing $\mathcal{B}$ with 4 key variables. The degree D is given by $D = d^2 \cdot 4^{R_\jmath - 3}$, using the same assumptions as the analysis for the interpolation attack that yielded Eq. (7.14). Since $2^\kappa \leq p^2$, it follows that the attack is not feasible if

$$\forall x \in \{0,1\}: \qquad p^x \cdot \left(\binom{4 - x + d^2 \cdot 4^{R_\jmath - 3}}{4 - x} \right)^2 \geq 2^\kappa ,$$

where we consider $\omega = 2$ and where we have 4 variables (namely, the key). Let $x' = 4 - x$. Note that

$$p^x \cdot \left(\binom{D + x'}{x'} \right)^2 \geq p^x \cdot \left(\frac{\prod_{j=1}^{x'} (D + j)}{x'!} \right)^2 \geq p^x \cdot \left(\frac{(D+1)^{x'}}{x'^{x'}} \right)^2 \geq p^x \cdot \left(\frac{D}{x'} \right)^{2 \cdot x'}$$

where $z^z \geq z!$ for each $z \geq 1$. Hence, this attack is prevented if

$$R_\jmath \geq 4 - \log_2(d) + \max_{x \in \{0,1\}} \frac{\kappa - x \cdot \log_2(p)}{4 \cdot (4 - x)}. \tag{7.15}$$

Recalling the restriction $4^{R_\jmath - 3} \cdot d^2 \geq 2^{\kappa/2}$ discussed in the analysis for the interpolation attack, we note that it will, in practice, not be possible to linearize directly as assumed above. Rather, algorithms such as [CKP+00] must go to a higher degree, and linearize a larger system, making the lower bound in Eq. (7.15) extremely conservative for preventing attacks of this kind. Even when ignoring the issues of data complexity, we point out that the number of rounds necessary for preventing interpolation attacks are largely sufficient for preventing linearization strategies when $\kappa = \log_2(p)$.

G.3 Gröbner Basis Attack

Given a system of n_e nonlinear equations in n_v variables, a Gröbner basis allows to factorize this system of equations and find a solution (if it exists). We refer to Section 7.4 for more details. Here we limit ourselves to recall that the cost of this attack depends on the number of nonlinear equations, their degree, the number of variables, and on the particular representation of the studied system. Here we consider the two extreme cases: one in which the attacker only works with the input and the output of the permutation (i.e., no additional variables are introduced), and one in which the attacker introduces intermediate variables in every round.

Inputs and Outputs.

The polynomial system in this modeling is the same as described in Section G.2, i.e., 4 key variables with polynomials of degree $d^2 \cdot 4^{R_\jmath - 3}$. The tools presented in Section 7.4 can be used for estimating how this attack performs depending on the number of equations the attacker has access to. Similar to what we described

Table 7.3: Running time (in seconds) for the F_4 algorithm on scaled down versions of $\mathcal{B}$ over $\mathbb{F}_{7741}$. r denotes the number of internal rounds.

r	1	2	3	4
Running Time (s)	28	433	18408	639614*

∗ The computation was not completed, but ran out of memory at the given time.

in Section G.2, this attack is optimized when it is possible to directly linearize the system, meaning that Eq. (7.15) provides a conservative lower bound on $R_{\mathcal{J}}$ to prevent this strategy as well.

Intermediate Variables.

For this strategy, we introduce additional variables in each round in order to reduce the overall degree growth. For simplicity we focus on a slightly modified permutation with only two external rounds: one at the beginning and one at the end. We emphasize that adding the remaining external rounds will not make the resulting equation system easier to solve.

In our representation, we replace both sums in the Lai–Massey construction of each internal round by two new variables. More formally, let

$$z_i^{(0)} = \sum_{j=0}^{3} \lambda_j^{(0)} \cdot x_j, \quad z_i^{(1)} = \sum_{j=0}^{3} \lambda_j^{(1)} \cdot x_j,$$

where $z_i^{(0)}, z_i^{(1)}$ are two new variables introduced in the i-th internal round. Using this approach, each internal round adds two degree-2 equations. Moreover, we use four variables for the key, and we introduce a layer of four new variables after the first external round (i.e., before starting with the internal rounds in our scenario) and after the last internal round (i.e., before the last external round in our scenario). Note that for the first internal round we can reuse the variables introduced after the first external round, and thus we do not need to add two new variables.

In total, we have thus 4 variables for the key, $2 \cdot 4 = 8$ variables for the transitions between the external and internal rounds, and $2(r-1)$ variables for r internal rounds. The number of equations is the same, and hence

$$n_e = n_v = 12 + 2(r-1).$$

Of these equations, 8 are of degree $d \geq 3$ and $4 + 2(r-1)$ are of degree 4.

We implemented this modeling for an increasing number of r intermediate rounds, and computed the associated grevlex Gröbner basis using the F_4 algorithm on the same setup described in Section I.2. The results are presented in Table 7.3. Based on these results, we find it highly unlikely that the Gröbner basis approach will outperform the other methods we have investigated. For a rough comparison, consider the interpolation attack in Section G.2, which requires $r \gtrapprox \kappa/4$. Thus, to outperform this attack, any algorithm with constant exponential growth will

have to grow by at most a factor of 16 for any additional intermediate round. The growth we observe in our experiments seems to increase from this factor. Indeed, the running time when going from 1 to 2 internal rounds is increased by a factor of around 15.5, and the factor between 2 and 3 rounds is 42.5. The factor between $r = 3$, and the data point where the computations for $r = 4$ ran out of memory is 34.7. This observation, along with the conservative choice of only including two external rounds in these tests, leads us to reasonably conjecture that the complexity of Gröbner basis attacks greatly exceeds the complexity of interpolation attacks.

The conclusion does not change when the attacker covers 2 internal rounds for free by exploiting the invariant subspace of the Lai–Massey construction.

H Preventing Infinitely Long Subspace Trails in Lai–Massey Constructions

As shown in [Vau99], a weakness of the Lai–Massey construction is the possibility to choose a nonzero input difference such that the quadratic function in the $\mathcal{I}$ rounds is not active. Here, we show how to choose the matrix $M_{\mathcal{I}}$ in order to fix this problem.

For reaching this goal, we follow the same strategy proposed for HadesMiMC in [GRS21]. Instead of talking of differences, we deal with subspaces and we make use of the *subspace trail* notation introduced in [GRR17].

Definition 1 ((Invariant) Subspace Trail [LAA+11; LMR15; GRR17])**.** *Let* $\mathcal{I} : \mathbb{F}_p^t \to \mathbb{F}_p^t$*. Let* $(\mathfrak{U}_0, \ldots, \mathfrak{U}_r)$ *denote a set of* $r + 1$ *subspaces with* $\dim(\mathfrak{U}_i) \leq \dim(\mathfrak{U}_{i+1})$*. If for each* $i \in \{1, \ldots, r\}$ *and for each* $a_i \in \mathbb{F}^t$ *there exists* $a_{i+1} \in \mathbb{F}^t$ *such that* $\mathcal{I}(\mathfrak{U}_i + a_i) \subseteq \mathfrak{U}_{i+1} + a_{i+1}$*, then* $(\mathfrak{U}_0, \ldots, \mathfrak{U}_r)$ *is a subspace trail of length* r*. If the subspace is invariant (that is,* $\mathfrak{U}_i = \mathfrak{U}_j$ *for each* $i, j = 0, \ldots, r$*), the trail is called an* invariant subspace trail*.*

As in the case of HadesMiMC, the choice of the linear layer M plays a crucial role for preventing subspace trails that can cover an arbitrary number of rounds. Here we focus on the general case where

$$y_i = x_i + F\left(\sum_{j=0}^{t-1} \lambda_j^{(1)} \cdot x_j, \sum_{j=0}^{t-1} \lambda_j^{(2)} \cdot x_j, \ldots, \sum_{j=0}^{t-1} \lambda_j^{(s)} \cdot x_j\right).$$

We note that this includes both the nonlinear layer in $\mathcal{I}$ of the $\mathcal{B}$ permutation, defined over $\mathbb{F}^4$, and the nonlinear layer in each of the rounds of the $\mathcal{H}_{\mathtt{K}}$ permutation, defined over $\mathbb{F}^8$.

For the follow-up, we recall the concept of infinitely long invariant/iterative subspace trails.

Definition 2 (Infinitely Long Invariant/Iterative Subspace Trail [GRS21])**.** *Let* $(\mathfrak{V}_0, \mathfrak{V}_1, \ldots, \mathfrak{V}_{l-1})$ *be a constant-dimensional subspace trail for* l *rounds. We call*

Algorithm 8: Determining if a given matrix is potentially vulnerable to subspace trails in HYDRA.

Data: Matrix $M \in \mathbb{F}_p^{n \times n}$, where $n \in \{4, 8\}$.
Result: `False` if there is a nonzero chance of vulnerability, `True` otherwise.

```
l ← n.
M' ← M.
for i ← 0 to l do
    if deg(Φ_{M'}) < n or Φ_{M'} is not irreducible then
        return False
    M' ← M × M'.
return True
```

this subspace trail an infinitely long iterative subspace trail of period l for the considered scheme if it repeats itself an arbitrary number of times, i.e., if

$$(\mathfrak{V}_0, \mathfrak{V}_1, \ldots, \mathfrak{V}_{l-1}, \mathfrak{V}_0, \mathfrak{V}_1, \ldots, \mathfrak{V}_{l-1}, \ldots, \mathfrak{V}_0, \mathfrak{V}_1, \ldots, \mathfrak{V}_{l-1}, \ldots)$$

is an infinitely long subspace trail.

H.1 Sufficient Condition for Preventing Infinitely Long Subspace Trails

The existence of an infinitely long subspace trail relies on a matrix M for which a nontrivial M-invariant subspace exists. Equivalently, a matrix M for which no M-invariant subspace exists is considered secure. Therefore, we can reuse [GRS21, Proposition 13], i.e., we determine if the minimal polynomial of a given $t \times t$ matrix has maximum degree t and is irreducible. If this is the case, no infinitely long subspace trails with inactive or active F functions exist, and the same result can also be applied to $M_{\mathcal{E}}$ in $\mathcal{B}$. We refer to Algorithm 8 for the detailed approach. We limit ourselves to recall the definition of minimal polynomial.

Definition 3. *Let $M \in \mathbb{F}_p^{t \times t}$ be an invertible matrix. The characteristic polynomial $\Psi \in \mathbb{F}_p[x]$ is defined as $\Psi(x) = \det(x \cdot I - M)$. The minimal polynomial $\Phi \in \mathbb{F}_p[x]$ is the monic polynomial of minimal degree such that*

1. *$\Phi(M) \times v = 0^t = (0, 0, \ldots, 0)^T \in \mathbb{F}_p^t$ for each $v \in \mathbb{F}_p^t$;*
2. *for each polynomial $P \in \mathbb{F}_p[x]$ that is annihilating (in the sense that $P(M) \times v = 0^t$ as before for each $v \in \mathbb{F}_p^t$), Φ divides P.*

This approach only provides a sufficient condition for ensuring that no invariant subspace exists. However, there may be matrices that do not satisfy it, but still provide security. We study this case in the following.

H.2 Preventing Infinitely Long Subspace Trails

As a starting point for our analysis, we first define the subspaces $\mathfrak{X}^{(i)}$.

Definition 4. *For $i \geq 0$ and $t \geq 2$, let $\mathfrak{X}^{(i)} \subseteq \mathbb{F}_p^t$ be the subspace defined as*

$$\mathfrak{X}^{(i)} = \left\{ x \in \mathbb{F}_p^t \;\middle|\; \sum_{l=0}^{t-1} \left(\lambda_l^{(h)} \cdot \left(M^j \times x \right)_l \right) = 0 \in \mathbb{F}^t \text{ for each } j \leq i, 0 \leq h \leq s \right\}$$
$$= \bigcap_{h=0}^{s} \left\{ x \in \mathbb{F}_p^t \;\middle|\; \sum_{l=0}^{t-1} \left(\lambda_l^{(h)} \cdot \left(M^j \times x \right)_l \right) = 0 \in \mathbb{F}^t \text{ for each } j \leq i \right\}. \quad (7.16)$$

Moreover, let I be the largest $i \geq t-1$ such that $1 \leq \dim(\mathfrak{X}^{(i)}) \leq t-1$ (that is, $\dim(\mathfrak{X}^{(I)}) \geq 1$ and $\dim(\mathfrak{X}^{(I+1)}) = 0$).

By taking a pair of texts in the same coset of $\mathfrak{X}^{(i)}$, the first i rounds are essentially linear, since the input difference of F is always zero. Depending on M, this behavior may repeat for an arbitrary number of rounds, and our goal is to avoid this by choosing M properly.

Inactive F Function.

We will first focus on the case where F is not active (i.e., the input differences are always zero). This corresponds to the case in which the inputs of F are elements of $\mathfrak{X}^{(i)}$ for a certain i. Working as in [GRS21], we derive the following result.

Proposition 3. *A subspace $\mathfrak{I} \subset \mathbb{F}_p^t$ defines an infinitely long subspace trail with inactive F if and only if $\mathfrak{I} \subseteq \mathfrak{X}^{(0)}$ and $M \times \mathfrak{I} = \mathfrak{I}$.*

Proof 3. *Clearly, a subspace $\mathfrak{I} \subseteq \mathfrak{X}^{(0)}$ fulfilling $M \times \mathfrak{I} = \mathfrak{I}$ generates an infinitely long invariant subspace trail with inactive F functions, since this function is inactive in the first round and then $\mathfrak{I}$ is repeated infinitely.*

Next, we show that, given an infinitely long invariant subspace trail $\mathfrak{I}$ with inactive F functions, it must satisfy $\mathfrak{I} \subseteq \mathfrak{X}^{(0)}$ and $M \times \mathfrak{I} = \mathfrak{I}$. Indeed, $\mathfrak{I} \subseteq \mathfrak{X}^{(0)}$, otherwise F would be active in the first round. Moreover, $\mathfrak{I}$ is invariant if and only if $M \times \mathfrak{I} = \mathfrak{I}$. The result follows immediately.

It remains to show that the nonexistence of infinitely long invariant subspace trails implies the nonexistence of infinitely long iterative subspace trails. For this purpose, we prove that if an infinitely long iterative subspace trail with inactive F functions exists, then an invariant one exists as well. Let $\{\mathfrak{I}, M \cdot \mathfrak{I}, M^2 \cdot \mathfrak{I}, \ldots, M^{l-1} \cdot \mathfrak{I}\}$ be an l-round iterative subspace trail. Let $\mathfrak{I}' := \langle \mathfrak{I}, M \cdot \mathfrak{I}, M^2 \cdot \mathfrak{I}, \ldots, M^{l-1} \cdot \mathfrak{I} \rangle$. By definition, $\mathfrak{I}'$ generates an invariant subspace. Moreover, it is a non-trivial subspace of $\mathbb{F}_p^t$ (that is, $\mathfrak{I}' \subset \mathbb{F}_p^t$) since on such a subspace F is never active by assumption. □

Finding a Subspace Trail. In order to prevent all invariant subspace trails with inactive F functions, it is thus sufficient to determine that the condition

$\mathfrak{I} = M \times \mathfrak{I}$ is never reached for a proper subspace $\mathfrak{I} \subseteq \mathfrak{X}^{(0)}$. This is the case if $\dim(\mathfrak{I} \cap (M \times \mathfrak{I})) \leq \dim(\mathfrak{I}) - 1$, which means that the dimension of $\mathfrak{I}$ exceeds the dimension of $\mathfrak{I} \cap (M \times \mathfrak{I})$. Further, note that an iterative subspace trail of the form $\{\mathfrak{I}, M \cdot \mathfrak{I}, M^2 \cdot \mathfrak{I}, \ldots, M^{l-1} \cdot \mathfrak{I}\}$ does not exist either in this case, since otherwise $\mathfrak{U} = M \times \mathfrak{U}$ for $\mathfrak{U} = \langle \mathfrak{I}, M \cdot \mathfrak{I}, M^2 \cdot \mathfrak{I}, \ldots, M^{l-1} \cdot \mathfrak{I} \rangle \subseteq \mathfrak{X}^{(0)} \implies \dim(\mathfrak{U}) < t$, where $l \in \mathbb{N}$ denotes the period of the iterative subspace trail.

Active F Function.

In order to provide security, we must also consider infinitely long subspace trails in which the input difference of the F function is nonzero. If such a subspace exists, it would be possible to skip the middle rounds without increasing the degree, which reduces the resistance against algebraic attacks.

To give a concrete example, consider again the case $t = 4$, where $\lambda_0 = \lambda_2 = 1$ and $\lambda_1 = \lambda_3 = -1$. Moreover, let

$$M_{\mathcal{J}} = \begin{pmatrix} \mu_0 & 1 & 1 & 1 \\ 1 & \mu_1 & 1 & 1 \\ 1 & 1 & \mu_1 & 1 \\ 1 & 1 & 1 & \mu_1 \end{pmatrix}$$

for $\mu_0, \mu_1 \in \mathbb{F}_p$. The two-dimensional subspace $\mathfrak{I} = \langle (1,0,0,0), (0,1,1,1) \rangle \subseteq \mathbb{F}_p^4$ generates an infinitely long invariant subspace trail in which F is active. Indeed, given $(x, y, y, y) \in \mathfrak{I}$, note that the input of F is $(x-y, x-y, x-y, x-y)$, which is in general not equal to zero.

In the following, we give a necessary condition that, if satisfied, guarantees that no infinitely long invariant subspace trail with active F functions exists.

Proposition 4. *If a subspace $\mathfrak{I} \subset \mathbb{F}_p^t$ defines an infinitely long invariant subspace trail with active F functions, then*

$$\langle (1, 1, \ldots, 1) \rangle \subseteq \mathfrak{I}.$$

Proof 4. *Let $\mathfrak{I}$ be an invariant subspace trail. For each $x \in \mathfrak{I}$,*

$$M \times (x_0 + F(x_0, \ldots, x_{t-1}), x_1 + F(x_0, \ldots, x_{t-1}), \ldots, x_{t-1} + F(x_0, \ldots, x_{t-1}))^T \in \mathfrak{I},$$

or equivalently $M \times x + F(x_0, \ldots, x_{t-1}) \cdot M \times (1, 1, \ldots, 1)^T \in \mathfrak{I}$, i.e.,

$$M \times \mathfrak{I} + \langle M \times (1, 1, \ldots, 1)^T \rangle = \mathfrak{I} \quad \implies \quad M \times (\mathfrak{I} + \langle (1, 1, \ldots, 1)^T \rangle) = \mathfrak{I}$$

where M is invertible. Since $\mathfrak{I}$ is an invariant subspace, we have that

$$\dim\left(M \times (\mathfrak{I} + \langle (1, 1, \ldots, 1)^T \rangle)\right) = \dim\left(\mathfrak{I} + \langle (1, 1, \ldots, 1)^T \rangle\right) = \dim(\mathfrak{I}),$$

and hence $\langle (1, 1, \ldots, 1)^T \rangle \subseteq \mathfrak{I}$. □

Finding a Subspace Trail. We can determine the existence of infinitely long subspace trails in the case of active F functions with a simple method. First, we start with the subspace $\mathfrak{I}_0 = \langle(1, \ldots, 1)\rangle$. If $\mathfrak{I}_0 = M \times \mathfrak{I}_0$, $\mathfrak{I}_0$ generates such a subspace trail and M is vulnerable. If $\mathfrak{I}_0 \neq M \times \mathfrak{I}_0$, we define $\mathfrak{I}_1 = \langle \mathfrak{I}_0, M \times \mathfrak{I}_0\rangle$ and again determine if $\mathfrak{I}_1 = M \times \mathfrak{I}_1$. Note that, since $\mathfrak{I}_0 \neq M \times \mathfrak{I}_0$ and M is nonzero, $\dim(\mathfrak{I}_1) > \dim(\mathfrak{I}_0)$. Eventually, we will either reach full dimension or an invariant subspace. If the largest possible dimension t is reached, no infinitely long invariant subspace trail for active F functions exists.

I Details about the Security Analysis of $\mathcal{H}_{\mathtt{K}}$

I.1 Maximum Differential Probability of a Generalized Lai–Massey Construction

Lemma 1. *Let $t \geq 2$. Let $\omega_0, \omega_1, \omega_2, \ldots, \omega_{t-1} \in \mathbb{F}_p \setminus \{0\}$ where $\sum_{h=0}^{t-1} \omega_h = 0$. Let $S^{''} : \mathbb{F}_p^t \to \mathbb{F}_p^t$ be defined as $S^{''}(x_0, x_1, \ldots, x_{t-1}) = y_0 \| y_1 \| \ldots \| y_{t-1}$ where*

$$\forall l \in \{0, 1, \ldots, t-1\} : \qquad y_l = x_l + \left(\sum_{h=0}^{t-1} \omega_h \cdot x_h\right)^2 .$$

Then, for each $\Delta_I, \Delta_O \in \mathbb{F}_p^t$

$$\mathrm{Prob}(\Delta_I \to \Delta_O) = \begin{cases} p^{-1} & \text{if} \quad \Delta_O[0] = \Delta_O[1] = \cdots = \Delta_O[t-1] (\neq 0) \\ 0 & \text{otherwise} \end{cases} .$$

Proof 5. *We count the number of solutions $x \in \mathbb{F}_p^t$ of $S^{''}(x + \Delta_I) - S^{''}(x) = \Delta_O$, that is, of*

$$\left(\sum_{h=0}^{t-1} \omega_h \cdot (x_h + \Delta_I[l])\right)^2 - \left(\sum_{h=0}^{t-1} \omega_h \cdot x_h\right)^2 = \Delta_O[l]$$

*for each $l \in \{0, 1, \ldots, t-1\}$. First of all, such system of equations admits solution(s) **only if** $\Delta_O[l] = \Delta_O[j]$ for each $l, j \in \{0, 1, \ldots, t-1\}$. Secondly, it is not hard to check that each one of the equations in the system is linear in $x_0, x_1, \ldots, x_{t-1}$. Hence, the number of solutions is at most p^{t-1} ($t-1$ variables are free to take any possible value, while the remaining one is fixed), which implies that Prob$(\Delta_I \to \Delta_O) \leq \frac{p^{t-1}}{p^t} = p^{-1}$.* □

We emphasize that the previous result can be easily generalized to the case in which the quadratic function is replaced by a generic degree-d function. In such a case, its maximum differential probability is $(d-1)/p$.

I.2 Details About Gröbner Basis Attacks Against $\mathcal{H}_{\mathtt{K}}$

This appendix focuses on Gröbner basis attacks against $\mathcal{H}_{\mathtt{K}}$. We start by briefly discussing the case where we only keep the 12 initial variables y, z and K, before

we consider in greater detail the case where the intermediate rounds of $\mathcal{H}_K$ are modeled with extra variables and equations. The final output of HYDRA is assumed to be known in all scenarios.

Polynomial System with Initial Variables

We make two observations on $\mathcal{H}_K(y, z)$, as described in Section 5.4. Firstly, the initial round $\mathcal{J}_0$ only affects y and z. Secondly, $\mathcal{J}_i$ includes only a single multiplication. Thus we can, after taking suitable linear combinations, write the output of $\mathcal{H}_K(y, z)$ as one polynomial in degree $2^{R_{\mathcal{JH}}}$, and seven polynomials of degree $2^{R_{\mathcal{JH}}-1}$. Upon generalizing this second point, we find that we can further write $\mathcal{H}_K(y, z)$ out as eight polynomials; one in degree $2^{R_{\mathcal{JH}}}$, one in degree $2^{R_{\mathcal{JH}}-1}$, and so on, all the way down to degree $2^{R_{\mathcal{JH}}-7}$. Moreover, this maximal degree will only be in the eight variables from y and z (the K-variables will only appear in monomials of smaller degree, due to the first observation). A similar statement holds for successive heads, $\mathcal{H}_K(\mathcal{R}_i(y, z))$, with each degree multiplied by a factor 2^i. Hence, while an attacker can use more output to sample more polynomials, the degree of these polynomials increases exponentially, thus limiting their use in the polynomial solving process. As noted in Section 7.3, we already require $R_{\mathcal{JH}} \geq 24$ (and we typically require an even greater number due to Eq. (7.12)). At these magnitudes, we find that modeling with only the initial variables will simply not be competitive when compared to other attack approaches.

Setup and Reduction of the Intermediate Variable Polynomial System

Each head $\mathcal{H}_K$ included in this modeling now increases the number of variables, so the attacker wants to include as few of them as possible. On the other hand, as $\mathcal{H}_K$ depends on the 12 unknown field elements comprising y, z and K, at least two heads are needed to construct an (over-)determined polynomial system. Thus, we choose to set up the system of equations using $\mathcal{H}_K(y, z)$ and $\mathcal{H}_K(\mathcal{R}_1(y, z))$.

Since there is only a single squaring repeated in $S_{\mathcal{J}}$, we can, after taking suitable linear combinations, model each intermediate round $\mathcal{J}_j$ with one quadratic equation and seven linear equations, in eight new variables. Similarly, the rolling function $\mathcal{R}_1$ can be modeled using two quadratic equations and six linear equations in eight new variables. The 16 output elements of $\mathcal{H}_K(y, z)$ and $\mathcal{H}_K(\mathcal{R}_1(y, z))$ are assumed known, and twelve variables are used for the input y, z and K. We use the linear equations in the system to eliminate variables. Further 16 variables can be eliminated using the known output. Assuming these relations are linear independent, we can reduce to a smaller system of $2r + 2$ quadratic equations in $2r - 2$ variables, where r is the number of rounds. This is indeed the case we have observed in all our experiments.

Practical Tests

We have tested the difficulty of computing a grevlex Gröbner basis for the reduced systems described above, with a varying number of rounds r, over $\mathbb{F}_{7741}$. The tests

Table 7.4: Running time (in seconds) and step degrees in the F_4 algorithm when solving polynomial systems from Hydra over $\mathbb{F}_{7741}$.

r	Step Degrees	Total Time	Time Max Step	Time Random
6	23$\underline{45}$	0.1	0.1	0.1
7	234$\underline{5}$55	0.9	0.4	0.5
8	2345$\underline{66}$	13.4	7.3	8.8
9	23456$\underline{7}$6	293.8	198.4	229.2
10	2345678$\underline{6}$	8799.5	6250.7	7740.8
11	23456$\underline{78}$88	229606.8	89768.6	209272.9

have been performed running the F_4 algorithm implemented in the computer algebra system MAGMA V2.22-6, on 72 x Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz, with 252 GiB RAM.

The results are presented in Table 7.4. 'Step Degrees' lists the degree of the polynomials associated with each step. The maximal step, i.e., the most costly step in terms of running time, is underlined. We find that all tested cases behave like quadratic semi-regular systems of $2r + 2$ polynomials in $2r - 2$ variables. In particular, the degree of the maximal step coincides with the degree of regularity (as described in Section 7.4) for such systems. We present the total running time of F_4, as well as the time spent at the maximal step, to give an impression of how the problem scales as r increases. For comparison, we also include the running time for random quadratic systems of $2r + 2$ polynomials in $2r - 2$ variables (i.e., the coefficients of all possible monomials up to degree two is a random element of $\mathbb{F}_p$), with a unique solution. Random systems are believed to be semi-regular with high probability (and they indeed turned out to be so in our tests), and the practical hardness of solving them has been studied in cryptographic solving challenges[1] [YDH+15]. In our tests, the step degrees for the random systems coincide with the associated systems from Hydra, yet the time for solving the former systems are consistently smaller.

Finally, we point out that the unique solution can be directly read from the grevlex Gröbner basis in all our experiments. Hence, there is no need for any further steps in the solving process.

Adding 2 Extra Rounds

Due to the above results, we conclude that the Gröbner basis attack against $\mathcal{H}_K$ is comparable to that of of solving a semi-regular system with $2r + 2$ quadratic polynomials in $2r - 2$ variables. However, we note that not all polynomials in our modeling will contain every variable, which adds a certain structure that an attacker might be able to use when reducing the matrices encountered in F_4. To account for this property, we add two extra rounds on top of this baseline (in addition to the conservative choice of using $\omega = 2$ for the linear algebra constant).

[1] https://www.mqchallenge.org/

J Full MPC Benchmarks

In Table 7.5 and Table 7.6, we give the full MPC benchmarks. Compared to Table 7.1, we additionally compare Hydra to $\text{GMiMC}_{\text{erf}}$ and MiMC-CTR and give benchmarks for more state sizes t. Looking at MiMC, one can observe, that it has a fast online phase performance, but it requires significantly more data transmission between the parties. Furthermore, its large number of multiplications also leads to an expensive offline phase. GMiMC, on the other hand, has a decent performance for very small t, but its number of rounds does not scale well with larger state sizes. Consequently, our benchmark show that it is somewhat competitive for $t = 8$, but its runtime and data transmission explodes with larger t. In any case, Hydra has the faster offline phase performance and also a comparable online phase performance with less data communication compared to MiMC and GMiMC, with the advantage of Hydra growing with the state size t.

K Effect of the linear layer

In Section 8, we noted that HadesMiMC has a slow online phase performance when evaluated in a LAN setting due to having many MDS matrix multiplications with bad plain performance. To highlight this effect, we compare Hydra and HadesMiMC (with and without its linear key schedule) to a (insecure) version of HadesMiMC (dubbed HadesMiMC-circ) for which we replaced the random MDS matrices with the matrix $M = \text{circ}(2, 1, 1, \ldots, 1)$. The corresponding matrix-vector multiplications can purely be implemented using additions. In Table 7.7 one can observe that this change, as expected, has no effect on the offline phase performance and the communication between the parties. However, the runtime of the online phase became significantly faster and is now twice as fast as the online phase of Hydra. However, combining both phases, one can observe that Hydra still leads to faster runtimes and less communication due to overall having less multiplications, with larger state sizes further increasing the advantage of Hydra.

Table 7.5: 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. **Bold** values are best values with key schedules, *Italic* the best without.

Cipher	Rounds	Offline Prec.	Offline Time ms	Offline Data MB	Online Depth	Online Time ms	Online Data kB	Combined Time ms	Combined Data MB
$t = 8$:									
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*
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 (No KS)[a]	10	480	129.47	10.83	*33*	6.95	11.80	136.42	10.84
Rescue	10	960	254.80	21.65	***33***	12.65	23.32	267.45	21.68
GMiMC	177	354	92.78	7.99	179	**5.96**	8.78	98.74	8.00
MiMC	81	1296	372.63	29.23	83	6.28	31.38	378.91	29.26
$t = 16$:									
Hydra	6, 42, 39	**212**	**52.72**	**4.78**	132	9.87	***6.81***	**62.59**	**4.79**
Ciminion (No KS)[a]	90, 14	*208*	*50.51*	*4.69*	111	*5.22*	7.06	*60.74*	*4.70*
Ciminion	90, 14	1647	458.36	37.13	1455	49.75	53.11	508.11	37.19
HadesMiMC	6, 71	334	84.86	7.54	79	69.08	8.42	153.95	7.55
Rescue (No KS)[a]	10	960	263.58	21.65	*33*	18.80	23.45	282.38	21.68
Rescue	10	1920	531.54	43.30	***33***	38.68	46.49	570.21	43.35
GMiMC	546	1092	292.14	24.63	548	15.81	26.62	307.96	24.66
MiMC	81	2592	770.12	58.46	83	**6.21**	62.62	776.33	58.52
$t = 32$:									
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
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 (No KS)[a]	10	1920	538.19	43.30	*33*	47.35	46.74	585.54	43.35
Rescue	10	3840	1253.76	86.60	***33***	109.80	92.82	1363.56	86.70
GMiMC	2114	4228	1424.02	95.35	2116	57.97	102.14	1481.99	95.46
MiMC	81	5184	1713.06	116.91	83	13.77	125.08	1726.83	117.04

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

Table 7.6: 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. **Bold** values are best values with key schedules, *Italic* the best without.

Cipher	Rounds	Offline Prec.	Offline Time ms	Offline Data MB	Online Depth	Online Time ms	Online Data kB	Combined Time ms	Combined Data MB
$t = 64$:									
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
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 (No KS)[a]	10	3840	1226.39	86.60	*33*	144.14	93.34	1370.53	86.70
Rescue	10	7680	2851.56	173.20	***33***	402.34	185.50	3253.90	173.39
GMiMC	8322	16644	6253.29	375.36	8324	255.78	400.63	6509.07	375.76
MiMC	81	10368	4060.68	233.82	83	23.71	250.01	4084.39	234.07
$t = 96$:									
Hydra	6, 42, 39	***622***	***172.42***	***14.03***	142	33.99	***21.21***	***206.40***	***14.05***
Ciminion (No KS)[a]	90, 14	808	213.55	18.22	151	*9.76*	27.54	223.31	18.25
Ciminion	90, 14	9447	3693.60	212.98	8655	257.04	303.99	3950.64	213.28
HadesMiMC	6, 71	1294	344.34	29.19	79	2114.35	32.74	2458.69	29.22
Rescue (No KS)[a]	10	5760	2011.83	129.90	*33*	303.15	139.93	2314.98	130.04
Rescue	10	11520	4457.93	259.80	***33***	803.78	278.17	5261.71	260.08
GMiMC	18626	37252	12907.70	840.11	18628	547.09	895.74	13454.79	841.01
MiMC	81	15552	5771.92	350.73	83	**27.49**	374.94	5799.41	351.11
$t = 128$:									
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
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 (No KS)[a]	10	7680	2943.21	173.20	*33*	737.84	186.52	3681.05	173.39
Rescue	10	15360	5934.39	346.40	***33***	1549.16	370.84	7483.55	346.77
GMiMC	33026	66052	23295.70	1489.61	33028	955.50	1587.45	24251.20	1491.20
MiMC	81	20736	7588.20	467.64	83	**35.43**	499.86	7623.63	468.14

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

Table 7.7: 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. **Bold** values are best values with key schedules, *Italic* the best without.

Cipher	Rounds	Offline Prec.	Offline Time ms	Offline Data MB	Online Depth	Online Time ms	Online Data kB	Combined Time ms	Combined Data MB
$t = 8$:									
Hydra	6, 42, 39	***171***	***39.99***	***3.86***	131	6.81	***5.37***	***46.80***	***3.87***
HadesMiMC (No KS)[a]	6, 71	238	51.49	5.37	*79*	10.67	5.99	62.16	5.38
HadesMiMC	6, 71	238	52.66	5.37	***79***	17.58	5.99	70.24	5.38
HadesMiMC-circ (No KS)[a]	6, 71	238	51.06	5.37	*79*	*3.57*	5.99	54.62	5.38
HadesMiMC-circ	6, 71	238	55.10	5.37	***79***	**3.82**	5.99	58.92	5.38
$t = 16$:									
Hydra	6, 42, 39	***212***	***52.72***	***4.78***	132	9.87	***6.81***	***62.59***	***4.79***
HadesMiMC (No KS)[a]	6, 71	334	88.46	7.54	*79*	41.36	8.42	129.82	7.55
HadesMiMC	6, 71	334	84.86	7.54	***79***	69.08	8.42	153.95	7.55
HadesMiMC-circ (No KS)[a]	6, 71	334	89.95	7.54	*79*	*5.84*	8.42	105.99	7.55
HadesMiMC-circ	6, 71	334	83.93	7.54	***79***	**6.06**	8.42	99.78	7.55
$t = 32$:									
Hydra	6, 42, 39	***294***	***72.67***	***6.63***	134	13.36	***9.69***	***86.03***	***6.64***
HadesMiMC (No KS)[a]	6, 71	526	138.03	11.87	*79*	128.36	13.29	266.38	11.88
HadesMiMC	6, 71	526	137.49	11.87	***79***	225.86	13.29	363.35	11.88
HadesMiMC-circ (No KS)[a]	6, 71	526	137.72	11.87	*79*	*6.14*	13.29	144.86	11.88
HadesMiMC-circ	6, 71	526	138.15	11.87	***79***	**6.70**	13.29	144.85	11.88
$t = 64$:									
Hydra	6, 42, 39	***458***	***119.07***	***10.33***	138	20.57	***15.45***	***139.64***	***10.35***
HadesMiMC (No KS)[a]	6, 71	910	242.49	20.53	*79*	411.90	23.02	654.39	20.55
HadesMiMC	6, 71	910	251.44	20.53	***79***	899.55	23.02	1150.99	20.55
HadesMiMC-circ (No KS)[a]	6, 71	910	242.01	20.53	*79*	*11.40*	23.02	253.51	20.55
HadesMiMC-circ	6, 71	910	244.90	20.53	***79***	**11.53**	23.02	256.33	20.55
$t = 128$:									
Hydra	6, 42, 39	***786***	***206.08***	***17.72***	146	37.49	***26.97***	***243.58***	***17.75***
HadesMiMC (No KS)[a]	6, 71	1678	456.94	37.85	*79*	1864.89	42.47	2321.83	37.89
HadesMiMC	6, 71	1678	463.59	37.85	***79***	4371.02	42.47	4834.61	37.89
HadesMiMC-circ (No KS)[a]	6, 71	1678	477.78	37.85	*79*	*21.48*	42.47	505.26	37.89
HadesMiMC-circ	6, 71	1678	463.89	37.85	***79***	**21.83**	42.47	485.72	37.89

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

References

[AABS+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols." In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 1–45.

[ADS+21] Damiano Abram, Ivan Damgård, Peter Scholl, and Sven Trieflinger. "Oblivious TLS via Multi-party Computation." In: *CT-RSA*. Vol. 12704. LNCS. 2021, pp. 51–74.

[AGP+19] Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. "Feistel Structures for MPC, and More." In: *ESORICS*. Vol. 11736. LNCS. 2019, pp. 151–171.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity." In: *ASIACRYPT*. Vol. 10031. LNCS. 2016, pp. 191–219.

[ALP+19] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. "Forkcipher: A New Primitive for Authenticated Encryption of Very Short Messages." In: *ASIACRYPT*. Vol. 11922. LNCS. 2019, pp. 153–182.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. "Ciphers for MPC and FHE." In: *EUROCRYPT*. Vol. 9056. LNCS. 2015, pp. 430–454.

[BBL+22] Augustin Bariant, Clémence Bouvier, Gaëtan Leurent, and Léo Perrin. "Algebraic Attacks against Some Arithmetization-Oriented Primitives." In: *IACR Trans. Symmetric Cryptol.* 2022.3 (2022), 73–101.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. "Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems." In: *CRYPTO*. Vol. 12172. LNCS. 2020, pp. 299–328.

[BDH+17] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. "Farfalle: parallel permutation-based cryptography." In: *IACR Trans. Symmetric Cryptol.* 2017.4 (2017), pp. 1–38.

[BDJ+06] Peter Bogetoft, Ivan Damgård, Thomas P. Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. "A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation." In: *Financial Cryptography*. Vol. 4107. LNCS. 2006, pp. 142–147.

[BDK+21] Nicolas Bordes, Joan Daemen, Daniël Kuijsters, and Gilles Van Assche. "Thinking Outside the Superbox." In: *CRYPTO*. Vol. 12827. LNCS. 2021, pp. 337–367.

[BDP+08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction." In: *EUROCRYPT*. Ed. by Nigel P. Smart. Vol. 4965. LNCS. 2008, pp. 181–197.

[BDP+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *The KECCAK reference*. https://keccak.team/files/Keccak-reference-3.0.pdf. 2011.

[Bea91] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization." In: *CRYPTO*. Vol. 576. LNCS. 1991, pp. 420–432.

[BFS+05] Magali Bardet, Jean-Charles Faugére, Bruno Salvy, and Bo-Yin Yang. "Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems." In: *Proc. of MEGA*. Vol. 5. 2005.

[BS90] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems." In: *CRYPTO*. Vol. 537. LNCS. 1990, pp. 2–21.

[BS93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[BW99] Alex Biryukov and David A. Wagner. "Slide Attacks." In: *Fast Software Encryption – FSE 1999*. Vol. 1636. LNCS. 1999, pp. 245–259.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. "Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives." In: *CCS*. ACM, 2017, pp. 1825–1842.

[CFG+18] Colin Chaigneau, Thomas Fuhr, Henri Gilbert, Jian Guo, Jérémy Jean, Jean-René Reinhard, and Ling Song. "Key-Recovery Attacks on Full Kravatte." In: *IACR Trans. Symmetric Cryptol.* 2018.1 (2018), pp. 5–28.

[CG20] Tingting Cui and Lorenzo Grassi. "Algebraic Key-Recovery Attacks on Reduced-Round Xoofff." In: *SAC*. Vol. 12804. LNCS. 2020, pp. 171–197.

[CGG+22] Carlos Cid, Lorenzo Grassi, Aldo Gunsing, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. "Influence of the Linear Layer on the Algebraic Degree in SP-Networks." In: *IACR Trans. Symmetric Cryptol.* 2022.1 (2022), pp. 110–137.

[CKP+00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. "Efficient algorithms for solving overdefined systems of multivariate polynomial equations." In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, pp. 392–407.

[CLO13] David Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms: an introduction to computational algebraic geometry and commutative algebra*. Springer Science & Business Media, 2013.

[Dae91] Joan Daemen. "Limitations of the Even-Mansour Construction." In: *ASIACRYPT*. Vol. 739. LNCS. 1991, pp. 495–498.

[DGG+21] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. "Ciminion: Symmetric Encryption Based on Toffoli-Gates over Large Finite Fields." In: *EUROCRYPT*. Vol. 12697. LNCS. 2021, pp. 3–34.

[DGH+21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. "MPC-Friendly Symmetric Cryptography from Alternating Moduli: Candidates, Protocols, and Applications." In: *CRYPTO*. Vol. 12828. LNCS. 2021, pp. 517–547.

[DHA+18] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. "The design of Xoodoo and Xoofff." In: *IACR Trans. Symmetric Cryptol.* 2018.4 (2018), pp. 1–38.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits." In: *ESORICS*. Vol. 8134. LNCS. 2013, pp. 1–18.

[DKR+21] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. "Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto." In: *IACR Cryptol. ePrint Arch.* (2021). Accepted at ACM CCS 2022.

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. "Minimalism in Cryptography: The Even-Mansour Scheme Revisited." In: *EUROCRYPT*. Vol. 7237. LNCS. 2012, pp. 336–354.

[DPS+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 643–662.

[DPS20] Jintai Ding, Albrecht Petzoldt, and Dieter S Schmidt. *Multivariate Public Key Cryptosystems*. https://doi.org/10.1007/978-1-0716-0987-3. Springer, 2020.

[DR01] Joan Daemen and Vincent Rijmen. "The Wide Trail Design Strategy." In: *Cryptography and Coding - IMA International Conference 2001*. Vol. 2260. LNCS. 2001, pp. 222–238.

[EM91] Shimon Even and Yishay Mansour. "A Construction of a Cipher From a Single Pseudorandom Permutation." In: *ASIACRYPT*. Vol. 739. LNCS. 1991, pp. 210–224.

[Fau99] Jean-Charles Faugére. "A new efficient algorithm for computing Gröbner bases (F_4)." In: *Journal of pure and applied algebra* 139.1-3 (1999), pp. 61–88.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems." In: *USENIX Security Symposium*. USENIX Association, 2021.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. "On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy." In: *EUROCRYPT*. Vol. 12106. LNCS. 2020, pp. 674–704.

[GOP+22] Lorenzo Grassi, Silvia Onofri, Marco Pedicini, and Luca Sozzi. "Invertible Quadratic Non-Linear Layers for MPC-/FHE-/ZK-Friendly Schemes over $\mathbb{F}_p^n$: Application to Poseidon." In: *IACR Trans. Symmetric Cryptol.* 2022.3 (2022), pp. 20–72.

[Gra22] Lorenzo Grassi. "On Generalizations of the Lai-Massey Scheme: the Blooming of Amaryllises." In: *IACR Cryptol. ePrint Arch.* (2022), p. 1245.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. "MPC-Friendly Symmetric Key Primitives." In: *CCS*. ACM, 2016, pp. 430–443.

[GRR17] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. "Subspace Trail Cryptanalysis and its Applications to AES." In: *IACR Trans. Symmetric Cryptol.* 2016.2 (2017), pp. 192–225.

[GRS21] Lorenzo Grassi, Christian Rechberger, and Markus Schofnegger. "Proving Resistance Against Infinitely Long Subspace Trails: How to Choose the Linear Layer." In: *IACR Trans. Symmetric Cryptol.* 2021.2 (2021), pp. 314–352.

[GSW+21] Chun Guo, François-Xavier Standaert, Weijia Wang, Xiao Wang, and Yu Yu. "Provable Security of SP Networks with Partial Non-Linear Layers." In: *IACR Transactions on Symmetric Cryptology* 2021.2 (2021), 353–388.

[HKR+21] Lukas Helminger, Daniel Kales, Sebastian Ramacher, and Roman Walch. "Multi-party Revocation in Sovrin: Performance through Distributed Trust." In: *CT-RSA*. Vol. 12704. LNCS. Springer, 2021, pp. 527–551.

[IKO+07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Zero-knowledge from secure multiparty computation." In: *STOC*. ACM, 2007, pp. 21–30.

[JK97] Thomas Jakobsen and Lars R. Knudsen. "The Interpolation Attack on Block Ciphers." In: *FSE*. Vol. 1267. LNCS. 1997, pp. 28–40.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation." In: *CCS*. ACM, 2020, pp. 1575–1590.

[Knu94] Lars R. Knudsen. "Truncated and Higher Order Differentials." In: *FSE*. Vol. 1008. LNCS. 1994, pp. 196–211.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer." In: *CCS*. ACM, 2016, pp. 830–842.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. "Mobile Private Contact Discovery at Scale." In: *USENIX Security Symposium*. USENIX Association, 2019, pp. 1447–1464.

[LAA+11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. "A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack." In: *CRYPTO*. Vol. 6841. LNCS. 2011, pp. 206–221.

[Lai94] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis." In: *Communications and Cryptography: Two Sides of One Tapestry*. Springer US, 1994, pp. 227–233.

[LM90] Xuejia Lai and James L. Massey. "A Proposal for a New Block Encryption Standard." In: *EUROCRYPT*. Vol. 473. LNCS. 1990, pp. 389–404.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. "A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro." In: *EUROCRYPT*. Vol. 9056. LNCS. 2015, pp. 254–283.

[LTW13] Sven Laur, Riivo Talviste, and Jan Willemson. "From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting." In: *ACNS*. Vol. 7954. LNCS. 2013, pp. 84–101.

[MN17] Bart Mennink and Samuel Neves. "Optimal PRFs from Blockcipher Designs." In: *IACR Trans. Symmetric Cryptol.* 2017.3 (2017), 228–252.

[MZ17] Payman Mohassel and Yupeng Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning." In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017, pp. 19–38.

[Sha79] Adi Shamir. “How to Share a Secret.” In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[Sil00] John R Silvester. “Determinants of block matrices.” In: *The Mathematical Gazette* 84.501 (2000), pp. 460–467.

[Vau99] Serge Vaudenay. “On the Lai-Massey Scheme.” In: *ASIACRYPT*. Vol. 1716. LNCS. 1999, pp. 8–19.

[YDH+15] Takanori Yasuda, Xavier Dahan, Yun-Ju Huang, Tsuyoshi Takagi, and Kouichi Sakurai. “MQ challenge: hardness evaluation of solving multivariate quadratic problems.” In: (2015).

8

Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations

Publication Data

Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations." In: *IACR Cryptol. ePrint Arch.* (2023), p. 1025. Preprint

The appended paper is a preprint available at https://eprint.iacr.org/2023/1025 which has been changed to use the design and formatting of this thesis.

Contributions

Main author.

Monolith: Circuit-Friendly Hash Functions with New Nonlinear Layers for Fast and Constant-Time Implementations

Lorenzo Grassi[2,5], Dmitry Khovratovich[3,8], Reinhard Lüftenegger[1], Christian Rechberger[1], Markus Schofnegger[4], Roman Walch[1,6,7]

[1] Graz University of Technology (Austria)
[2] Ponos Technology (Switzerland)
[3] Ethereum Foundation (Luxembourg)
[4] Horizen Labs (United States)
[5] Ruhr University Bochum (Germany)
[6] Know-Center (Austria)
[7] TACEO (Austria)
[8] ABDK Consulting (Estonia)

Abstract

Hash functions are a crucial component in incrementally verifiable computation (IVC) protocols and applications. Among those, recursive SNARKs and folding schemes require hash functions to be both fast in native CPU computations and compact in algebraic descriptions (constraints). However, neither SHA-2/3 nor newer algebraic constructions, such as Poseidon, achieve both requirements.

In this work we overcome this problem in two steps. First, for certain prime field domains we propose a new design strategy called Kintsugi, which explains how to construct nonlinear layers of high algebraic degree which allow fast native implementations and at the same time also an efficient circuit description for zero-knowledge applications. Then we suggest another layer, based on the Feistel Type-3 scheme, and prove wide trail bounds for its combination with an MDS matrix.

Finally, we propose a new permutation design named Monolith to be used as a sponge or compression function. It is the first arithmetization-oriented function with a native performance comparable to SHA3-256. At the same time, it outperforms Poseidon in a circuit using the Merkle tree prover in the Plonky2 framework. Contrary to previously proposed designs, Monolith also allows for efficient constant-time native implementations which mitigates the risk of side-channel attacks.

1 Introduction

1.1 Hash Functions in Zero-Knowledge Frameworks

Zero-knowledge use cases and particularly the area of computational integrity combined with zero knowledge have seen a rise in popularity in the last couple of years. Many new protocols [GWC19; ZGK+22; KST22; BC23] and low-level

primitives [AGR+16; AAE+20; GKR+21] have been designed and published recently, in an attempt to increase the performance in this setting. The emergence of folding techniques and recursive SNARKs (incrementally verifiable computation, or IVC [Val08]) make it possible to efficiently prove the integrity of complex computations. Proofs with 2^{27} steps have been recorded[1] whereas SNARK-based verifiable delay functions (VDFs) might require proving up to 2^{40} operations [KMT22]. A single IVC operation is typically a compact arithmetic computation (polynomial) in a certain prime field or an assertion to some low-degree polynomial predicate. With VC programs (also called circuits) being that large and containing cryptographic protocols, more and more programs contain hash functions as subroutines. Hash functions and their underlying permutations are used not only for data integrity checks, but also to instantiate commitment schemes, authenticated encryption [PSS19; CFG+22], non-interactive proofs based on the Fiat–Shamir transform, and many other techniques.

Hash Functions in IVC Applications.

For "classical" applications of hash functions, general-purpose standard choices like SHA-2 or SHA-3 are usually not the bottleneck.[2] However, the situation is different in the IVC applications mentioned above. For hashing and membership proofs in ZK, with folding schemes [KST22; KS23; BC23] and private mixers like Tornado being an example [PSS19], the size of hash function as an *arithmetic circuit over a prime field* is more important as a cost metric than the "native" software performance (e.g., on a x86 architecture). New hash functions have tried to bridge this gap [AGR+16; AAE+20; GHR+23; GKR+21; SAD20; BBC+23].

Another example is using hash functions as a commitment tool in IVC frameworks where the underlying commitment scheme may not be homomorphic – with STARKs being a notable example [BBH+19]. With a prover and a verifier engaging in commit-open protocols (again, over certain prime fields), this use case requires to efficiently construct an entire Merkle tree in a prime field domain over large amounts of data. So far, though, the computations were performed natively on x86 hardware and not (yet) inside a circuit. Here, classical hash functions have been used up until recently.

Both cases appear in recursive schemes, in particular in recursive STARKs [COS20], which are an attractive IVC concept due to relatively little overhead and the possibility of parallelism for large or long computations. These schemes are used in an increasing number of applications, including zero-knowledge virtual machines [Monb; Mona; Zha22] and decentralized signature aggregation [But22] protocols as notable examples. In recursive STARKs the computation and its proof are broken into chunks $C_1, C_2, \ldots, C_k$ such that the proof π_i certifies that chunks from C_1 to C_i are computed correctly using the previous proof π_{i-1} and a proof of computing C_i. On each recursion step a prover computes a Merkle tree over the witness data and then proves some tree openings in a circuit. Thus, the *same* hash function is used in the circuit and in the native computation. In

[1] https://research.protocol.ai/sites/snarks/

[2] Newer choices that are faster by a small factor do exist [ANW+13; BDP+18].

this scenario, up to 90% of a prover's computation may be spent on the hash function call and proofs [COS20; (RI23], and a construction of a function that excels in both areas is a crucial open problem.

Relevant IVC Techniques: Lookups and Small Domains.

Two major developments in IVC have helped us in this work. The first one is the lookup technique. Starting with Plookup, the IVC operations include not only arithmetic expressions and predicates but also lookup statements of form $a \in T$, where T is a table available to the verifier [GW20; PH23; STW23]. Depending on the polynomial commitment used within IVC, the table may be preprocessed [ZBK+22; ZGK+22; EFG22], so that in the former case only the number of lookups contribute to the prover cost, whereas in the latter case the table size itself is the minimal cost. STARKs use non-homomorphic FRI commitments and thus belong to the second group. The lookup technique not only reduced the cost of traditional hash functions in circuits[1] but also allowed for cheap transformations of high algebraic degree [GKL+22; SLS+23].

The second improvement is purely technical but nevertheless vital for the performance. It consists of using small prime fields of ≤ 64 bits with primes of special forms like $2^k - 1$ [mon22; mon23; RIS23], which gained special attention for high performance of arithmetic operations in the field. STARKs [BBH+19] can use them since they do not require a group where the discrete logarithm problem is assumed to be hard. The performance growth is significant: Switching to an efficient 64-bit field improves the performance by a factor of up to 10 for the Poseidon hash function [GKS23]. An important feature of these domains is that the modular reduction can be implemented with mere additions and bit shifts, which are vectorizable on modern CPU architectures and are much faster than their counterparts in large prime fields. There are also various works in the recent literature discussing smaller primes for IVC applications [HLN23; Hab23].

1.2 Our Contributions

We approach the problem of creating a fast and circuit-friendly hash function in several steps. First we summarize the technical ideas of the new design, and then we introduce the construction of the new hash function Monolith.

Efficient Nonlinearity and Compact Circuits over Prime Fields.

Our first main contribution is a generic design of components over certain prime fields $\mathbb{F}_p$, which, on one hand, can be implemented with just a few (and possibly vector) constant-time instructions on the x86 architecture, and on another hand can be written as a small circuit over $\mathbb{F}_p$. This strategy, which we call Kintsugi, is an evolution of the ideas behind the Reinforced Concrete [GKL+22] and Tip5 [SLS+23] components. First, an element from a bigger field is efficiently split into smaller bitarrays, which is possible due to the form of the prime. Then

[1] https://zcash.github.io/halo2/design/gadgets/sha256/table16.html

we apply *constant-time S-boxes*, which are instantiated by Daemen's χ function and similar ones [Dae95] that can be implemented in a batch using fast vector instructions, or as lookup tables in circuits. Finally, the outputs are assembled back to a field element with no overflow or collision, which is asserted in circuits with minimal overhead.

Low-Degree Components with Provable Differential Bounds.

Our second contribution is a concept of using a Feistel Type-3 [ZMI89] function together with an MDS layer. It is offered as a replacement to the power function x^d in Poseidon [GKR+21] and similar constructions. The advantage is that we can use faster squarings x^2 instead of more expensive (as d must be coprime with $p-1$) power functions over $\mathbb{F}_p$, and simultaneously obtain predicates of low degree in circuits. Whereas the Feistel layer alone is known to have weak diffusion, we show that together with an MDS it comes close to a regular SPN.

To the best of our knowledge, we are the first to prove the results on the differential properties of the component using a strategy analogous to the wide trail design [DR02]. In particular, we prove lower bounds on the number of active nonlinear functions in trails. Similar to extended generalized Feistel networks introduced in [BMT13], we believe that this result and its possible extension to Feistel structures of other types may be useful in the design of any symmetric primitive, not only for arithmetization-friendly schemes, but also in the case of more classical use cases (as already happened for the Lilliput cipher [BFM+16]).

Monolith: Fast, Constant-Time, ZK-Oriented Hashing.

The combination of such techniques leads us to the design of Monolith[1], a family of permutations which are both efficient in native and inside of circuits and can be turned into hash functions and other permutation-based schemes.

Construction of Monolith. Our scheme consists of a few rounds, each using the following three components.

- The first one is Bricks (Section 4.3), which is instantiated with a Feistel Type-3 construction with square mappings.
- The second component is Concrete (Section 4.4), which is the multiplication with a circulant MDS matrix. Together with Bricks it provides the diffusion necessary to protect against statistical attacks.
- Finally, the third component is Bars (Section 4.5), which is based on the Kintsugi outlined above. We prove that each such Bar operation has a high degree and provides high security against algebraic attacks. The Bar function is applied only to a few field elements in each round.

[1] A *monolithic* building is a seamless structure where components are intimately fused in order to provide the most secure and robust construction.

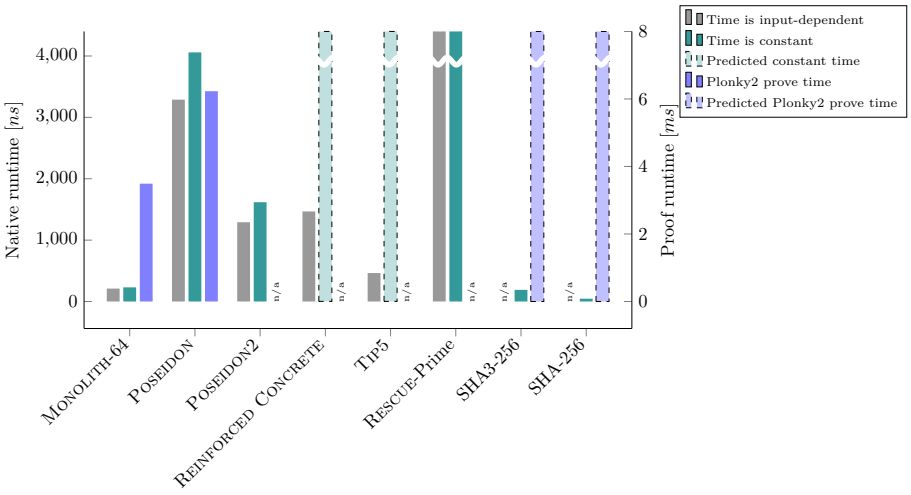


Figure 8.1: Runtime comparison of different hash functions. The benchmarks are from Table 8.3 and the numbers for Monolith-64, Poseidon, and Poseidon2 are taken for the 64-bit prime field and a state size of $t = 12$ (sponge mode). Proof times are benchmarks for a proof of preimage knowledge (Table 8.5). Numbers for SHA3-256 and SHA-256 are extrapolated from a `circom` implementation using R1CS [Bal23].

The combination of these three components provides security against statistical and algebraic attacks while allowing for an efficient implementation. Our initial analysis has found a 3-round attack on a weakened version, and also suggests that all potential attacks should stop at at most 4 rounds. Since improvements are expected (we encourage third-party cryptanalysis), we set the number of rounds uniformly to 6. Details can be found in Section 5.

Performance Evaluation. We give an extensive comparison between our new proposal and its competitors in Section 6. Our benchmarks confirm that the native performance of Monolith is comparable to SHA-3, which makes it the first circuit-friendly compression function achieving this goal. At the same time, Monolith is efficient within IVC systems. In contrast to Reinforced Concrete and Tip5, Monolith also has the crucial advantage that it allows for a constant-time implementation without significant performance losses, and it can also be reasonably used in proof systems without lookup arguments. A quick overview of the performance numbers is given in Fig. 8.1. We focus on the Plonky2 proving system since it is currently one of the most popular ones for FRI-based proofs.

Further, compared to Tip5, Monolith is around twice as fast and gives the user more freedom regarding the choice of the prime number. It can even be used with prime fields as low as 31 bits, which is a setting recently considered in applications and various proving frameworks [RIS23] due to advantageous

implementation characteristics. Moreover, compared to the widely used Poseidon permutation, Monolith shows a native performance improvement by a factor of around 15. Finally, Monolith allows for an efficient circuit implementation, since it can be represented by a low number of degree-2 constraints, leading to a faster prover and verifier performance compared to Poseidon when implemented in the FRI-based Plonky2 proof system [mon22] (see Table 8.5).

2 Fast and Circuit-Friendly Functions over $\mathbb{F}_p$

We suggest a generic strategy to create nonlinear components over $\mathbb{F}_p$ that are efficient in native, constant-time, and circuit implementations.

2.1 The Kintsugi Design Strategy

When wormonolith:king over $\mathbb{F}_p$, informally, we cannot just split a field element into smaller chunks, process them independently, and then reassemble. This is due to the fact that the field size is a prime and thus cannot be represented as a product of smaller domains.

To solve this problem, we present a generic strategy for specifically chosen prime numbers. Elements of it can be found in earlier works on Reinforced Concrete [GKL+22] and Tip5 [SLS+23]. The main principles are the following.

1. Assume we work in a prime field where the prime is a sum of just a few (possibly negative) powers of two, such as $p_{\texttt{gen2}} = 2^n - 1$ or $p_{\texttt{gen1}} = 2^n - 2^\eta + 1$.

2. Split the integer form of a field element into chunks according to carefully chosen boundaries aligned with these powers of two (more details to follow) and such that the resulting (smaller) chunks fit a lookup table in a ZK circuit.

3. Identify the combination of chunk values that never appear due to the fact that p is not a power of two.

4. Design intra-chunk transformations $\mathcal{S}_i$ such that

   - impossible chunk combinations never appear (this is, e.g., done by mamonolith:king some chunk values fixed points), and
   - they can be implemented in constant time, for example with an AndRX (AND-rotation-XOR) transformation [AJN14].

5. Combine the chunks back into a large element, after a possible shuffle (note that only the shuffles that guarantee that the output element is in the field are possible).

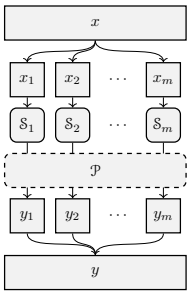


Figure 8.2: The KINTSUGI strategy, where $\mathcal{S}_i(2^{s_i-1}) = 2^{s_i-1}$, $\mathcal{S}_i(\mathbf{0}^{s_i}) = \mathbf{0}^{s_i}$ if p is of the form $p_{\texttt{gen1}}$, and $\mathcal{P}$ denotes a potential shuffling operation applied to the vector $(\mathcal{S}_1(x_1), \mathcal{S}_2(x_2), \ldots, \mathcal{S}_m(x_m))$.

We call this strategy KINTSUGI.[1] An illustration is shown in Fig. 8.2. More formally, it can be defined as

$$x \mapsto \mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(x). \tag{8.1}$$

with the following components.

Decomposition $\mathcal{D}$.

The decomposition $\mathcal{D}$ decomposes the original field element $x \in \mathbb{F}_p$ into $m > 1$ smaller elements $x'_1, x'_2, \ldots, x'_m$, such that

$$x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i-1} s_j} \cdot x'_i$$

over integers, where $x'_i \in \mathbb{Z}_{2^{s_i}} \equiv \mathbb{F}_{2^{s_i}}$ and $\sum s_i = n = \lceil \log_2(p) \rceil$, i.e., we apply binary decomposition. If $p = p_{\texttt{gen1}}$, we additionally require that $s_1 + s_2 + \cdots + s_l = \eta$ for some l. Equivalently, $x_i := (x \gg (s_1 + s_2 + \cdots + s_{i-1})) \odot (2^{s_i} - 1)$.

S-Boxes $\mathcal{S}$.

The operation $\mathcal{S}$ is the parallel application of m S-boxes, i.e.,

$$\mathcal{S}(x'_1, x'_2, \ldots, x'_m) = \mathcal{S}_1(x'_1) \mid\mid \mathcal{S}_2(x'_2) \mid\mid \cdots \mid\mid \mathcal{S}_m(x'_m), \tag{8.2}$$

where $\mathcal{S}_i : \mathbb{F}_2^{s_i} \to \mathbb{F}_2^{s_i}$. We additionally require certain fixed points. If p is of the form $p_{\texttt{gen2}}$ or $p_{\texttt{gen1}}$, then $\mathcal{S}_i(\mathbf{1}^{s_i} = 2^{s_i} - 1) = \mathbf{1}^{s_i}$. If p is of the form $p_{\texttt{gen1}}$, then also $\mathcal{S}_i(\mathbf{0}^{s_i}) = \mathbf{0}^{s_i}$.

Almost any invertible AndRX transformation works well for $\mathcal{S}$ and can be implemented in constant time as its components are basic x86 operations. Here we limit ourselves to give some concrete examples for the case $p_{\texttt{gen2}} = 2^n - 1$.

[1] Kintsugi is the Japanese art of repairing broken pottery by mending the areas of breakage with lacquer dusted or mixed with e.g. powdered gold. Here, we break the state and we recombine it after applying a particular function to each small chunk.

- *Bit Shuffle.* Clearly, both $\mathbf{1}^s$ and $\mathbf{0}^s$ are fixed points under the bit shuffling operation. Moreover, it is essentially for free in hardware.
- *Efficient Linear Operations.* Linear operations over $\mathbb{F}_2^n$ of the form

$$x \mapsto x \oplus (x \lll i) \oplus (x \lll j)$$

  with non-null $i \neq j$, and where $\lll$ denotes the circular shift operation, are (i) invertible for odd s and (ii) result in $\mathbf{1}^s$ and $\mathbf{0}^s$ being fixed points.
- *Efficient Nonlinear Operations.* Nonlinear operations over $\mathbb{F}_2^n$ such as

$$x \mapsto x \oplus (\bar{x} \lll 1) \odot (x \lll 2)$$

  for odd n, where $\bar{x} := x \oplus \mathbf{1}^s$, are also possible. This corresponds to the χ-function [Dae95, Table A.1] already used in Keccak/SHA-3, which is known to be invertible for $\gcd(n, 2) = 1$. Moreover, $\mathbf{1}^s$ and $\mathbf{0}^s$ are fixed points.

A rotation of the bits at the output of $\mathcal{S}_i$ may be necessary in order to reduce the number of fixed points. Similar examples can be constructed for other primes, as given in Section 4.

Composition $\mathcal{C}$.

The final operation $\mathcal{C}$ is the inverse of the decomposition. Given $(x'_1, x'_2, \ldots, x'_m)$ we interpret them as integers and compute

$$y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i-1} s_j} \cdot x'_i \bmod p.$$

One may additionally permute $\{x'_i\}_i$, but our construction does not need this extra operation and we omit it for brevity.

2.2 Well-Definition and Bijectivity

Here we prove that our $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ defined in Eq. (8.1) and in particular its $\mathcal{S}$ components are invertible and well-defined.

Lemma 1. *Let $\mathcal{S}_i$ be a permutation over $\mathbb{F}_2^{s_i}$ such that $\mathcal{S}_i(\mathbf{1}^{s_i}) = \mathbf{1}^{s_i}$, where $i \in \{1, 2, \ldots, m\}$. If $p = 2^{\sum_{i \leq m} s_i} - 1 = 2^n - 1$, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection on $\mathbb{F}_p$.*

Proof 1. *Clearly, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection over $\mathbb{Z}_{2^n}$ as it is merely a chunkwise application of invertible S-boxes. Note that $2^n - 1$ is mapped to itself, as $\mathcal{D}$ and $\mathcal{C}$ preserve it by definition, and each S_i maps the binary 1-vector to itself also by definition. Therefore, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ maps $\mathbb{Z}_{2^n - 1} = \mathbb{Z}_{2^n} \setminus \{2^n - 1\}$ bijectively to itself.* □

Lemma 2. *Let $\mathcal{S}_i$ be a permutation over $\mathbb{F}_2^{s_i}$ such that $\mathcal{S}_i(\mathbf{0}^{s_i}) = \mathbf{0}^{s_i}$ and $\mathcal{S}_i(\mathbf{1}^{s_i}) = \mathbf{1}^{s_i}$. If $p = 2^n - 2^\eta + 1 = 2^{\sum_{i \le m} s_i} - 2^{\sum_{i \le \ell} s_i} + 1$, then $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection on $\mathbb{F}_p$.*

Proof 2. *Again, $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ is a bijection over $\mathbb{Z}_{2^n}$ as it is merely a chunkwise application of invertible S-boxes. Let us investigate its behaviour on $x \ge 2^n - 2^\eta$.*

- *If $x = 2^n - 2^\eta$, it is decomposed into $(2^{s_m} - 1, 2^{s_{m-1}} - 1, \ldots, 2^{s_{t+1}} - 1, 0, 0, \ldots, 0)$. All these values are fixed points under $\mathcal{S}_i$, and hence $2^n - 2^\eta$ is mapped to itself by Bar.*
- *If $x > 2^n - 2^\eta$, it is decomposed into $(2^{s_m} - 1, 2^{s_{m-1}} - 1, \ldots, 2^{s_{l+1}-1}, a_l, \ldots, a_2, a_1)$ where at least one of a_i is nonzero. All first $m - t$ values are fixed points, whereas at least one of the last l values is nonzero and thus not mapped to zero. Therefore, x is mapped to some $y > 2^n - 2^\eta$.*

Due to the bijectivity of $\mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(\cdot)$ over $\mathbb{Z}_{2^n}$, we obtain that the set $\{x > 2^n - 2^\eta\}$ is mapped to itself and therefore $\mathbb{Z}_{2^n - 2^\eta + 1}$ is mapped to itself as well. □

2.3 Kintsugi, Earlier Bars, and Side-Channel Considerations

Here, we briefly explain the differences and the analogies between the Kintsugi strategy just described and the Bars functions proposed in Reinforced Concrete (and subsequently used in Tip5). Recall that in Reinforced Concrete an element of $\mathbb{F}_p$ is represented as a vector from $\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_l}$.

- We rely on the structure of the prime p. Thanks to its composition of a few powers of two, the decomposition now is simply a bit extraction rather than a chain of modular reductions, which is expensive both natively and inside the proof system. The bijectivity of Kintsugi is guaranteed under the minor and easily satisfied condition that some specific inputs are fixed points.
- The S-boxes of Reinforced Concrete or Tip5 do not have a simple representation, and must be implemented as tables both for native and circuit computations. The Kintsugi strategy instantiates the S-boxes with AndRX transformations, which are fast and constant-time in native x86 implementations but can easily be transformed to table lookups for circuits.

Side-Channel Leakage and Countermeasures.

Lookup tables in symmetric primitives are a well-known source of side channel leakage. When confidential information is processed (e.g., committing to coin secrets with ZK hash functions in privacy-preserving payment systems), an adversary may recover a large portion of it from timing differences of lookups into memory or caches. These works are well-known since at least two decades in the context of encryption [Pag02; Ber05; OST06], and the high-level ideas

have found first applications in zero-knowledge proof systems [TBP20]. The lookup-oriented designs Reinforced Concrete and Tip5 use specific tables for which a constant-time implementation with reasonable overhead is nontrivial. It is thus of utmost importance to have a design where lookups can be replaced with constant-time operations.

2.4 Statistical and Algebraic Properties

In this section we prove generic statement that link together algebraic and statistical properties of mappings over $\mathbb{F}_p$, which we will use in the security analysis of our construction Monolith.

Lemma 3. *Let $p \geq 3$ be a prime number, and let $\mathcal{F}$ denote the squaring function $x \to x^2$ over $\mathbb{F}_p$. Let F be any interpolant of $\mathcal{F}$ over $\mathbb{F}_2^{\lceil \log_2 p \rceil}$, i.e., for any $a < p$ and its bit representation a we have that $\mathsf{F}(\mathsf{a})$ is the bit representation of $\mathcal{F}(a)$. Then F has degree at least d, where d is the maximum positive integer such that $d < \log_2 \sqrt{p}$ and $\lceil 2^{d-0.5} \rceil$ is odd.*[1]

Proof 3. *We prove this result by contradiction. Suppose that the degree of F is smaller than d. Then the* XOR *sum of its output over any hypercube of degree d is equal to zero [Lai94], including the hypercube*

$$\mathfrak{H} := \{\mathsf{a}_0 = (0, 0, \ldots, 0), \ldots, \mathsf{a}_{2^d-1} = (0, \ldots, 0, \underbrace{1, \ldots, 1}_{d \text{ ones}})\}.$$

Note that $\mathcal{F}(a_i) = i^2 < p$ by the definition of d. Now consider $\mathfrak{B} = \{\mathsf{a}_i \in \mathfrak{H} \mid i > 2^{d-0.5}\}$, so that (i) $2^{2d} > \mathcal{F}(b \in \mathfrak{B}) > 2^{2d-1}$ and (ii) the $2d$-th least significant bit is set. By simple computation, the size of $\mathfrak{B}$ is $2^d - \lceil 2^{d-0.5} \rceil$. Whenever this number is odd, F does not XOR *to 0 at the $2d$-th least significant bit, which contradicts the previous fact. As a result, the squaring has at least degree d if $\lceil 2^{d-0.5} \rceil$ is odd and $d < \log_2 \sqrt{p}$.* □

Lemma 4 (Differential)**.** *Let F be a function that maps $\mathbb{F}_p$ to itself with a differential $\Delta_I \to \Delta_O$ holding with probability $0 < \alpha < 1$, i.e., $|\{x \in \mathbb{F}_p \mid \mathcal{F}(x + \Delta_I) = \mathcal{F}(x) + \Delta_O\}| = p \cdot \alpha$. Then we have*

$$\deg(\mathcal{F}) > \alpha \cdot p\,, \tag{8.3}$$

where $\deg(\mathcal{F})$ is the degree of $\mathcal{F}$ as a polynomial over $\mathbb{F}_p$.

Proof 4. *By definition, $\mathcal{F}(x + \delta_{in}) = \mathcal{F}(x) + \delta_{out}$ has at most $\alpha \cdot p < p$ solutions $x_1, x_2, \ldots, x_{\alpha p}$. Therefore, the polynomial $\mathcal{G}(x) := \mathcal{F}(x + \delta_{in}) - \mathcal{F}(x) - \delta_{out}$ is divisible by the polynomial $(x - x_1) \cdot (x - x_2) \cdot \cdots \cdot (x - x_{\alpha \cdot p})$ of degree $\alpha \cdot p$, and so it has a degree of at least $\alpha \cdot p$. As the degree of the polynomial $\mathcal{G}$ is smaller than the degree of $\mathcal{F}$ by 1, we obtain that $\deg(\mathcal{F}) > \alpha \cdot p$.* □

[1] For example, $\lceil 2^{d-0.5} \rceil$ is odd for $d \in \{2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 16, 21, \ldots\}$.

Lemma 5 (Linear Approximation)**.** *Let F be a function that maps $\mathbb{F}_p$ to itself such that there exists a linear approximation (a, b) with probability $0 < \beta < 1$, that is, $\frac{|\{x \in \mathbb{F}_p | \mathcal{F}(x) = a \cdot x + b\}|}{p} = \beta$. Then we have*

$$\deg(\mathcal{F}) \geq \beta \cdot p\,. \tag{8.4}$$

Proof 5. *By definition, the equation $\mathcal{F}(x) = A \cdot x + B$ has at most $\beta \cdot p$ solutions $x_1, x_2, \ldots, x_{\beta \cdot p}$. Therefore, the polynomial $\mathcal{G}(x) := \mathcal{F}(x) - (a \cdot x + b)$ is divisible by the polynomial $(x - x_1) \cdot (x - x_2) \cdot \cdots \cdot (x - x_{\beta p})$ of degree $\beta \cdot p$. Similar to before, we conclude that F has degree at least equal to $\beta \cdot p$.* □

Based on the previous result, we can immediately conclude the following.

Corollary 1. *Let $\mathcal{F}$ be a function that maps $\mathbb{F}_p$ to itself with $b < p$ fixed points, that is, $|\{x \in \mathbb{F}_p : \ F(x) = x\}| = b$. It follows that*

$$\deg(\mathcal{F}) \geq b\,. \tag{8.5}$$

3 Feistel Type-3 Layer and the Wide Trail Strategy

Here we introduce another component and explore its security properties. This component, which is the Feistel Type-3 layer [ZMI89], is nonlinear and complements Kintsugi Bars. The reason is that even though Bars is a high-degree function, and thus counteracts algebraic attacks, it also has weak differential properties and is thus vulnerable to differential cryptanalysis. In contrast, this new layer has a low degree but strong resistance against differential attacks. We follow the naming convention of Reinforced Concrete (the first look-up based hash function) where the nonlinear layer providing protection against statistical attacks is called Bricks, and use the same name for uniformity.

The Feistel Type-3 layer is a member of a larger Feistel family [HR10], which has been largely neglected in the favour of SPN schemes in block cipher and hash function design, primarily for its complexity and worse diffusion properties. However, we have found its simple version with the square mapping particularly attractive as it is cheaper in circuits and, most importantly, its blend with an MDS layer yields differential properties similar to those in regular SPNs.

Concretely, a generalized Bricks^F on t elements $x_1, x_2, \ldots, x_t$ is defined by

$$\text{Bricks}^F(x_1, \ldots, x_t) := (x_1, x_2 + \mathcal{F}_1(x_1), x_3 + \mathcal{F}_2(x_2), \ldots, x_t + \mathcal{F}_{t-1}(x_{t-1})), \tag{8.6}$$

where $\mathcal{F}_i$ are nonlinear functions. While alone it does not provide fast diffusion, a combination with a matrix layer (as suggested in [BMT13; BFM+16]) increases the differential properties. This approach is well-known in the SPN design as the wide trail design strategy [DR01], where one proves a lower bound for the number of "active" nonlinear components in *any* differential trail and thus establishes an upper bound for the success probability of a differential attack. Here we follow

this line of research, and for the first time we derive bounds for the SPN structure where the nonlinear layer is a Feistel Type-3 function.[1]

Proposition 1. *Consider an r-round construction, where each round consists of the application of BRICKS^F over $\mathbb{F}_q^t$ (for $q = p^s$ with $p \geq 2$ and $s \geq 1$) as in Eq. (8.6) followed by the multiplication with a $t \times t$ MDS matrix. The minimum number s_{min} of active functions $\mathcal{F}_i$ in any differential trail satisfies*

$$s_{min} \geq (t-1) \cdot \left(\frac{3r - 2 - (-2)^{1-r}}{9} \right) \geq (t-1) \cdot \left(\frac{3r - 2.5}{9} \right) .$$

Proof 6. *Denote the number of active words in the input and the output of the i-th BRICKS^F layer by a_i and b_i, respectively. Then we exploit two properties.*

- *Each active input word x_i to BRICKS^F activates $\mathcal{F}_i$ if $i < t$, hence a words activate at least $a - 1$ functions $\mathcal{F}_i$.*
- *Each active output word y_i of BRICKS^F implies that $\mathcal{F}_{i-1}$ or $\mathcal{F}_{i-2}$ is active if $i > 1$. Hence b words activate at least $\frac{b-1}{2}$ functions.*

Together with the MDS property, which states that $b_k + a_{k+1} \geq t + 1$ for each $k \geq 1$, we obtain the following inequalities for the number s_k of active functions $\mathcal{F}_i$ in round k:

$$\begin{aligned}
s_1 &\geq \max\left\{a_1 - 1, \tfrac{b_1-1}{2}\right\}, \quad b_1 + a_2 \geq t+1,\\
s_2 &\geq \max\left\{a_2 - 1, \tfrac{b_2-1}{2}\right\}, \quad b_2 + a_3 \geq t+1,\\
&\vdots\\
s_{r-1} &\geq \max\left\{a_{r-1} - 1, \tfrac{b_{r-1}-1}{2}\right\}, \quad b_{r-1} + a_r \geq t+1,\\
s_r &\geq \max\left\{a_r - 1, \tfrac{b_r-1}{2}\right\},
\end{aligned}$$

where r is the number of rounds.

Now let $(s_1, s_2, \ldots, s_r)$ be a tuple of values of some valid trail that minimizes $s_{min} := s_1 + s_2 + s_3 + \ldots + s_r$. Note that this tuple turns all inequalities into strict equations, as otherwise we could reduce s_{min}. Now consider any MDS property $b_i + a_{i+1} = t + 1$. If $(b_i - 1)/2 < s_i$, we can increase b_i to make those equal and to not increase s_{min}. Similarly, if $a_{i+1} < s_{i+1}$, we can increase a_i to make those equal and to not increase s_{min}. Thus we conclude that for an optimal tuple the values b_i and a_{i+1} are the maxima that determine s_i and s_{i+1} respectively. This simplifies our system, i.e.,

$$\begin{aligned}
&s_1 = \tfrac{b_1-1}{2}, \quad b_1 + a_2 = t+1, && s_2 = a_2 - 1 = \tfrac{b_2-1}{2}, \quad b_2 + a_3 = t+1,\\
&s_3 = a_3 - 1 = \tfrac{b_3-1}{2}, \quad b_3 + a_4 = t+1, \quad \ldots && s_r = a_r - 1,
\end{aligned}$$

and even further, i.e.,

$$\begin{aligned}
&2s_1 + s_2 = t - 1, && 2s_2 + s_3 = t - 1, && 2s_3 + s_4 = t - 1,\\
&\ldots && 2s_{r-1} + s_r = t - 1, && s_r = a_r - 1.
\end{aligned}$$

[1] Our new bound improves the ones recently proposed in [Gra23] for an analogous (but different) scheme.

It is simple to note that if $s_r > 0$, then we could decrease s_{min}. Indeed, if we decrease s_r to 0, we would have to increase s_{r-1} by $s_r/2$, then decrease s_{r-2} by $s_r/4$ and so on, altogether decreasing s_{min} by $s_r \cdot (1 - 1/2 + 1/4 - 1/8 + \cdots) > 0$. Note also that for $s_r \leq t - 1$ all other s_i are non-negative. Thus, the minimum is achieved by $s_r = 0$ and

$$s_{r-1} = \tfrac{t-1}{2}, \quad s_{r-2} = \tfrac{t-1}{4}, \quad s_{r-3} = \tfrac{3(t-1)}{8}, \quad \ldots, \quad s_{r-i} = \tfrac{t-1}{3} \cdot \left(1 + \tfrac{(-1)^{i+1}}{2^i}\right).$$

Substituting these values into the formula for s_{min}, we obtain

$$\begin{aligned} s_{min} &= \sum_{i=0}^{r-1} \frac{t-1}{3} \cdot \left(1 + \frac{(-1)^{i+1}}{2^i}\right) = \frac{t-1}{3} \cdot \sum_{i=0}^{r-1} \left(1 + \frac{(-1)^{i+1}}{2^i}\right) \\ &= \frac{t-1}{3} \cdot \left(r - \sum_{i=0}^{r-1} (-2)^{-i}\right) = \frac{t-1}{3} \cdot \left(r - \frac{2 \cdot (1 - (-2)^{-r})}{3}\right) \\ &= (t-1) \cdot \left(\frac{3r - 2 - (-2)^{1-r}}{9}\right). \end{aligned}$$

□

4 Specification of Monolith

Monolith is a family of permutations which can be used within hash functions and other symmetric constructions. We define the permutation Monolith-64 over $p_{\texttt{Goldilocks}} = 2^{64} - 2^{32} + 1$ with the state consisting of $t = 8$ or $t = 12$ elements. The permutation Monolith-31 is defined over $p_{\texttt{Mersenne}} = 2^{31} - 1$ with the state consisting of $t = 16$ or $t = 24$ elements.

4.1 Modes of Operation

Monolith supports sponge modes and a 2-to-1 compression function.

Sponge-Based Schemes.

First, Monolith can instantiate a sponge [BDP+07; BDP+08] and thus various symmetric constructions such as variable-length hash functions, commitment schemes, authenticated encryption, and stream ciphers. The recently proposed SAFE framework [AKM+22; KBM23] instructs how to handle domain separation and padding in these constructions. In a sponge, the permutation state is split into an outer part with a rate of r elements and an inner part with a capacity of c elements. As we uniformly suggest the 128-bit security level, we set $c = \left\lfloor \frac{256}{\log_2 p} \right\rfloor$ and $r = 2c$.

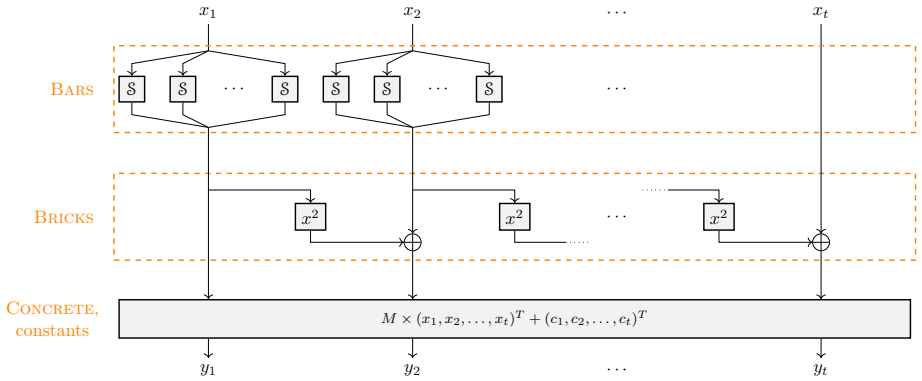


Figure 8.3: One round of the Monolith construction, where $x_i, y_i \in \mathbb{F}_p$.

2-to-1 Compression Function.

We also suggest a fixed-length 2-to-1 *compression function*. Concretely, it takes t $\mathbb{F}_p$ elements as input and produces $t/2$ $\mathbb{F}_p$ elements as output. It is defined as $x \in \mathbb{F}_p^t \mapsto \mathrm{Tr}_{t/2}(\mathcal{P}(x) + x) \in \mathbb{F}_p^t$, where $\mathrm{Tr}_{t/2}$ yields the first $t/2$ elements of the inputs. This compression function can be used in Merkle trees and has recently also been applied in similar constructions, including Anemoi [BBC+23], Griffin [GHR+23], and Poseidon2 [GKS23]. For the 128-bit security level, we set $t = \left\lfloor \frac{512}{\log_2 p} \right\rfloor$, i.e., $t = 8$ for the 64-bit field and $t = 16$ for the 31-bit field (thus factually yielding a little less than 128 bits).

4.2 Permutation Structure

The Monolith permutation is defined as

$$\text{Monolith}(\cdot) = \mathcal{R}_r \circ \cdots \circ \mathcal{R}_2 \circ \mathcal{R}_1 \circ \text{Concrete}(\cdot),$$

where r is the number of rounds and $\mathcal{R}_i$ over $\mathbb{F}_p^t$ are defined as

$$\mathcal{R}_i(\cdot) = c^{(i)} + \text{Concrete} \circ \text{Bricks} \circ \text{Bars}(\cdot), \qquad \forall i \in \{1, 2, \ldots, r\},$$

where Concrete is a linear operation, Bars and Bricks are nonlinear operations over $\mathbb{F}_p^t$, $c^{(1)}, \ldots, c^{(r-1)} \in \mathbb{F}_p^t$ are pseudo-random round constants, and $c^{(r)} = \vec{0}$. Note that a single Concrete operation is applied before the first round. A graphical overview of one round of the construction is shown in Fig. 8.3.

4.3 Bricks

The component Bricks over $\mathbb{F}_p^t$ is defined as a Feistel Type-3 Bricks^F (8.6) with the square map $x \mapsto x^2$, i.e.,

$$\text{Bricks}(x_1, x_2, \ldots, x_t) := (x_1, x_2 + x_1^2, x_3 + x_2^2, \ldots, x_t + x_{t-1}^2).$$

4.4 Concrete

The Concrete layer is defined as

$$\textsc{Concrete}(x_1, x_2, \ldots, x_t) := M \times (x_1, x_2, \ldots, x_t)^T,$$

where $M \in \mathbb{F}_p^{t \times t}$ is an MDS matrix. Since the multiplication with an MDS matrix is in general expensive and requires a number of operations in $\mathcal{O}(t^2)$, we use matrices with special properties.

- *Goldilocks Prime* $p_{\texttt{Goldilocks}}$. We use the circulant matrix circ$(23, 8, 13, 10, 7, 6, 21, 8)$ for $t = 8$ and the matrix circ$(7, 23, 8, 26, 13, 10, 9, 7, 6, 22, 21, 8)$ for $t = 12$, as found and implemented by the Winterfell STARK library.[1] These matrices have the unique advantage of having small elements in the time and frequency domain (i.e., before and after DFT application), allowing for especially fast native performance.

- *Mersenne Prime* $p_{\texttt{Mersenne}}$. We instantiate M via the matrix used in Tip5 [SLS+23] for $t = 16$, since it is also MDS for $p_{\texttt{Mersenne}}$.[2] Since we are not aware of any fast MDS matrix for $t = 24$, we suggest to use a random Cauchy matrix [YMT97] in the concrete layer at the cost of a slower native performance. The problem of finding a fast MDS matrix for this larger state size (which would significantly increase the native performance of Monolith-31 with $t = 24$) is left as future work.

4.5 Bars

The Bars layer is defined as

$$\textsc{Bars}(x_1, x_2, \ldots, x_t) := \textsc{Bar}(x_1) \,||\, \textsc{Bar}(x_2) \,||\, \cdots \,||\, \textsc{Bar}(x_u) \,||\, x_{u+1} \,||\, \cdots \,||\, x_t \tag{8.7}$$

for a t-element state, where $u \in \{1, \ldots, t\}$ denotes the number of Bar applications in a single round. Each Bar application is defined as

$$\textsc{Bar}(x) = \mathcal{C} \circ \mathcal{S} \circ \mathcal{D}(x),$$

where $\mathcal{C}, \mathcal{S}$ and $\mathcal{D}$ are the operations defined in Section 2. In the following, we describe them individually for Monolith-64 and Monolith-31.

Bars for Monolith-64.

In Eq. (8.7) we set $t \in \{8, 12\}$ (compression or sponge use case, resp.) and we set $u = 4$ (i.e., 4 Bar operations are applied in each round).

[1] https://github.com/facebook/winterfell/tree/main/crypto/src/hash/mds

[2] https://github.com/Neptune-Crypto/twenty-first/blob/master/twenty-first/src/shared_math/tip5.rs

Operations $\mathcal{D}$ and $\mathcal{C}$. We use a decomposition into 8-bit values s.t.

$$x = 2^{56}x'_8 + 2^{48}x'_7 + 2^{40}x'_6 + 2^{32}x'_5 + 2^{24}x'_4 + 2^{16}x'_3 + 2^8x'_2 + x'_1.$$

The composition $\mathcal{C}$ is the inverse operation of the decomposition $\mathcal{D}$.

S-Boxes $\mathcal{S}$. In Eq. (8.2) we set $m = 8$. Then all $\mathcal{S}_i$ over $\mathbb{F}_2^8$ are identical as (see [Dae95, Table A.1])

$$\mathcal{S}_i(y) = \big(y \oplus \big((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\big)\big) \lll 1, \tag{8.8}$$

where $\lll$ is a circular shift (here we interpret an integer as a big-endian 8-bit string) and $\overline{y}$ is the bitwise negation.

Bars for Monolith-31.

In Eq. (8.7) we set $t \in \{16, 24\}$ (compression or sponge use case, resp.) and we set $u = 8$ (i.e., 8 Bar operations are applied in each round).

Operations $\mathcal{D}$ and $\mathcal{C}$. The decomposition $\mathcal{D}$ is given by

$$x = 2^{24}x'_4 + 2^{16}x'_3 + 2^8x'_2 + x'_1,$$

where $x'_4 \in \mathbb{Z}_2^7$ and $x'_3, x'_2, x'_1 \in \mathbb{Z}_2^8$. The composition $\mathcal{C}$ is the inverse operation of the decomposition $\mathcal{D}$.

S-Boxes $\mathcal{S}$. In Eq. (8.2) we set $m = 4$ using $\{8, 7\}$-bit lookup tables. Then the S-boxes are defined as (see [Dae95, Table A.1])

$$\begin{aligned} \forall i \in \{1, 2, \ldots, m-1\}: \quad \mathcal{S}_i(y) &= \big(y \oplus \big((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\big)\big) \lll 1, \\ \mathcal{S}_m(y') &= \big(y' \oplus \big((\overline{y'} \lll 1) \odot (y' \lll 2)\big)\big) \lll 1, \end{aligned} \tag{8.9}$$

where $y \in \mathbb{F}_2^8$ and $y' \in \mathbb{F}_2^7$.

4.6 Round Constant Generation

The actual values of pseudo-randomly chosen round constants have no impact on the security. For completeness we provide a generation method in Section A.3.

4.7 Number of Rounds

In Table 8.1, we propose to use $r = 6$ rounds for Monolith-64 and Monolith-31, for which we claim $2\log_2(p_{\texttt{Goldilocks}}) \approx 128$ bits and $4\log_2(p_{\texttt{Mersenne}}) \approx 124$ bits of security, respectively. These numbers are conservatively chosen based on the security analysis proposed in Section 5.

Table 8.1: Parameters for Monolith.

Name	p	Security	Rounds r	Width t		# Bar u
				2-to-1	Sponge	
Monolith-64	$2^{64} - 2^{32} + 1$	128	6	8	12	4
Monolith-31	$2^{31} - 1$	124	6	16	24	8

5 Security Analysis

As some of the components or combinations are new, our analysis contains several nontrivial ideas and may be of separate interest to cryptanalysts and designers. Here are several insights.

- In the spirit of the wide trail strategy [DR02], we prove tight bounds for the number of active squarings in differential characteristics for the Type-3 Feistel-MDS combination in Section 5.1.
- We study rebound attacks in Section 5.2, a research direction that is often missed in the ZK hash function design. We demonstrate practical attacks on a reduced version of Monolith and argue the security of the full version.
- Using differential and linear properties of Bar, we prove lower bounds on its algebraic degree in Section 5.3, which imply resistance against algebraic attacks after a few rounds.
- While arguing the security of Monolith against algebraic attacks, we study the complexity of Gröbner basis attacks on toy versions of Monolith with smaller primes but still realistic Bars layers in Section 5.4.

To summarize, we are not able to break 6 rounds of the proposed scheme or a weaker version of it (i.e., without some of the components) with any basic attacks proposed in the literature. As future work, we encourage to study reduced-round or/and toy variants of our design.

5.1 Differential Cryptanalysis

Given pairs of inputs with some fixed input differences, differential cryptanalysis [BS90] considers the probability distribution of the corresponding output differences produced by the cryptographic primitive. Since the Bars layer is not supposed to have good statistical properties, we simply assume that the attacker can skip it with probability 1.

As the maximum differential probability of a squaring is $1/p$, Proposition 1 immediately implies the following bound.

Corollary 2. *Any 4-round differential characteristic for Monolith has a probability of at most* $p^{\frac{-9(t-1)}{8}}$.

As a result, any characteristic that spans over 5 rounds and more would cover more squarings than the number of state elements, and thus a solution to it cannot be found by standard means. Therefore, a differential-based collision attack on 5 rounds looks infeasible.

5.2 Other Statistical Attacks

We claim that 6 rounds are sufficient for preventing other statistical attacks as well. Here we provide argument to support such conclusion for one of the most powerful statistical attacks against a hash function, that is, the rebound attack. For that goal, we propose an analysis of the number of the fixed points and of the truncated differential characteristics.

Fixed Points.

Contrary to Reinforced Concrete, the Bars layer of Monolith has very few fixed points. Both local maps $x \oplus \big((\overline{x} \lll 1) \odot (x \lll 2) \odot (x \lll 3)\big)$ and $x \oplus \big((\overline{x} \lll 1) \odot (x \lll 2)\big)$ have about $(7/4)^n$ fixed points (for even and odd n, respectively) when considered over $\mathbb{F}_2^n$ (a bit value is preserved if the product of nearby bits is 0). However, all of them except $\mathbf{0}$ and $\mathbf{1} = 2^n - 1$ are destroyed by the circular shift (verified experimentally).

A Bar of Monolith-64, consisting of 8 such S-boxes, admits $2^8 - 2^4 + 1 = 241$ fixed points out of $2^{64} - 2^{32} + 1$. This implies that the probability that a point is fixed is approximately 2^{-56} for Bar and less than $2^{-56 \cdot 4} = 2^{-224}$ for Bars. Similarly, a Bar of Monolith-31 admits $2^4 - 1 = 15$ fixed points out of $2^{31} - 1$. This implies that the probability that a point is fixed is approximately 2^{-27} for Bar and less than $2^{-27 \cdot 8} = 2^{-216}$ for Bars.

For comparison, we recall that a Bar of Reinforced Concrete has $2^{134.5}$ fixed points out of 2^{254} possibilities. Hence, the probability of encountering a fixed point is approximately $2^{-119.5 \cdot 3} = 2^{-358.5}$ for Bars. At the current state of the art, we are not aware of any attack that exploits these fixed points.

Truncated Differential and Rebound Attacks.

Truncated differential attacks [Knu94] are used mostly against primitives that have incomplete diffusion over a few rounds. This is not the case here as the Concrete matrix is MDS. We have not found any other attacks where a truncated differential can be used as a subroutine either.

Rebound attacks [MRS+09] are widely used to analyze the security of various types of hash functions against shortcut collision attacks since the beginning of the SHA-3 competition. It starts by choosing internal state values in the middle of the computation, and then computing in the forward and backward directions to arrive at the inputs and outputs. It is useful to think of it as having central (often called "inbound") and the above mentioned "outbound" parts. In the attack, solutions to the inbound phase are first found, and then are filtered in the outbound phase.

Whereas it is not possible to prove the resistance to the rebound attacks rigorously, we can provide some meaningful arguments to demonstrate that they are not feasible. The inbound phase deals with truncated and regular differentials. By Corollary 2 we see that a solution for a 5-round differential cannot be found, and so the inbound phase cannot cover more than 4 Bricks layers. In the outbound phase, the Concrete layers that surround these Bricks layers make all differentials diffuse to the entire state, so that the next Bricks layers destroy all of those. We hence conclude that 6 rounds of Monolith are sufficient to prevent rebound attacks.

The best attack of this kind that we were able to conduct ourselves is a near-collision attack on the reduced 3-round permutation without the Bars layer. In our attack we show how to find a state that satisfies a differential $\Delta_1 \to \Delta_8$ for certain Δ_1, Δ_8 which are equal in the last $\mathbb{F}_p$ word, i.e., $\Delta_{1,t} = \Delta_{8,t}$. As a concrete application, this yields a zero difference in this word for the compression function $x \mapsto \mathrm{Trunc}_n(\mathcal{P}(x) + x)$, which is a near-collision.

The inbound phase covers 3 layers of Bricks separated by 2 Concrete layers:

$$\Delta_1 \xleftarrow[t \to 1]{\textsc{Concrete}} \Delta_2 \xleftarrow[1]{\textsc{Bricks}}$$
$$\underbrace{\Delta_3 \xrightarrow[1 \to t]{\textsc{Concrete}} \Delta_4 \xleftarrow[t]{\textsc{Bricks}} \Delta_5 \xrightarrow[t \leftarrow 2]{\textsc{Concrete}} \Delta_6 \xrightarrow[2]{\textsc{Bricks}} \Delta_7}_{\text{inbound phase}}$$
$$\xrightarrow[2 \to t]{\textsc{Concrete}} \Delta_8.$$

To find such a state pair, we apply the following approach.

1. In the inbound phase we arbitrarily choose δ and set $\Delta_3 = [0, 0, \ldots, 0, \delta]$ such that its non-zero difference is in the last word only and propagates through Bricks $^{-1}$ untouched. That is, $\Delta_2 = \Delta_3$. Let Δ_1 be $\textsc{Concrete}^{-1}(\Delta_2)$.

2. The inbound phase covers the expansion of Δ_2 to t words and back to the 2-word difference $\Delta_7 = [0, 0, \ldots, 0, \delta_2, \delta_3]$. Note that we have $\Delta_6 = [0, 0, \ldots, 0, \delta_2, \delta_4]$. We arbitrarily set δ_2, δ_3 such that $\Delta_{8,t} = \Delta_{1,t}$ and then choose δ_4 such that
$$\textsc{Concrete}(\Delta_2) = \Delta_{4,1} = \Delta_{5,1} = \textsc{Concrete}^{-1}(\Delta_6).$$

3. As a result, the differential path for the full 3-round scheme is established, and we determine the state. The (δ_3, δ_4) differential determines the input word x_{t-1} of the third Bricks layer, and the equation
$$\textsc{Bricks}(\mathbf{X} + \Delta_4) = \textsc{Bricks}(\mathbf{X}) + \Delta_5.$$
determines input words $x_1, x_2, \ldots, x_{t-1}$ of the second Bricks layer. Note that this is a system of linear equations, and solving it we can determine the full state.

Overall we obtain a partial collision at a negligible cost (the cost for solving the linear system of equations can be approximated by $\mathcal{O}(t^3)$, which is much smaller than the cost for constructing the collision in the case of a random permutation approximated by $\mathcal{O}(p^{1/2})$). We are not aware of any possible extension of such attack to more rounds and/or including Bars, which is left as an open problem for future work.

5.3 Algebraic Analysis: Degree and Density of the Bars Polynomials

Lower Bound on the Degree over $\mathbb{F}_2$.

Using the fact that $\lceil 2^{d-0.5} \rceil$ is odd for $d = 15$ and $d = 30$, Lemma 3 implies the following bound on the degree over $\mathbb{F}_2$.

Proposition 2. *Let $p \in \{p_{Mersenne}, p_{Goldilocks}\}$. Let $\mathcal{F}'$ be an interpolant over $\mathbb{F}_2^{\lceil \log_2 p \rceil}$ of the squaring operation $\mathcal{F}(x) = x^2$ over $\mathbb{F}_p$. Then $\mathcal{F}'$ has degree at least d, where (i) $d = 30$ for $p = 2^{64} - 2^{32} + 1$, and (ii) $d = 15$ for $p = 2^{31} - 1$.*

As the squaring operation is a component of Bricks, we get that it has degree $d \geq 30$ as well.

Lower Bound on the Degree over $\mathbb{F}_p$.

Lemma 6. *Let $n > 4$.*

- *The maximum differential probability over $\mathbb{F}_2^n$ of the S-box Eq. (8.8)*

$$y \mapsto \left(y \oplus \left((\overline{y} \lll 1) \odot (y \lll 2) \odot (y \lll 3)\right)\right) \lll 1$$

  is at least 13/32.

- *The maximum differential probability over $\mathbb{F}_2^n$ of the S-box Eq. (8.9)*

$$y' \mapsto \left(y' \oplus \left((\overline{y'} \lll 1) \odot (y' \lll 2)\right)\right) \lll 1$$

  is at least 1/8.

In particular we have input pairs of form $(x_1, x_2, \ldots, x_{n-1}, 0), (x_1, x_2, \ldots, x_{n-1}, 1)$ mapping to $(y_1, y_2, \ldots, 0, y_n), (y_1, y_2, \ldots, 1, y_n)$ with at least the same probability (13/32 and 1/8, resp.).

Proof 7. *Consider two input states x, y with a single bit difference in bit i such that $x_i = 1 - y_i = 0$. Let us derive sufficient conditions when the output states x', y' differ in bit $i - 1$ only and $x'_i = 1 - y'_i = 0$. This happens if the product in the S-box bit mapping is 0 whenever bit i is `XOR`ed or is part of the product, i.e.,*

$$\overline{y_{i+1}} \odot y_{i+2} \odot y_{i+3} = 0, \qquad y_{i+1} \odot y_{i+2} = 0,$$
$$\overline{y_{i-1}} \odot y_{i+1} = 0, \qquad \overline{y_{i-2}} \odot y_{i-1} = 0.$$

The number of 5-tuples satisfying this system is 13 out of 32 possible. Therefore, a differential holds with probability $13/32$.

For the S-box Eq. (8.9) we observe that a single bit difference in bit i *is mapped to a single bit difference in bit* $i-1$ *if*

$$\overline{y_{i+1}} \odot y_{i+2} = 0, \quad y_{i+1} = 0, \quad \overline{y_{i-1}} = 0,$$

which holds for one 3*-tuple out of 8 ones. Therefore, the differential holds with probability* $1/8$. □

Lemma 7. *The* Bar *function for* $p = 2^{64} - 2^{32} + 1$. *The* Bar *function for* $p = 2^{31} - 1$ *has a differential probability of at least* $2^{-1.2}$.

Proof 8. *The differential probability of* Bar *as a function over* $\mathbb{F}_2$ *is at least the probability of a single S-box, as we can select inputs that activate only one S-box. By Lemma 6 the* $\mathbb{F}_2$*-differential in the last bit implies the* $\mathbb{F}_p$ *differential* $1 \to 2$ *of the same probability. When 8 S-boxes are used, the* $\mathbb{F}_2^{64}$ *differential holds for at least* $13 \cdot 2^{59}$ 64*-bit inputs. To get to* $\mathbb{F}_p$ *we should exclude from those the ones that possibly exceed* p*, i.e.,* 2^{32} *ones. The probability is then lower-bounded by* $2^{-1.4}$.

Similarly, for 31*-bit inputs, Lemma 6 implies that* $3+1$ *concatenated S-boxes together yield a differential probability of at least* $13/32$ *(we activate the weaker* 8*-bit S-box) both when viewed over* $\mathbb{F}_2^{31}$ *and over* $\mathbb{F}_p$. □

Proposition 3. *The* Bars *operation (and its inverse) has degree at least (i)* 2^{62} *for* $p = 2^{64} - 2^{32} + 1$*, and (ii)* 2^{29} *for* $p = 2^{31} - 1$.

The real degree and density values for smaller p are presented in Appendix.

5.4 The CICO Problem for Keyless Algebraic Attacks

A large class of attacks on permutation-based hash functions is reduced to the CICO problem [BDP+09].

Definition 1 (CICO Problem)**.** *A permutation* $\mathcal{P} : \mathbb{F}_p^t \to \mathbb{F}_p^t$ *is secure against the* v*-CICO problem if no algorithm with expected complexity smaller than* p^v *finds* $I_1 \in \mathbb{F}_p^{t-v}$ *and* $O_2 \in \mathbb{F}_p^{t-v}$ *such that* $\mathcal{P}(I_1 \,||\, \underbrace{\mathbf{0}}_{v \text{ words}}) = \underbrace{\mathbf{0}}_{v \text{ words}} \,||\, O_2$.

We use to argue the security of Monolith against some classes of algebraic attacks (in particular Gröbner basis ones) as follows. First, we interpret the output elements as polynomials of the input elements. Then we find a solution to the system of v polynomial equations of $t - v$ input variables (as the remaining v ones are set to zero). Let us now consider two situations.

Univariate Case.

A univariate system appears if $v = t - 1$ or we guess $t - v + 1$ variables. Note that our guess may be invalid if the number of equations exceeds the number of variables, so we have to repeat the guess p^{v-1} times.

- If $v = 1$ and we have guessed $t - 2$ variables, then we have to solve a single polynomial equation faster than in time p. The degree of the polynomial reaches p after 2 applications of the Bars layer, i.e., after 2 rounds. Therefore, solving the equation will require time $\approx p$.

- If $v > 1$, and we have guessed $t - v - 1$ variables, then the chance of a CICO solution for a guess is p^{v-1}. A system of polynomial equations has degree close to p. Solving a system of univariate dense polynomials of degree d is close to d, so we expect spending at least time p to obtain the value of a last variable. Therefore the total complexity still exceeds $p \cdot p^{v-1} = p^v$.

Multivariate Case: Gröbner Bases.

In a more general case we work with a system of v polynomial equations of $t - v$ input variables. The system likely remains solvable if we guess extra $t - 2v$ variables to have both v equations and variables. The main technique of solving these systems is to use Gröbner bases, as described with the following steps.

1. Compute a Gröbner basis for the zero-dimensional ideal of the system of polynomial equations with respect to the *degrevlex* term order.

2. Convert the *degrevlex* Gröbner basis into a *lex* Gröbner basis using the FGLM algorithm [FGL+93].

3. Factor the univariate polynomial in the *lex* Gröbner basis and determine the solutions for the corresponding variable. Back-substitute those solutions, if needed, to determine solutions for the other variables.

The total complexity of a Gröbner basis attack is hence the sum of the respective complexities of the above steps. We argue that even the first step is prohibitively expensive for Monolith.

The complexity of computing a *degrevlex* Gröbner basis, which we denote as $\mathcal{C}_{\text{GB}}$, is difficult to estimate for structured primitives like Monolith. For regular sequences with m equations of degree $d_1, d_2, \ldots, d_m$ and m variables it is shown to be

$$\mathcal{C}_{\text{GB}}^{regular} = \mathcal{O}\left(\binom{m + d_{\text{th}}}{m}^{\omega}\right) \tag{8.10}$$

where d_{th} is called the (theoretical) degree of regularity and computed as $1 + \sum_{i=1}^{m}(d_i - 1)$, and ω, the linear algebra exponent, for equations of reasonable size is close to 2.8.

Unfortunately for real usecases the complexity of GB computation is known to be quite volatile [BSGL20; Sau21]. The main source of discrepancy is arguably the degree of regularity, which is practically the maximum degree reached by polynomials in the GB algorithms. Another problem is scalability: it is nontrivial to scale down an original system of equations to some variant that is solvable on a PC, and get estimates from there. We tackle these problems and the full algebraic security of Monolith as follows:

- We consider a very small, weakened version of Monolith, denoted `Monolith`-Weak1R – with a small prime p, a state with $t = 4$ elements, only one round $\mathcal{F} \coloneqq \text{Concrete} \circ \text{Bricks} \circ \text{Bars} \circ \text{Concrete}$, and only two Bar instances in the round.

- We suggest an arguably optimal representation of `Monolith`-Weak1R as a system of polynomial equations of small degree.

- For various small primes we run an actual GB computation and show that the experimental degree of regularity, d_{mag}, is lower bounded by $d_{\text{th}}/4$.

- We argue that the actual GB computation cost in our experiments can be lower bounded by a modified version of (8.10) with a high security margin:

$$\mathcal{C}_{\text{GB}}^{\text{Monolith}-1R} \gg \mathcal{C}^{bound}(n, d_{\text{th}}) = \binom{n + d_{\text{th}}/4}{n} \tag{8.11}$$

  where n is uniformly 10 in our model and D depends on p and the S-box structure. We note, we use $\omega = 1$ in $\mathcal{C}^{bound}(n, d_{\text{th}})$.

- We show that the application of (8.10) to the full-size state of Monolith and the original primes yields a complexity estimate of 2^{334} and higher.

Based on that, we argue that the full version of Monolith is secure against a GB attack.

Algebraic Model for Bar.

We suggest the following algebraic model for Bar for a decomposition of a prime field element into m buckets with sizes $2^{s_1}, 2^{s_2}, \ldots, 2^{s_m}$:

$$\begin{cases} x = x_1 b_1 + x_2 b_2 + \cdots + x_m b_m, \\ 0 = \prod_{j=0}^{2^{s_i}-1} (x_i - j), \quad 1 \le i \le m, \\ y = \mathcal{L}_1(x_1) b_1 + \mathcal{L}_2(x_2) b_2 + \cdots + \mathcal{L}_m(x_m) b_m. \end{cases}$$

Here, $b_1 = 1$ and $b_i \coloneqq 2^{s_1 + \cdots + s_i}$ for $2 \le i \le m$ and $\mathcal{L}_i(x_i)$ is the interpolation polynomial of degree $2^{s_i} - 1$ for S-box $\mathcal{S}_i$ given by

$$\mathcal{L}_i(x_i) \coloneqq \sum_{0 \le k \le 2^{s_i}-1} \mathcal{S}_i(k) \prod_{\substack{0 \le j \le 2^{s_i}-1 \\ j \ne k}} \frac{x_i - j}{k - j}.$$

The resulting system consists of $m+2$ equations, namely m equations of respective degrees $2^{s_1}, \ldots, 2^{s_m}$, 1 equation of degree $\max_i 2^{s_i} - 1$, and 1 equation of degree 1. The $m + 2$ variables are $x_1, \ldots, x_m, x, y$.

Table 8.2: Results of Gröbner basis computations on small-scale instances of a single round of `Monolith`-Weak1R in the CICO setting. d_{mag} denotes the maximum degree reached during a GB computation with Magma. T is time in microseconds (10^{-6}). Extrapolated estimates are in italic.

	`Monolith`-Weak1R				Monolith-1R	
p	13	29	61	113	31-bit	64-bit
m, n	10, 10	10, 10	10, 10	10, 10	64, 64	48, 48
s_i	2, 2	2, 3	2, 4	4, 3	8,8,8,7	8,...,8
d_{th}	18	34	66	74	9177	9181
d_{mag}	11	14	19	24	*2295*	*2296*
$d_{\text{th}} : d_{\text{mag}}$	1.62	2.43	3.47	3.08	*4*	*4*
$\log_2 T$	16.5	21.5	25.5	30.5		
$\log_2 \mathcal{C}^{bound}(n, d_{\text{th}})$	10.8	16	23	24.3	419.8	333.7

Algebraic Model for One Round of `Monolith`-Weak1R.

We consider a CICO problem with $t = 4$ words, i.e., we are loomonolith:king for $x_2, x_3, x_4 \in \mathbb{F}_p$ and $y_2, y_3, y_4 \in \mathbb{F}_p$ such that

$$\mathcal{F}(0, x_2, x_3, x_4) = (0, y_2, y_3, y_4)\,,$$

where `Monolith`-Weak1R's function $\mathcal{F} :=$ Concrete $\circ$ Bricks $\circ$ Bars $\circ$ Concrete is a single round of Monolith with an added Concrete layer. For Concrete, we use the circulant matrix $M = \text{circ}(2, 1, 1, 1)$, which is not MDS and can thus be seen as an optimistic choice (from the attacker's perspective). We use the following system of equations:

$$\begin{cases} 0 = \text{Concrete}^{-1}(u_1, u_2, u_3, u_4)_0, \\ v_1 = \text{Bar}(u_1), \\ v_2 = \text{Bar}(u_2), \\ 0 = (\text{Concrete} \circ \text{Bricks})(v_1, v_2, u_3, u_4)_0, \end{cases}$$

where $\mathcal{H}(\cdot)_i$ denotes the i-th element of the output of the function $\mathcal{H}$ for $i \in \{1, 2, 3, 4\}$. We note, each Bar function decomposes a prime field element into 2 buckets and $v_i = \text{Bar}(u_i)$ denotes above algebraic model for Bar with $m = 2$. The resulting equation system consists of 10 equations with

- 4 equations for each Bar system $v_i = \text{Bar}(u_i)$, $i = 1, 2$, and
- 2 equations for modelling the CICO constraint at the input and the output.

In total, we have 10 variables, namely $u_1, u_2, u_3, u_4, v_1, v_2$ and 2 internal variables for each Bar system.

Discussion of Gröbner Basis Experiments.

The results of our Gröbner basis experiments on small-scale instances of one round of Monolith with $t = 4$ words and modelled as a CICO problem are depicted in Table 8.2. We conducted our experiments on a machine with an Intel Xeon E5-2630 v3 @ 2.40GHz (32 cores) and 378GB RAM under Debian 11 using Magma V2.26-2.

We see that the real degree d_{mag} is always higher than $d_{\text{reg}}/4$, and we also see that $\mathcal{C}^{bound}$ values are indeed a safe lower bound for the actual computation time for instances of `Monolith`-Weak1R. Assuming the $\mathcal{C}^{bound}$ lower bound, and tamonolith:king into account that Monolith-1R is stronger than `Monolith`-Weak1R, we obtain that the Gröbner basis cost for the Monolith-1R CICO problem should be at least 2^{420} for 31-bit and 2^{333} for the 64-bit version. This hints that the full Monolith is secure against Gröbner basis attacks. We conclude that using Monolith with 6 rounds provides ample security margin against Gröbner basis attacks.

6 Performance Evaluation

6.1 Native Performance

In this section we compare the native performance of Monolith to its competitors with results in Table 8.3. All benchmarks were taken on an AMD Ryzen 9 7900X CPU (singlethreaded, 4.7 GHz).

We included implementations of Monolith into the framework in [IAI21], and also added instantiations of widely popular Poseidon [GKR+21], its modification Poseidon2 [GKS23], and also Griffin [GHR+23] with $p = 2^{64} - 2^{32} + 1$ following their original instance generation scripts.[1] [2] We benchmark these hash functions with a state size of $t = 8$ for the compression mode and of $t = 12$ for the sponge mode in order to have a fair comparison. We also compare against Tip5 with its fixed state size of $t = 16$ using the implementation from [SLS+23],[3] and against Tip4 $'$, a faster instance of Tip5 with a fixed state size $t = 12$, using the implementation from [Sal23].[4] We also compare against Reinforced Concrete instantiated with the scalar field of the BN254 curve, and against SHA3-256/SHA-256 as implemented in `RustCrypto`.[5]

Finally, we compare Monolith-31 with Poseidon and Poseidon2 over the $p_{\texttt{Mersenne}}$ prime field and state sizes of $t = 16$ and $t = 24$ (again for sponge and compression mode), as well as for a constant time implementation (constant time $\mathbb{F}_p$ operations and no lookup tables).

[1] The source code is available at https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo/-/tree/master/plain_impls.

[2] See, e.g., https://github.com/anemoi-hash/hash_f64_benchmarks

[3] https://github.com/Neptune-Crypto/twenty-first

[4] https://github.com/Nashtare/winterfell

[5] https://github.com/RustCrypto/hashes

Table 8.3: Native performance comparison in nano seconds (ns) of different hash functions for variable and constant time implementations. Benchmarks are given for one permutation call, i.e., hashing ≈ 500 bits for all but SHA functions.

Hashing algorithm	Time (ns)		Const. Time (ns)	
	2-to-1	sponge	2-to-1	sponge
$p = 2^{64} - 2^{32} + 1$:	$t = 8$	$t = 12$	$t = 8$	$t = 12$
Monolith-64	**129.9**	**210.5**	**148.5**	**230.4**
Poseidon	1897.6	3288.7	2347.6	4059.1
Poseidon2	944.6	1291.5	1149.2	1617.9
Rescue-Prime	12128.0	19095.0		
Griffin	1815.0	1988.4		
Tip5 ($t = 16$)		463.6		
Tip4′		247.9		
$p = 2^{31} - 1$:	$t = 16$	$t = 24$	$t = 16$	$t = 24$
Monolith-31	**210.3**	**1015.3**	**237.9**	**1120.5**
Poseidon	4478.8	8539.7	4372.9	8538.0
Poseidon2	792.8	1257.4	840.7	1355.3
Other:				
Reinforced Concrete (BN254)		1467.1		
SHA3-256				189.8
SHA-256			45.3	

We see that Monolith-64 is significantly faster than any other arithmetization-oriented hash function. For example, the fastest one, i.e., Poseidon2, is slower by a factor 7.3 for $t = 8$. Tip4 ′, the fastest lookup table based design, is also slower by a factor of 1.9 when using Monolith with the compression mode, and also slower by $36\,ns$ compared to Monolith with the same state size $t = 12$.

Most interestingly, the performance gap between arithmetization-friendly hash functions and traditional ones is now closed, with SHA3-256 being slower than Monolith-64 with $t = 8$ and only faster by $21\,ns$ than Monolith-64 in the sponge mode with $t = 12$.

Regarding Monolith-31 for the 31 bit Mersenne prime field we observe that we still get a fast native performance with $210\,ns$ for $t = 16$. This is significantly faster than Tip5 which has the same state size, but is implemented with the larger 64 bit prime field. Only for $t = 24$ we observe a slower native performance which is due to the usage of a generic MDS matrix in the Concrete layer instead of an optimized circular matrix as we use for the other state sizes. However, competing designs, such as Tip5 also rely on MDS matrices and thus will suffer from the same performance loss. Despite this unoptimized linear layer one can observe that Monolith-31 is still faster than the fastest competitor for the same prime field and state size, i.e., Poseidon2.

Another advantage of Monolith over Tip5, Tip4 ′, and Reinforced Concrete is that its native performance does not rely on lookup tables and its structure allows for constant-time implementations without significant performance loss. The binary χ-like layer can be efficiently implemented using a vectorized implementation that does not require an explicit (de-)composition,

Table 8.4: Plonkish arithmetization comparison for various 64-bit schemes. The numbers are for a single permutation.

Primitive	Lookups	Nonlinear constraints	Degree	Witness size	Area-degree product
Monolith-64-compression	192	44	**2**	460	920
Monolith-64-sponge	192	64	**2**	480	960
Tip5	160	60	7	380	2660
Tip4′	160	40	7	360	2520
Poseidon/Poseidon2 (sponge)	0	118	7	118	826
Rescue-Prime (sponge)	0	96	7	96	**672**

Table 8.5: Performance of proving a Poseidon and Monolith permutation using $p_{\texttt{Goldilocks}}$ and sponge mode in the Plonky2 proof system.

Primitive	Prove Time ms	Verify Time ms	Proof Size B
Monolith-64-sponge	3.49	0.63	112732
Poseidon	6.23	1.12	70288

while unrolling the lookup-tables containing repeated power maps in Reinforced Concrete, Tip5, and Tip4 ′ adds considerable workload to the computation. Thus, the overhead of going to a constant-time implementation only consists of supporting constant-time prime field arithmetic for Monolith, which can help in efficiently preventing side-channel attacks such as the ones proposed in [TBP20].

We observe, that using a constant-time modular reduction leads to a slight slowdown of all benchmarked designs. However, the resulting runtimes are still significantly faster than the non-constant-time runtimes of traditional arithmetization friendly hash functions, such as Poseidon and Griffin, and the variable-time version of Tip4 ′ for $t = 8$ and $t = 12$. Moreover, a constant time Monolith-64 in compression mode is still faster than SHA3-256 for $t = 8$ (even if we acknowledge the different security margin of the two constructions).

Finally, for completeness, we give the runtime of each part of the Monolith permutation for both a constant- and variable-time version in Appendix C.

6.2 Performance in Proof Systems

A modern zero-knowledge proof system defines *arithmetization rules* for the circuit it attempts to prove. Most new proof systems support the *Plonkish* arithmetization, where all input, output, and intermediate variables are placed into a *witness matrix* W with m columns and n rows. The data in each row is restricted by polynomial equations determining the values and computations being used. One of these generic equations of degree 2 is $a_i x_1 x_2 + b_i x_3 + c_i x_4 + d_i = 0$, where a_i, b_i, c_i, d_i are public constants for the i-th row [GWC19]. The Plonkish

arithmetization allows for different tradeoffs between the number of columns or variables being used and the resulting degrees. Additionally, various tuples within a row may be constrained to a set of values in a predefined table $\mathfrak{T}$.

A precise comparison of different arithmetizations is hard without implementing and testing. However, a significant part of the work is to construct m degree-n polynomials for the witness columns and to prove that they satisfy the polynomial equations. The total work is then estimated as an element in $\mathcal{O}(d \cdot n \cdot m)$, where d is the maximum degree of a row polynomial. The cost of using table lookups for FRI-based schemes is currently equivalent to the use of a single polynomial of degree $t = \max\{n, |\mathfrak{T}|\}$.

In this section we give possible arithmetizations for translating Monolith into a set of Plonkish constraints and refer to Section D.1 for R1CS constraints. Our Plonkish arithmetization is designed to accommodate lookup constraints capable of efficiently loomonolith:king up 8-bit values. If the proof system is able to use larger tables (e.g., 16-bit ones), then multiple lookup constraints can be combined into just one larger constraints, reducing the total number of constraints.

Plonkish Arithmetization.

Each composition Concrete ∘ Bricks is described with t polynomial equations of degree 2. Then, for each Bar in the Bars layer, we enforce the correct relations with $x = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} x'_i$ and $y = \sum_{i=1}^{m} 2^{\sum_{j=1}^{i} s_j} y'_i$, while also mamonolith:king sure that the limbs in the decomposition correspond to field elements. For $p_{\texttt{Goldilocks}}$, this means enforcing that either the least significant 32 bits of Bar's input are 0 or the most significant bits are not all 1, i.e.,

$$(x_4 2^{24} + x_3 2^{16} + x_2 2^8 + x_1) \cdot (x_8 2^{24} + x_7 2^{16} + x_6 2^8 + x_5 - z) = 0\,,$$
$$(z - 2^{32} + 1) \cdot z' = 1\,.$$

For $p_{\texttt{Mersenne}} = \texttt{0x7fffffff}$ we need to make sure that the combined values are $\neq p$, which is equivalent to them not being $2^8 - 1$ (three) or $2^7 - 1$ (one), i.e.,

$$(x_4 + x_3 + x_2 + x_1 - 2^7 - 3 \cdot 2^8 + 4) \cdot z' = 1.$$

We describe the application of m individual S-boxes with m lookup constraints $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$. These also include the range checks for each input which are also necessary for the correctness of the constraints above.

Apart from $2m$ lookup variables per Bar, we define u variables at the output of the first Concrete layer (these are the inputs to the Bars layer) and t variables at the output of each of the following Concrete layers (except for the last one). The reason is that the variables after the first Concrete layer store linear relations in the input, and only the u variables entering the Bars layer are needed. For the last layer, the output variables can be used directly. In total, we have $6 \cdot (2um + u) + 5t + u$ variables, where $\{u = 4, m = 8\}$ for the $p_{\texttt{Goldilocks}}$ case and $\{u = 8, m = 4\}$ for the $p_{\texttt{Mersenne}}$ case (considering S-boxes of ≈ 8 bits).

In Table 8.4 we compare the (non-optimized) arithmetization of Monolith with the ones of other 64-bit designs (see Section D.2 for details). To achieve a fair comparison, we do not apply any constraint or witness optimization but try to follow the same approach. We see that both the number of lookups and constraints in Monolith is slightly larger than in Tip5 and Tip4', but the constraint degree is smaller by the factor of 3.5, which should result in an overall decrease of the prover time by a factor of at least 2 (estimated as area-degree product). This is reasonable since Tip5 and Tip4' are able to process more field elements with a permutation call. Poseidon, Poseidon2, and Rescue-Prime due to their comparably small witness size and no lookup tables are estimated to still provide faster proving performance, closely followed by Monolith-64 with its low-degree nonlinear layers. Again, we stress that these numbers are derived from non-optimized arithmetizations and are subject to change. For example, one can leverage the low degree of Monolith to reduce witness size by trading with a larger degree round function. We refer to Section D.3 for details. Furthermore, these estimates are based on a simplified performance metric (are-degree-product) which does not consider every aspect of prover performance and benchmarks in real proof systems might differ.

Benchmarks in Plonky2.

We implemented Monolith-64 in the Plonky2[1] proof system to verify the estimations of Table 8.4.[2] Since Plonky2 already comes with a custom gate of Poseidon in sponge mode ($t = 12$) using $p_{\texttt{Goldilocks}}$ where the whole gate is put into just one row of the trace, we implement Monolith-64-sponge with the same parameters. To highlight the main advantage of Monolith-64, namely its fast native performance, we benchmark proving a Monolith-64 permutation while using Monolith-64 as the hash function to build the Merkle trees in Plonky2. Similarly, we benchmark Poseidon when using Poseidon as the Plonky2 hash function (which is the default setting in Plonky2). The results can be seen in Table 8.5. One can observe that since Monolith requires more witnesses than Poseidon and both gates use just one row in the trace, the resulting proof is larger. However, the combination of proving Monolith-64 while using it at as the Plonky2 hash function leads to half the prover and verifier runtime compared to Poseidon.

Acknowledgments

This work was partially supported by a gift from the Ethereum foundation. Lorenzo Grassi is partially supported by the German Research foundation (DFG) within the framework of the Excellence Strategy of the Federal Government and the States – EXC 2092 CaSa – 39078197. Roman Walch was supported by the "DDAI" COMET Module within the COMET – Competence Centers for Excellent Technologies Programme, funded by the Austrian Federal Ministry for

[1] https://github.com/mir-protocol/monolith:plonky2

[2] Our implementation is available at https://github.com/HorizenLabs/monolith.

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.

Finally, we thank Nicholas Mainardi for helping with the implementation of Monolith in Plonky2 and for improving the efficiency of the gate.

A Fast Reduction for Primes of the Form $\phi^2-\phi+1$ and $2^\rho - 1$

A.1 Fast Reduction for Primes of the Form $\phi^2 - \phi + 1$

Here we describe the fast reduction modulo a prime number of the form $\phi^2-\phi+1$. Note that this includes $p = 2^{64} - 2^{32} + 1$, where $\phi = 2^{32}$. We focus on the case of a multiplication, where two n-bit inputs result in an output of at most $2n$ bits.

Given $\mathbb{F}_p$ for $p = \phi^2 - \phi + 1$, it follows that

$$\phi^2 = \phi - 1 \implies \phi^3 = \phi^2 - \phi = -1.$$

Now, let us write a value x to be reduced as

$$x = x_0 + \phi^2 x_1 + \phi^3 x_2,$$

where $x_0 \in \mathbb{Z}_{2^n}$ and $x_1, x_2 \in \mathbb{Z}_{2^{n/2}}$. Then

$$x = x_0 + (\phi - 1)x_1 - x_2 \pmod p,$$

where note that $\log_2(x_0 + (\phi - 1)x_1 - x_2) \approx \log_2(p)$. This reduction can be computed using only a small number of additions and subtractions.

A.2 Fast Reduction for Primes of the Form $2^\rho - 1$

Here we describe the fast reduction modulo a prime number of the form $2^\rho - 1$ which includes $p = 2^{31} - 1$. We focus on the case of a multiplication, where two ρ-bit inputs result in an output of at most 2ρ bits.

Given $\mathbb{F}_p$ for $p = 2^\rho - 1$, it follows that $2^\rho = 1 + p$. Now, let us write a value x to be reduced as

$$x = x_0 + 2^\rho x_1,$$

where $x_0 \in \mathbb{Z}_{2^\rho}$ and $x_1 \in \mathbb{F}_p$. Then

$$x = x_0 + x_1 + \underbrace{(2^\rho - 1) \cdot x_1}_{=0 \pmod p} = x_0 + x_1 \pmod p.$$

This reduction can be computed using only a small number of additions and binary shifts.

A.3 Generation of Round Constants

The round constants $c_1^{(i)}, c_2^{(i)}, \ldots, c_t^{(i)}$ for the i-th round are generated using the well-known approach of seeding a pseudo-random number generator and reading its output stream. In particular, we use SHAKE128-128 with rejection sampling, i.e., we discard elements which are not in $\mathbb{F}_p$. SHAKE128-128, thereby, is seeded with the initial seed "MONOLITH " followed by the state size t and number of rounds r, each represented as one byte, the prime p represented by $\lceil \log_2(p)/8 \rceil$

bytes in little endian representation, and the decomposition sizes in the bar layer, where each s_i is represented as one byte. Thus, the seed is

```
b'Monolith\x08\x06\x01\x00\x00\x00\xff\xff\xff\xff
\x08\x08\x08\x08\x08\x08\x08\x08'
```

for MONOLITH-64 with $t = 8, r = 6$ and

```
b'Monolith\x10\x06\xff\xff\xff\x7f\x08\x08\x08\x07'
```

for MONOLITH-31 with $t = 16, r = 6$.

B Security Analysis – Additional Material

B.1 Degree and Density over $\mathbb{F}_p$: Practical Results.

Evaluating the actual density of the polynomial resulting from BAR applied to a single field element in $\mathbb{F}_p$, where $p \in \{2^{64} - 2^{32} + 1, 2^{31} - 1\}$, is infeasible in practice. Indeed, any enumeration and subsequent interpolation approach would take far too long.

Therefore, in our experiments we focus on smaller finite fields defined by "similar" prime numbers. In particular, we focus on n-bit primes of the form $2^n - 2^\eta + 1$ for η as close to n as possible. We then apply the S-box S_i to smaller parts of the field element, exactly as in BAR where the S-box is applied to each 8-bit part of the larger field element. We also vary the sizes of the parts to which the S_i are applied in order to get a broader picture.

The results of our evaluation are shown in Table 8.6. For example, in the first case, where $p = 2^8 - 2^4 + 1$, S_i is applied to the first 4 bits (starting from the least significant bit) and then to the next 4 bits, covering the entire field element. The size of these parts is indicated in the second column. As we can see, the maximum degree is reached for all tested primes of the form $2^n - 2^\eta + 1$, where $\eta > 1$. Moreover, for these primes, the density is always close to 100%, mostly matching it. We also applied S_i to elements of $\mathbb{F}_{2^n-1}$ directly, where $n \in \{5, 7, 13\}$, which resulted in almost maximum-degree polynomials of low density (specifically, only 6, 18, and 630 monomials exist in the polynomial representation, respectively). This suggests that increasing the number of S-box applications per field element (i.e., increasing the number of smaller parts to which S_i are applied) is beneficial for the density of the resulting polynomial.

We also evaluated the degrees and density values resulting from the inverse S-boxes applied to the field elements, in order to get an estimation of the algebraic strength of the inverse operation. The results match the results given in Table 8.6, where always more than 99% monomials are reached together with a degree close to the maximum.

Degree and Density over $\mathbb{F}_p^t$: Practical Results.

We also ran tests regarding the density over the entire state. Naturally, this task gets harder with an increased number of rounds, since the degrees are rising too

Table 8.6: Degree and density of the polynomials resulting from Bar applied to various field elements.

p	Bit splittings	Degree	Density
$2^8 - 2^4 + 1$	$\{4, 4\}$	239 $(= p - 2)$	100%
$2^{13} - 2^8 + 1$	$\{8, 5\}, \{4, 4, 5\}$	7935 $(= p - 2)$	$> 99\%$ (7934/7935)
$2^{13} - 2^5 + 1$	$\{5, 8\}, \{5, 4, 4\}$	8159 $(= p - 2)$	$> 99\%$ (8157/8159)
$2^{14} - 2^{10} + 1$	$\{10, 4\}, \{5, 5, 4\}$	15359 $(= p - 2)$	$> 99\%$ (15358/15359)
$2^{14} - 2^4 + 1$	$\{4, 10\}$	16367 $(= p - 2)$	100%
$2^{14} - 2^4 + 1$	$\{4, 5, 5\}$	16367 $(= p - 2)$	$> 99\%$ (16364/16367)
$2^{13} - 1$	$\{5, 8\}, \{8, 5\}, \{4, 9\}, \{9, 4\}$	8189 $(= p - 2)$	$> 99\%$ (8188/8189)
$2^7 - 1$	$\{3, 4\}, \{4, 3\}$	125 $(= p - 2)$	$> 99\%$ (124/125)
$2^5 - 1$	–	26 $(= p - 5)$	$\approx 21\%$ (6/29)
$2^7 - 1$	–	120 $(= p - 7)$	$\approx 14\%$ (18/125)
$2^{13} - 1$	–	8178 $(= p - 13)$	$\approx 8\%$ (629/8189)

Table 8.7: Degree and density of the polynomials after a single round, where $t = 4$ and two input variables are used (with the other two input elements being fixed).

p	Bit splittings	Degree	Density
$2^8 - 2^4 + 1$	$\{4, 4\}$	239 $(= p - 2)$	$> 99\%$ (28785/28920)
$2^7 - 1$	$\{3, 4\}$	125 $(= p - 2)$	$> 98\%$ (7919/8001)
$2^7 - 1$	$\{4, 3\}$	125 $(= p - 2)$	$> 98\%$ (7919/8001)

quickly. In our tests we focused on $p \in \{2^8 - 2^4 + 1, 2^7 - 1\}$ and $t = 4$, and we give the results together with the sizes of the smaller S-boxes in Table 8.7.

As can be seen, the maximum number of monomials is almost reached after a single round. We suspect that some of the monomials are not reached due to cancellations, which is reasonable when considering these small prime fields. Still, we acknowledge this fact by adding another round on top of that in order to ensure that all polynomial representations of the state are dense and of maximum degree. Thus, having 6 rounds achieves 4 rounds of security margin regarding degrees and density of polynomials.

B.2 Non-Applicable Attacks

We emphasize that we do not claim security of Monolith against zero-sum partitions [BCC11] (which can be set up via higher-order differentials [Knu94; BCD+20] and/or integral/square attacks [DKR97]). In such an attack, the goal is to find a collection of disjoint sets of inputs and corresponding outputs for the given permutation that sum to zero (i.e., satisfy the zero-sum property). Our choice is motivated by the fact that, to the best of our knowledge, it is not possible to turn such a distinguisher into an attack on the hash and/or compression

function. For example, in the case of SHA-3/Keccak [Nat15; BDP+11], while 24 rounds of Keccak-f can be distinguished from a random permutation using a zero-sum partition [BCC11] (that is, full Keccak-f), preimage/collision attacks on Keccak can only be set up for up to 6 rounds of Keccak-f [GLL+20]. Indeed, the authors of Keccak-f deem a 12-round version of the primitive to provide ample security margin [BDP+18]. For this reason and as already done in similar work [GKR+21; GHR+23], we ignore zero-sum partitions for practical applications.

C Benchmarks of Different Round Functions

In Table 8.8, we give the runtime of each part of the Monolith permutation for both a constant- and variable-time implementation.

Table 8.8: Native performance of each different round function in Monolith. Implemented in Rust. * indicates an implementation without circulant MDS matrix.

Operation	Time (ns)		Const. Time (ns)	
$p = 2^{64} - 2^{32} + 1$:	$t = 8$	$t = 12$	$t = 8$	$t = 12$
Concrete	19.5	33.6	19.5	33.6
Bricks	12.2	19.3	16.0	21.8
Bars	10.4	12.9	10.4	12.9
$p = 2^{31} - 1$:	$t = 16$	$t = 24$	$t = 16$	$t = 24$
Concrete	31.8	138.1*	31.9	138.1*
Bricks	17.0	21.7	17.0	21.7
Bars	8.4	12.0	8.4	12.0

D Arithmetization Details

D.1 R1CS

It is possible, though more expensive, to implement Monolith in legacy proof systems that only support R1CS equations without any table lookups. In contrast to Reinforced Concrete, our design admits a reasonably small R1CS representation described in the following. First, we use $t - 1$ constraints to generate equations for Bricks. For Bars, we decompose each element that goes into a Bar into bits thus using one constraint per Bar for the actual decomposition plus $\log_2(p) \cdot \#\text{Bar}$ constraints for ensuring that the bits are either 0 or 1. Then each output bit of Bar requires 3 multiplications (2 for `AND` and 1 for `XOR`) for the 8-bit S-box and 2 multiplications for the 7-bit one as used in Monolith-31. By combining the composition constraints with the following bricks layer we

get 1028 constraints for Monolith-64 and 944 constraints for Monolith-31 per Bars. Finally, the Concrete layer can be included in the constraints of Bricks and Bars, resulting in a total for $R \cdot (1027 + t)$ R1CS constraints for Monolith-64 and $R \cdot (943 + t)$ constraints for Monolith-31, where R is the number of rounds.

D.2 Circuits for Other Hash Functions

The Tip5 function applies four 64-bit S-boxes with lookups per round, so 32 8-bit lookups per round. It also uses 12 degree-7 power functions per round. We allocate variables for the outputs of the power functions in addition to 64 lookup variables per round.

Similarly, the Tip4' function also applies 32 8-bit lookups per round to the smaller state. However, it uses 8 degree-7 power functions per round, proportionally reducing the number of variables.

The Poseidon2 function (as well as Poseidon which has the same number of rounds and the same arithmetization) with $t = 12$ defined for $p_{\texttt{Goldilocks}}$ has 8 full and 22 partial rounds, thus 118 degree-7 functions in total. We allocate variables for all outputs of the S-boxes, and link the others via linear equations.

Regarding Rescue-Prime, an instance with $t = 12$ defined for $p_{\texttt{Goldilocks}}$ requires 8 rounds which each consist of two subrounds which alternate between nonlinear layers featuring the x^d and $x^{1/d}$ power maps. Due to this construction one can find degree-7 constraints spanning a whole round of rescue, leading to 96 degree-7 constraints in total.

D.3 Multiround Constraints for Monolith

We consider $p = p_{\texttt{Goldilocks}}$ and $t = 12$. When implementing both Monolith and Tip5 in a single gate, we can immediately observe various similarities. For example, considering 8-bit lookups, the number of lookups is almost the same, with Tip5 using slightly fewer ones due to its lower number of rounds (note that both permutations use four lookup words per round). Moreover, the number of necessary columns is similar in a round-based approach.

The major advantage of Monolith becomes apparent after considering the degree of the constraints. Indeed, while Tip5 uses a maximum degree of 7 (which is the smallest integer d such that $\gcd(p_{\texttt{Goldilocks}} - 1, d) = 1$), Monolith uses a maximum degree of only 2. Not only does this lead to more efficient constraints, but it allows for different tradeoffs. For example, consider $p = p_{\texttt{Goldilocks}}, t = 12$ and a state after the Concrete layer defined by 12 variables $w_1^{(1)}, \dots w_{12}^{(1)}$. After the subsequent application of Bars, we add 4 new variables $w_1^{(2)}, \dots, w_4^{(2)}$ for the state elements modified by the lookup table. We now apply Bricks and then Concrete to the state. Note that describing the state in $w_5^{(1)}, \dots w_{12}^{(1)}, w_1^{(2)}, \dots, w_4^{(2)}$ after these transformations results in degree-2 constraints (ignoring the table lookups), since only one Bricks layer has been applied. Hence, we may now choose to only add 4 new variables $w_1^{(3)}, \dots, w_4^{(3)}$ after the application of the last

Concrete layer at the positions of the table lookups. After the next Bars layer, the state is defined by 8 polynomial equations in $w_5^{(1)}, \dots w_{12}^{(1)}, w_1^{(2)}, \dots, w_4^{(2)}$ of degree 2 and by the 4 new variables $w_1^{(4)}, \dots, w_4^{(4)}$ resulting from the table lookups. After applying the next Bricks and Concrete layers, we arrive at a state defined by 12 polynomial constraints in $w_5^{(1)}, \dots w_{12}^{(1)}, w_1^{(2)}, \dots, w_4^{(2)}, w_1^{(4)}, \dots, w_4^{(4)}$ of degree 4. A graphical overview of this approach is shown in Fig. 8.4.

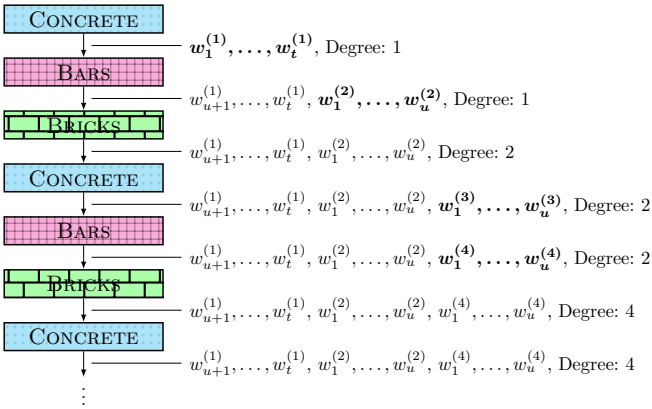


Figure 8.4: Variables (or trace elements) when using Monolith with degree-4 constraints. Newly added variables are emphasized in **bold** and the degree indicates the maximum degree of the polynomial equations describing the corresponding state in the given variables.

As a result, with degree-4 constraints we can save $t - u$ trace elements in each pair of rounds, where u is the number of Bar applications in the Bars layers. This allows us to achieve a slimmer row with even fewer columns. We point out that this advantage of Monolith's low degree also applies in a similar fashion when comparing to other hash functions which use x^d, such as Poseidon, Poseidon2, Rescue, Griffin, Anemoi, and many more.

References

[AAE+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. "Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols." In: *IACR Trans. Symmetric Cryptol.* 2020.3 (2020), pp. 1–45.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity." In: *ASIACRYPT 2016*. Vol. 10031. LNCS. 2016, pp. 191–219.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. "NORX: Parallel and Scalable AEAD." In: *ESORICS 2014*. Vol. 8713. LNCS. 2014, pp. 19–36.

[AKM+22] Jean-Philippe Aumasson, Dmitry Khovratovich, Bart Mennink, and Porçu Quine. *SAFE (Sponge API for Field Elements) - A Toolbox for ZK Hash Applications*. https://eprint.iacr.org/2023/522. 2022.

[ANW+13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. "BLAKE2: Simpler, Smaller, Fast as MD5." In: *ACNS 2013*. Vol. 7954. LNCS. 2013, pp. 119–135.

[Bal23] Balazs Komuves. *hash-circuits*. 2023. URL: https://github.com/bkomuves/hash-circuits (visited on 10/06/2023).

[BBC+23] Clémence Bouvier, Pierre Briaud, Pyrros Chaidos, Léo Perrin, Robin Salen, Vesselin Velichkov, and Danny Willems. "New Design Techniques for Efficient Arithmetization-Oriented Hash Functions: Anemoi Permutations and Jive Compression Mode." In: *CRYPTO 2023*. Vol. 14083. LNCS. 2023, pp. 507–539.

[BBH+19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup." In: *CRYPTO 2019*. Vol. 11694. LNCS. 2019, pp. 701–732.

[BC23] Benedikt Bünz and Binyi Chen. "ProtoStar: Generic Efficient Accumulation/Folding for Special Sound Protocols." In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/620, p. 620.

[BCC11] Christina Boura, Anne Canteaut, and Christophe De Cannière. "Higher-Order Differential Properties of Keccak and *Luffa*." In: *FSE 2011*. Vol. 6733. LNCS. 2011, pp. 252–269.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. "Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems." In: *CRYPTO 2020*. Vol. 12172. LNCS. 2020, pp. 299–328.

[BDP+07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *Sponge functions*. In: Ecrypt Hash Workshop 2007, http://www.csrc.nist.gov/pki/HashWorkshop/PublicComments/2007_May.html. 2007.

[BDP+08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "On the Indifferentiability of the Sponge Construction." In: *EUROCRYPT 2008*. Vol. 4965. LNCS. 2008, pp. 181–197.

[BDP+09] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak sponge function family main document." In: *Submission to NIST (Round 2)* 3.30 (2009), pp. 320–337.

[BDP+11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. *Note on zero-sum distinguishers of Keccak-f*. Available at https://keccak.team/files/NoteZeroSum.pdf. 2011.

[BDP+18] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. "KangarooTwelve: Fast Hashing Based on Keccak-p." In: *ACNS 2018*. Vol. 10892. LNCS. 2018, pp. 400–418.

[Ber05] Daniel J. Bernstein. *Cache-timing attacks on AES*. Avaiable at http://cr.yp.to/antiforgery/cachetiming-20050414.pdf. 2005.

[BFM+16] Thierry P. Berger, Julien Francq, Marine Minier, and Gaël Thomas. "Extended Generalized Feistel Networks Using Matrix Representation to Propose a New Lightweight Block Cipher: Lilliput." In: *IEEE Trans. Computers* 65.7 (2016), pp. 2074–2089.

[BMT13] Thierry P. Berger, Marine Minier, and Gaël Thomas. "Extended Generalized Feistel Networks Using Matrix Representation." In: *SAC 2013*. Vol. 8282. LNCS. 2013, pp. 289–305.

[BS90] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems." In: *CRYPTO 1990*. Vol. 537. LNCS. 1990, pp. 2–21.

[BSGL20] Eli Ben-Sasson, Lior Goldberg, and David Levit. *STARK Friendly Hash – Survey and Recommendation*. Cryptology ePrint Archive, Paper 2020/948. https://eprint.iacr.org/2020/948. 2020.

[But22] Vitalik Buterin. *What we want out of STARK signature aggregation*. available at https://t.ly/UZMKw. 2022.

[CFG+22] Shumo Chu, Boyuan Feng, Brandon H. Gomes, Francisco Hernández Iglesias, and Todd Norton. *MantaPay Protocol Specification*. available at https://github.com/Manta-Network/spec/blob/main/monolith:manta-pay/spec.pdf. 2022.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography." In: *EUROCRYPT 2020*. Vol. 12105. LNCS. 2020, pp. 769–793.

[Dae95] Joan Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis.* Doctoral Dissertation. Available at https://cs.ru.nl/~joan/papers/JDA_Thesis_1995.pdf. 1995.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. "The Block Cipher Square." In: *FSE 1997*. Vol. 1267. LNCS. 1997, pp. 149–165.

[DR01] Joan Daemen and Vincent Rijmen. "The Wide Trail Design Strategy." In: *Cryptography and Coding - IMA International Conference 2001*. Vol. 2260. LNCS. 2001, pp. 222–238.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Available at https://cs.ru.nl/~joan/papers/JDA_VRI_Rijndael_2002.pdf. Springer, 2002.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. "cq: Cached quotients for fast lookups." In: *IACR Cryptol. ePrint Arch.* (2022). https://eprint.iacr.org/2022/1763.

[FGL+93] Jean-Charles Faugère, Patrizia M. Gianni, Daniel Lazard, and Teo Mora. "Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering." In: *J. Symb. Comput.* 16.4 (1993), pp. 329–344.

[GHR+23] Lorenzo Grassi, Yonglin Hao, Christian Rechberger, Markus Schofnegger, Roman Walch, and Qingju Wang. "Horst Meets Fluid-SPN: Griffin for Zero-Knowledge Applications." In: *CRYPTO 2023*. Vol. 14083. LNCS. 2023, pp. 573–606.

[GKL+22] Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. "Reinforced Concrete: A Fast Hash Function for Verifiable Computation." In: *ACM CCS*. 2022, pp. 1323–1335.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon: A New Hash Function for Zero-Knowledge Proof Systems." In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 519–535.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. "Poseidon2: A Faster Version of the Poseidon Hash Function." In: *AFRICACRYPT 2023*. Vol. 14064. LNCS. 2023, pp. 177–203.

[GLL+20] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. "Practical Collision Attacks against Round-Reduced SHA-3." In: *J. Cryptol.* 33.1 (2020), pp. 228–270.

[Gra23] Lorenzo Grassi. "Bounded Surjective Quadratic Functions over Fnp for MPC-/ZK-/FHE-Friendly Symmetric Primitives." In: *IACR Trans. Symmetric Cryptol.* 2023.2 (2023), pp. 94–131.

[GW20] Ariel Gabizon and Zachary J. Williamson. "plookup: A simplified polynomial protocol for lookup tables." In: *IACR Cryptol. ePrint Arch.* (2020).

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Non-interactive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. 2019.

[Hab23] Ulrich Haböck. *Brakedown's expander code*. Cryptology ePrint Archive, Paper 2023/769. https://eprint.iacr.org/2023/769. 2023.

[HLN23] Ulrich Haböck, Daniel Lubarov, and Jacqueline Nabaglo. *Reed-Solomon Codes over the Circle Group*. Cryptology ePrint Archive, Paper 2023/824. https://eprint.iacr.org/2023/824. 2023.

[HR10] Viet Tung Hoang and Phillip Rogaway. "On generalized Feistel networks." In: *Annual Cryptology Conference*. Springer. 2010, pp. 613–630.

[IAI21] IAIK. *Hash functions for Zero-Knowledge applications Zoo*. https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo. IAIK, Graz University of Technology. Aug. 2021.

[KBM23] Dmitry Khovratovich, Mario Marhuenda Beltrán, and Bart Mennink. "Generic Security of the SAFE API and Its Applications." In: *IACR Cryptol. ePrint Arch.* (2023). to appear at ASIACRYPT'23, p. 520.

[KMT22] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. "MinRoot: Candidate Sequential Function for Ethereum VDF." In: *IACR Cryptol. ePrint Arch.* (2022).

[Knu94] Lars R. Knudsen. "Truncated and Higher Order Differentials." In: *FSE 1994*. Vol. 1008. LNCS. 1994, pp. 196–211.

[KS23] Abhiram Kothapalli and Srinath T. V. Setty. "HyperNova: Recursive arguments for customizable constraint systems." In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/573, p. 573.

[KST22] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes." In: *CRYPTO 2022*. Vol. 13510. LNCS. 2022, pp. 359–388.

[Lai94] Xuejia Lai. "Higher Order Derivatives and Differential Cryptanalysis." In: *Communications and Cryptography: Two Sides of One Tapestry*. Springer US, 1994, pp. 227–233.

[Mona] *monolith:Polygon zkEVM Documentation*. https://docs.hermez.io/zkEVM/Overview/Overview/. 2022.

[Monb] *ZKEVM Introduction*. https://github.com/privacy-scaling-explorations/zkevm-specs/blob/master/specs/introduction.md. 2022.

[mon22] monolith:Polygon. *Introducing Plonky2*. 2022. URL: https://blog.polygon.technology/introducing-monolith:plonky2/ (visited on 10/20/2022).

[mon23] monolith:Polygon. *Plonky3*. 2023. URL: https://github.com/Plonky3/Plonky3 (visited on 06/12/2023).

[MRS+09] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. "The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl." In: *FSE 2009*. Vol. 5665. LNCS. 2009, pp. 260–276.

[Nat15] National Institute of Standards and Technology. "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions." In: *Federal Information Processing Standards Publication (FIPS)* (202 2015).

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES." In: *CT-RSA 2006*. Vol. 3860. LNCS. 2006, pp. 1–20.

[Pag02] D. Page. *Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel*. Cryptology ePrint Archive. https://eprint.iacr.org/2002/169. 2002.

[PH23] Shahar Papini and Ulrich Haböck. "Improving logarithmic derivative lookups using GKR." In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/1284.

[PSS19] Alexey Pertsev, Roman Semenov, and Roman Storm. *Tornado Cash Privacy Solution Version 1.4*. available at https://t.ly/ys_pW. 2019.

[(RI23] Jeremy Bruestle (RISC0). private communication. 2023.

[RIS23] RISC Zero. *RISC Zero : General-Purpose Verifiable Computing*. 2023. URL: https://www.monolith:risczero.com/ (visited on 02/02/2023).

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. *Rescue-Prime: a Standard Specification (SoK)*. Cryptology ePrint Archive, Report 2020/1143. 2020.

[Sal23] Robin Salen. *Two additional instantiations from the Tip5 hash function construction*. https://toposware.com/paper_tip5.pdf. 2023.

[Sau21] Jan Ferdinand Sauer. *Blog: Gröbner Basis – Attacking a Tiny Sponge*. available at https://jfs.sh/blog/gb-attacking-tiny-sponge/. 2021.

[SLS+23] Alan Szepieniec, Alexander Lemmens, Jan Ferdinand Sauer, Bobbin Threadbare, and Al-Kindi. *The Tip5 Hash Function for Recursive STARKs*. Cryptology ePrint Archive, Paper 2023/107. https://eprint.iacr.org/2023/107. 2023.

[STW23] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. "Unlocking the lookup singularity with Lasso." In: *IACR Cryptol. ePrint Arch.* (2023). https://eprint.iacr.org/2023/1216.

[TBP20] Florian Tramèr, Dan Boneh, and Kenny Paterson. "Remote Side-Channel Attacks on Anonymous Transactions." In: *USENIX Security Symposium*. USENIX Association, 2020, pp. 2739–2756.

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency." In: *TCC 2008*. Vol. 4948. LNCS. 2008, pp. 1–18.

[YMT97] A. M. Youssef, S. Mister, and S. E. Tavares. "On the Design of Linear Transformations for Substitution Permutation Encryption Networks." In: *School of Computer Science, Carleton University*. 1997, pp. 40–48.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. "Caulk: Lookup Arguments in Sublinear Time." In: *CCS*. ACM, 2022, pp. 3121–3134.

[ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. *Baloo: Nearly Optimal Lookup Arguments*. Cryptology ePrint Archive. https://eprint.iacr.org/2022/1565. 2022.

[Zha22] Ye Zhang. *Introducing zkEVM*. https://monolith.scroll.io/blog/zkEVM. 2022.

[ZMI89] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses." In: *CRYPTO 1989*. Vol. 435. LNCS. 1989, pp. 461–480.

9

Multi-Party Revocation in Sovrin: Performance through Distributed Trust

Publication Data

Lukas Helminger, Daniel Kales, Sebastian Ramacher, and Roman Walch. "Multi-party Revocation in Sovrin: Performance through Distributed Trust." In: *CT-RSA*. Vol. 12704. Lecture Notes in Computer Science. Springer, 2021, pp. 527–551

The appended paper is the author's full version available at https://eprint.iacr.org/2020/724 which has been changed to use the design and formatting of this thesis. The full version includes the missing appendices from the conference version which consist of further definitions (such as ideal functionalities), proofs, and benchmarks.

Contributions

Main author.

Multi-Party Revocation in Sovrin: Performance through Distributed Trust

Lukas Helminger[1,2], Daniel Kales[1], Sebastian Ramacher[3], Roman Walch[1,2]

[1] Graz University of Technology, Graz, Austria
[2] Know-Center GmbH, Graz, Austria
[3] AIT Austrian Institute of Technology, Vienna, Austria

Abstract

Accumulators provide compact representations of large sets and compact membership witnesses. Besides constant-size witnesses, public-key accumulators provide efficient updates of both the accumulator itself and the witness. However, bilinear group based accumulators come with drawbacks: they require a trusted setup and their performance is not practical for real-world applications with large sets.

In this paper, we introduce multi-party public-key accumulators dubbed *dynamic (threshold) secret-shared accumulators*. We present an instantiation using bilinear groups having access to more efficient witness generation and update algorithms that utilize the shares of the secret trapdoors sampled by the parties generating the public parameters. Specifically, for the q-SDH-based accumulators, we provide a maliciously-secure variant sped up by a secure multi-party computation (MPC) protocol (IMACC'19) built on top of SPDZ and a maliciously secure threshold variant built with Shamir secret sharing. For these schemes, a performant proof-of-concept implementation is provided, which substantiates the practicability of public-key accumulators in this setting.

We explore applications of dynamic (threshold) secret-shared accumulators to revocation schemes of group signatures and credentials system. In particular, we consider it as part of Sovrin's system for anonymous credentials where credentials are issued by the foundation of trusted nodes.

1 Introduction

Digital identity management systems become an increasingly important corner stone of digital workflows. Self-sovereign identity (SSI) systems such as Sovrin[1] are of central interest as underlined by a recent push in the European Union for a cross-border SSI system.[2] But all these systems face a similar issue, namely that of efficient revocation. Regardless of whether they are built from signatures, group signatures or anonymous credentials, such systems have to consider mechanisms to revoke a user's identity information. Especially for identity management systems with a focus on privacy, revocation may threaten those privacy guarantees.

[1] https://sovrin.org/
[2] https://essif-lab.eu/

As such various forms of privacy-preserving revocations have emerged in the literature including approaches based on various forms of deny- or allowlists including [NFH+09; Ver16; BS01; BL11; BL12; ATS+09; CL02; FHM11; NKH+05; NS04; GGM14] among many others.[1]

One promising approach regarding efficiency is based on denylists (or allowlists) via cryptographic accumulators which were introduced by Benaloh and de Mare [BM93]. They allow one to accumulate a finite set $\mathcal{X}$ into a succinct value called the accumulator. For every element in this set, one can efficiently compute a witness certifying its membership, and additionally, some accumulators also support efficient non-memberships witnesses. However, it should be computationally infeasible to find a membership witness for non-accumulated values and a non-membership witness for accumulated values, respectively. Accumulators facilitate privacy-preserving revocation mechanisms, which is especially relevant for privacy-friendly authentication mechanisms like group signatures and credentials. For a denylist approach, the issuing authority accumulates all revoked users and users prove in zero-knowledge that they know a non-membership witness for their credential. Alternatively, for a allowlist approach, the issuing authority accumulates all users and users then prove in zero-knowledge that they know a membership witness. As both approaches may involve large lists, efficient accumulator updates as well as efficient proofs are important for building an overall efficient system. For example, in Sovrin [KL16] and Hyperledger Indy[2] such an accumulator-based approach with allowlists following the ideas of [GGM14] is used. Their credentials contain a unique revocation ID attribute, i_R, which are accumulated. Each user obtains a membership witness proving that their i_R is contained in the accumulator. Once a credential is revoked, the corresponding i_R gets removed from the accumulator and all users have to update their proofs accordingly. The revoked user is no longer able to prove knowledge of a verifying witness and thus verification fails.

Accumulators are an important primitive and building block in many cryptographic protocols. In particular, Merkle trees [Mer89] have seen many applications in both the cryptographic literature but also in practice. For example, they have been used to implement Certificate Transparency (CT) [Lau14] where all issued certificates are publicly logged, i.e., accumulated. Accumulators also find application in credentials [CL02], ring, and group signatures [LLN+16; DRS18], anonymous cash [MGG+13], authenticated hash tables [PTT08], among many others. When looking at accumulators deployed in practice, many systems rely on Merkle trees. Most prominently we can observe this fact in CT. Even though new certificates are continuously added to the log, the system is designed around a Merkle tree that gets recomputed all the time instead of updating a dynamic public-key accumulator. The reason is two-fold: first, for dynamic accumulators to be efficiently computable, knowledge of the secret trapdoor used to generate the public parameters is required. Without this information, witness generation and

[1] For a discussion of approaches for group signatures, see, e.g., [SSU16].

[2] https://hyperledger-indy.readthedocs.io/projects/hipe/en/latest/text/0011-cred-revocation/README.html

accumulator updates are simply too slow for large sets (cf. [KOR19]). Secondly, in this setting it is of paramount importance that the log servers do not have access to the secret trapdoor. Otherwise malicious servers would be able to present membership witnesses for every certificate even if it was not included in the log.

The latter issue can also be observed in other applications of public-key accumulators. The approaches due to Garman et al. [GGM14] and the one used in Sovrin rely on the Strong-RSA and q-SDH accumulators, respectively. Both these accumulators have trapdoors: in the first case the factorization of the RSA modulus and in the second case a secret exponent. Therefore, the security of the system requires those trapdoors to stay secret. Hence, these protocols require to put significant trust in the parties generating the public parameters. If they would act maliciously and not delete the secret trapdoors, they would be able to break all these protocols in one way or another. To circumvent this problem, Sander [San99] proposed a variant of an RSA-based accumulator from RSA moduli with unknown factorization. Alternatively, secure multi-party computation (MPC) protocols enable us to compute the public parameters and thereby replace the trusted third party. As long as a large enough subset of parties is honest, the secret trapdoor is not available to anyone. Over the years, efficient solutions for distributed parameter generation have emerged, e.g., for distributed RSA key generation [FLO+18; CCD+20; CHI+20], or distributed ECDSA key generation [LN18].

Based on the recent progress in efficient MPC protocols, we ask the following question: *what if the parties kept their shares of the secret trapdoor?* Are the algorithms of the public-key accumulators exploiting knowledge of the secret trapdoor faster if performed within an (maliciously-secure) MPC protocol than their variants relying only on the public parameters?

1.1 Our Techniques

We give a short overview of how our construction works which allows us to positively answer this question for accumulators in the discrete logarithm setting. Let us consider the accumulator based on the q-SDH assumption which is based on the fact that given powers $g^{s^i} \in \mathbb{G}$ for all i up to q where $s \in \mathbb{Z}_p$ is unknown, it is possible to evaluate polynomials $f \in \mathbb{Z}_p[X]$ up to degree q at s in the exponent, i.e., $g^{f(s)}$. This is done by taking the coefficients of the polynomial, i.e., $f = \sum_{i=0}^{q} a_i X^i$, and computing $g^{f(s)}$ as $\prod_{i=0}^{q} (g^{s^i})^{a_i}$. The accumulator is built by defining a polynomial with the elements as roots and evaluating this polynomial at s in the exponent. A witness is simply the corresponding factor canceled out, i.e., $g^{f(s)(s-x)^{-1}}$. Verification of the witness is performed by checking whether the corresponding factor and the witness match $g^{f(s)}$ using a pairing equation.

If s is known, all computations are more efficient: $f(s)$ can be directly evaluated in $\mathbb{Z}_p$ and the generation of the accumulator only requires one exponentiation in $\mathbb{G}$. The same is true for the computation of the witness. For the latter, the asymptotic runtime is thereby reduced from $\mathcal{O}(|\mathcal{X}|)$ to $\mathcal{O}(1)$. This improvement comes at a cost: if s is known, witnesses for non-members can be produced.

On the other hand, if multiple parties first produce s in an additively secret-shared fashion, these parties can cooperate in a secret-sharing based MPC protocol. Thereby, all the computations can still benefit from the knowledge of s. Indeed, the parties would compute their share of $g^{f(s)}$ and $g^{f(s)(s-x)^{-1}}$ respectively and thanks to the partial knowledge of s could still perform all operations – except the final exponentiation – in $\mathbb{Z}_p$. Furthermore, all involved computations are generic enough to be instantiated with MPC protocols with different trust assumptions. These include the dishonest majority protocol SPDZ [DKL+13; DPS+12] and honest majority threshold protocols based on Shamir secret sharing [Sha79]. While in SPDZ, an honest party can always detect malicious behavior, Shamir adds robustness against parties dropping out or failing to provide their shares.

1.2 Our Contribution

Starting from the very recent treatment of accumulators in the UC model [Can01] by Baldimtsi et al. [BCY20], we introduce the notion of *(threshold) secret-shared accumulators*. As the name suggests, it covers accumulators where the trapdoor is available in a (potentially full) threshold secret-shared fashion with multiple parties running the parameter generation as well as the algorithms that profit from the availability of the trapdoor. Since the MPC literature discusses security in the UC model, we also chose to do so for our accumulators.

Based on recent improvements on distributed key generation of discrete logarithms, we provide dynamic public-key accumulators without trusted setup. During the parameter generation, the involved parties keep their shares of the secret trapdoor. Consequently, we present MPC protocols secure in the semi-honest and the malicious security model, respectively, implementing the algorithms for accumulator generation, witness generation, and accumulator updates exploiting the shares of the secret trapdoor. Specifically, we give such protocols for q-SDH accumulators [Ngu05; DHS15], which can be build from dishonest-majority full-threshold protocols (e.g., SPDZ [DKL+13; DPS+12]) and from honest-majority threshold MPC protocols (e.g., Shamir secret sharing [Sha79]). In particular, our protocol enables updates to the accumulator independent of the size of the accumulated set. For increased efficiency, we consider this accumulator in bilinear groups of Type-3. Due to their structure, the construction nicely generalizes to any number of parties.

We provide a proof-of-concept implementation of our protocols in two MPC frameworks, MP-SPDZ [Kel20] and FRESCO.[1] We evaluate the efficiency of our protocols and compare them to the performance of an implementation, having no access to the secret trapdoors as usual for the public-key accumulators. We evaluate our protocol in the LAN and WAN setting in the semi-honest and malicious security model for various choices of parties and accumulator sizes. For the latter, we choose sizes up to 2^{14}. Specifically, for the q-SDH accumulator, we observe the expected $\mathcal{O}(1)$ runtimes for witness creation and accumulator updates, which cannot be achieved without access to the trapdoor. Notably, for the tested

[1] https://github.com/aicis/fresco

numbers of up to 5 parties, the MPC-enabled accumulator creation algorithms are faster for 2^{10} elements in the LAN setting than its non-MPC counterpart (without access to the secret trapdoor). For 2^{14} elements the algorithms are also faster in the WAN setting.

Finally, we discuss how our proposed MPC-based accumulators might impact revocation in distributed credential systems such as Sovrin [KL16]. In this scenario, the trust in the nodes run by the Sovrin foundation members can further be reduced. In addition, this approach generalizes to any accumulator-based revocation scheme and can be combined with threshold key management systems. We also discuss applications to CT and its privacy-preserving extension [KOR19]. In particular, the size of the witnesses stored in certificates or sent as part of the TLS handshake is significantly reduced without running into performance issues.

1.3 Related Work

When cryptographic protocols are deployed that require the setup of public parameters by a trusted third party, issues similar to those mentioned for public-key accumulators may arise. As discussed before, especially cryptocurrencies had to come up with ways to circumvent this problem for accumulators but also the common reference string (CRS) of zero-knowledge SNARKs [CGG+17]. Here, trust in the CRS is of paramount importance on the verifier side to prevent malicious provers from cheating. But also provers need to trust the CRS as otherwise zero-knowledge might not hold. We note that there are alternative approaches, namely subversion-resilient zk-SNARKS [BFS16] to reduce the trust required in the CRS generator. However, subversion-resilient soundness and zero-knowledge at the same time has been shown to be impossible by Bellare et al. Abdolmaleki et al. [ABL+17] provided a construction of zk-SNARKS, which was later improved by Fuchsbauer [Fuc18], achieving subversion zero-knowledge by adding a verification algorithm for the CRS. As a result, only the verifier needs to trust the correctness of the CRS. Groth et al. [GKM+18] recently introduced the notion of an updatable CRS where first generic compilers [ARS20] are available to lift any zk-SNARK to an updatable simulation sound extractable zk-SNARK. There the CRS can be updated and if the initial generation or one of the updates was done honestly, neither soundness nor zero-knowledge can be subverted. In the random oracle model (ROM), those considerations become less of a concern and the trust put into the CRS can be minimized, e.g., as done in the construction of STARKs [BBH+19].

Approaches that try to fix the issue directly in the formalization of accumulators and corresponding constructions have also been studied. For example, Lipmaa [Lip12] proposed a modified model tailored to the hidden order group setting. In this model, the parameter setup is split into two algorithms, Setup and Gen where the adversary can control the trapdoors output by Setup, but can neither influence nor access the randomness used by Gen. However, constructions in this model so far have been provided using assumptions based on modules over Euclidean rings, and are not applicable to the efficient standard constructions we are interested in. More recently, Boneh et al. [BBF19] revisited the RSA accu-

mulator without trapdoor which allows the accumulator to be instantiated from unknown order groups without trusted setup such as class groups of quadratic imaginary orders [HM00] and hyperelliptic curves of genus 2 or 3 [DG20].

The area of secure multiparty computation has seen a lot of interest both in improving the MPC protocols itself to a wide range of practical applications. In particular, SPDZ [DPS+12; DKL+13] has seen a lot of interest, improvements and extensions [KOS15; KOS16; CDE+18; OSV20]. This interest also led to multiple MPC frameworks, e.g., MP-SPDZ [Kel20], FRESCO and SCALE-MAMBA,[1] enabling easy prototyping for researchers as well as developers. For practical applications of MPC, one can observe first MPC-based systems turned into products such as Unbound's virtual hardware security model (HSM).[2] For such a virtual HSM, one essentially wants to provide distributed key generation [FLO+18] together with threshold signatures [DK01] allowing to replace a physical HSM. Similar techniques are also interesting for securing wallets for the use in cryptocurrencies, where especially protocols for ECDSA [GG18; LN18] are of importance to secure the secret key material. Similarly, such protocols are also of interest for securing the secret key material of internet infrastructure such as DNSSEC [DOK+20]. Additionally, addressing privacy concerns in machine learning algorithms has become increasingly popular recently, with MPC protocols being one of the building blocks to achieve private classification and private model training as in [WGC19] for example. Recent works [SA19] also started to generalize the algorithms that are used as parts of those protocols allowing group operations on elliptic curve groups with secret exponents or secret group elements.

2 Preliminaries

In this section, we introduce cryptographic primitives we use as building blocks. For notation and assumptions, we refer to Appendix A.

2.1 UC security and ABB

In this paper, we mainly work in the UC model first introduced by Canetti [Can01]. The success of the UC model stems from its universal composition theorem, which, informally speaking, states that it is safe to use a secure protocol as a sub-protocol in a more complex one. This strong statement enables one to analyze and proof the security of involved protocols in a modular way, allowing us to build upon work that was already proven to be secure in the UC model. In preparation for the security analysis of our MPC accumulators, we recall the definition of the UC model [Can01].

Definition 1. *Let* $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ *respectively* $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ *denote the random variables describing the output of environment* $\mathcal{E}$ *when interacting with an adversary* $\mathcal{A}$

[1] https://homes.esat.kuleuven.be/~nsmart/SCALE/
[2] https://www.unboundtech.com/usecase/virtual-hsm/

and parties performing protocol Π*, respectively when interacting with a simulator* $\mathcal{S}$ *and an ideal functionality* $\mathcal{F}$*. Protocol* Π *UC emulates the ideal functionality* $\mathcal{F}$ *if for any adversary* $\mathcal{A}$ *there exists a simulator* $\mathcal{S}$ *such that, for any environment* $\mathcal{E}$ *the distribution of* $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ *and* $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ *are indistinguishable.*

The importance of the UC model for secure multiparty computation stems from the arithmetic black box (ABB) as introduced by Damgård and Nielsen [DN03]. The ABB models a secure general-purpose computer in the UC model. It allows performing arithmetic operations on private inputs provided by the parties. The result of these operations is then revealed to all parties. Working with the ABB provides us with a tool of abstracting arithmetic operations, including addition and multiplication in fields.

2.2 SPDZ, Shamir, and Derived Protocols

Our protocols build upon SPDZ [DPS+12; DKL+13] and Shamir secret sharing [Sha79], concrete implementations of the abstract ABB. SPDZ itself is based on an additive secret-sharing over a finite field $\mathbb{F}_p$ with information-theoretic MACs making the protocol statistically UC secure against an active adversary corrupting all but one player. On the other hand, Shamir secret sharing is a threshold sharing scheme where $k \leq n$ out of n parties are enough to evaluate the protocol correctly. Therefore, it is naturally robust against parties dropping out during the computation; however, it assumes an honest-majority amongst all parties for security. Shamir secret sharing can be made maliciously UC secure in the honest-majority setting using techniques from [CGH+18] or [LN17].

We will denote the ideal functionality of the online protocol of SPDZ and Shamir secret sharing by $\mathcal{F}_{\text{Abb}}$. For an easy use of these protocols later in our accumulators, we give a high-level description of the functionality together with an intuitive notation. We assume that the computations are performed by n (or k) parties and we denote by $\langle s \rangle \in \mathbb{F}_p$ a secret-shared value between the parties in a finite field with p elements, where p is prime. The ideal functionality $\mathcal{F}_{\text{Abb}}$ provides us with the following basis operations: Addition $\langle a + b \rangle \leftarrow \langle a \rangle + \langle b \rangle$ (can be computed locally), multiplication $\langle ab \rangle \leftarrow \langle a \rangle \cdot \langle b \rangle$ (interactive 1-round protocol), sampling $\langle r \rangle \xleftarrow{R} \mathbb{F}_p$, and opening a share $\langle a \rangle$. For convenience, we assume that we have also access to the inverse function $\langle a^{-1} \rangle$. Computation of the inverse can be efficiently implemented using a standard form of masking as first done in [BB89]. Given an opening of $\langle z \rangle = \langle r \cdot a \rangle$, the inverse of $\langle a \rangle$ is then equal to $z^{-1}\langle r \rangle$. However, there is a small failure probability if either a or r is zero. In our case, the field size is large enough that the probability of a random element being zero is negligible.

There is one additional sub-protocol which we will often need. Recent work [SA19] introduced protocols – in particular based on SPDZ – for group operations of elliptic curve groups supporting secret exponents and secret group elements. The high-level idea is to use the original SPDZ in the exponent group and for the authentication of the shares of an elliptic curve point a similar protocol as in SPDZ. For this work, we only need the protocol for exponentiation

of a public point with a secret exponent. Let $\mathbb{G}$ be a cyclic group of prime order p and $g \in \mathbb{G}$. Further, let $\langle a \rangle \in \mathbb{F}_p$ be a secret-shared exponent.

$\mathsf{Exp}_{\mathbb{G}}(\langle a \rangle, g)$: The parties locally compute $\langle g^a \rangle \leftarrow g^{\langle a \rangle}$.

Since the security proof of this sub-protocol in [SA19] does not use any exclusive property of an elliptic curve group, it applies to any cyclic group of prime order.

All protocols discussed so far are secure in the UC model, making them safe to use in our accumulators as sub-protocols. Therefore, we will refer to their ideal functionality as $\mathcal{F}_{\text{ABB+}}$. As a result, our protocols become secure in the UC model as long as we do not reveal any intermediate values.

2.3 Accumulators

We rely on the formalization of accumulators by Derler et al. [DHS15]. We start with the definitions of static and dynamic accumulators.

Definition 2 (Static Accumulator)**.** *A static accumulator is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify})$ *which are defined as follows:*

$\mathsf{Gen}(1^\kappa, q)$: *This algorithm takes a security parameter* κ *and a parameter* q*. If* $q \neq \infty$*, then* q *is an upper bound on the number of elements to be accumulated. It returns a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, where* $\mathsf{sk}_\Lambda = \emptyset$ *if no trapdoor exists. We assume that the accumulator public key* pk_Λ *implicitly defines the accumulation domain* D_Λ*.*

$\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$ *and a set* $\mathcal{X}$ *to be accumulated and returns an accumulator* $\Lambda_\mathcal{X}$ *together with some auxiliary information* aux*.*

$\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x_i)$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, an accumulator* $\Lambda_\mathcal{X}$*, auxiliary information* aux *and a value* x_i*. It returns* $\perp$*, if* $x_i \notin \mathcal{X}$*, and a witness* wit_{x_i} *for* x_i *otherwise.*

$\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_{x_i}, x_i)$: *This algorithm takes a public key* pk_Λ*, an accumulator* $\Lambda_\mathcal{X}$*, a witness* wit_{x_i} *and a value* x_i*. It returns* true *if* wit_{x_i} *is a witness for* $x_i \in \mathcal{X}$ *and* false *otherwise.*

Definition 3 (Dynamic Accumulator)**.** *A dynamic accumulator is a static accumulator with PPT algorithms* $(\mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate})$ *defined as follows:*

$\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, an accumulator* $\Lambda_\mathcal{X}$*, auxiliary information* aux*, as well as an element* x *to be added. If* $x \in \mathcal{X}$*, it returns* $\perp$*. Otherwise, it returns the updated accumulator* $\Lambda_{\mathcal{X}'}$ *with* $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$ *and updated auxiliary information* aux'*.*

$\mathsf{Delete}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, an accumulator* $\Lambda_\mathcal{X}$*, auxiliary information* aux*, as well as an element* x *to be added. If* $x \notin \mathcal{X}$*, it returns* $\perp$*. Otherwise, it returns the updated accumulator* $\Lambda_{\mathcal{X}'}$ *with* $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$ *and updated auxiliary information* aux'*.*

$\mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)$: *This algorithm takes a key pair* $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda)$*, a witness* wit_{x_i} *to be updated, auxiliary information* aux *and an* x *which was added to/deleted from the accumulator, where* aux *indicates addition or deletion. It returns an updated witness* wit'_{x_i} *on success and* $\perp$ *otherwise.*

This formalization of accumulators gives access to a trapdoor if it exists and sk_Λ is set to $\emptyset$ if it is not available. We recall collision freeness:

Definition 4 (Collision Freeness)**.** *A cryptographic accumulator is* collision-free*, if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\varepsilon(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{l}(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda) \leftarrow \mathsf{Gen}(1^\kappa, q), (\mathsf{wit}_{x_i}{}^\star, x_i{}^\star, \mathcal{X}^\star, r^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{pk}_\Lambda) : \\ \mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda^\star, \mathsf{wit}_{x_i}{}^\star, x_i{}^\star) = \mathsf{true} \wedge x_i{}^\star \notin \mathcal{X}^\star\end{array}\right] \leq \varepsilon(\kappa),$$

where $\Lambda^\star \leftarrow \mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X}^\star; r^\star)$ *and the adversary gets access to the oracles* $\mathcal{O} = \{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot), \mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot)\}$ *and, if the accumulator is dynamic, additionally to* $\{\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot), \mathsf{Delete}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot), \mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \cdot, \cdot, \cdot)\}$.

2.4 Pairing-based Accumulator

We recall the q-SDH-based accumulator from [DHS15], which is based on the accumulator by Nguyen [Ngu05]. The idea here is to encode the accumulated elements in a polynomial. This polynomial is then evaluated for a fixed element and the result is randomized to obtain the accumulator. A witness consists of the evaluation of the same polynomial with the term corresponding to the respective element cancelled out. For verification, a pairing evaluation is used to check whether the polynomial encoded in the witness is a factor of the one encoded in the accumulator. As it is typically more efficient to work with bilinear groups of Type-3 [GPS08], we state the accumulator as depicted in Scheme 1 in this setting. Correctness is clear, except for the $\mathsf{WitUpdate}$ subroutine: To update witness wit_{x_i} of the element x_i after the element x was added to the accumulator $\Lambda_\mathcal{X}$ to create the new accumulator $\Lambda_{\mathcal{X}'} = \Lambda_\mathcal{X}^{(x+s)}$, one computes:

$$\begin{aligned}\Lambda_\mathcal{X} \cdot \mathsf{wit}_{x_i}^{(x-x_i)} &= \Lambda_\mathcal{X}^{(x_i+s)\cdot(x_i+s)^{-1}} \cdot \Lambda_\mathcal{X}^{(x-x_i)\cdot(x_i+s)^{-1}} \\ &= \Lambda_\mathcal{X}^{(x+s)\cdot(x_i+s)^{-1}} = \Lambda_{\mathcal{X}'}^{(x_i+s)^{-1}}\end{aligned}$$

which results in the desired updated witness. Similar, if the element x gets removed instead, one computes the following to get the desired witness:

$$\begin{aligned}(\Lambda_{\mathcal{X}'}^{-1} \cdot \mathsf{wit}_{x_i})^{(x-x_i)^{-1}} &= \Lambda_{\mathcal{X}'}^{-(x_i+s)\cdot(x_i+s)^{-1}\cdot(x-x_i)^{-1}} \cdot \Lambda_{\mathcal{X}'}^{(x+s)\cdot(x_i+s)^{-1}\cdot(x-x_i)^{-1}} \\ &= \Lambda_{\mathcal{X}'}^{(x_i+s)^{-1}\cdot(x-x_i)^{-1}\cdot(x-x_i+s-s)} = \Lambda_{\mathcal{X}'}^{(x_i+s)^{-1}}\end{aligned}$$

The proof of collision freeness follows from the q-SDH assumption. For completeness, we still restate the theorem from [DHS15] adopted to the Type-3 setting.

$\underline{\mathsf{Gen}(1^\kappa, q)}$: Let $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BGen}(n)$. Choose $s \xleftarrow{R} \mathbb{Z}_p^*$ and return $\mathsf{sk}_\Lambda \leftarrow s$ and $\mathsf{pk}_\Lambda \leftarrow (\mathsf{BG}, (g_1^{s^i})_{i=1}^q, g_2^s)$.

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse $\mathcal{X} \subset \mathbb{Z}_p^*$. Choose $r \xleftarrow{R} \mathbb{Z}_p^*$. If $\mathsf{sk}_\Lambda \neq \emptyset$, compute $\Lambda_\mathcal{X} \leftarrow g_1^{r\prod_{x\in\mathcal{X}}(x+s)}$. Otherwise, expand the polynomial $\prod_{x\in\mathcal{X}}(x+X) = \sum_{i=0}^n a_i X^i$, and compute $\Lambda_\mathcal{X} \leftarrow ((\prod_{i=0}^n g_1^{s^i})^{a_i})^r$. Return $\Lambda_\mathcal{X}$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0, r, \mathcal{X})$.

$\underline{\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse aux as $(r, \mathcal{X})$. If $x \notin \mathcal{X}$, return $\perp$. If $\mathsf{sk}_\Lambda \neq \emptyset$, compute and return $\mathsf{wit}_x \leftarrow \Lambda_\mathcal{X}^{(x+s)^{-1}}$. Otherwise, run $(\mathsf{wit}_x, \ldots) \leftarrow \mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X} \setminus \{x\}; r)$, and return wit_x.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Return 1 if $e(\Lambda_\mathcal{X}, g_2) = e(\mathsf{wit}_x, g_2^x \cdot g_2^s)$, otherwise return 0.

$\underline{\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse aux as $(r, \mathcal{X})$. If $x \in \mathcal{X}$, return $\perp$. Set $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$. If $\mathsf{sk}_\Lambda \neq \emptyset$, compute and return $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^{x+s}$ and $\mathsf{aux}' \leftarrow (r, \mathcal{X}', \mathsf{add} \leftarrow 1, \Lambda_\mathcal{X}, \Lambda_{\mathcal{X}'})$. Otherwise, return $\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X}'; r)$ with aux extended with $(\mathsf{add} \leftarrow 1, \Lambda_\mathcal{X}, \Lambda_{\mathcal{X}'})$.

$\underline{\mathsf{Delete}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse aux as $(r, \mathcal{X})$. If $x \notin \mathcal{X}$, return $\perp$. Set $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$. If $\mathsf{sk}_\Lambda \neq \emptyset$, compute and return $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^{(x+s)^{-1}}$ and $\mathsf{aux}' \leftarrow (r, \mathcal{X}', \mathsf{add} \leftarrow -1, \Lambda_\mathcal{X}, \Lambda_{\mathcal{X}'})$. Otherwise, return $\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X}'; r)$ with aux extended with $(\mathsf{add} \leftarrow 0, \Lambda_\mathcal{X}, \Lambda_{X'})$.

$\underline{\mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)}$: Parse aux as $(\perp, \perp, \mathsf{add}, \Lambda_\mathcal{X}, \Lambda_{\mathcal{X}'})$. If $\mathsf{add} = 0$, return $\perp$. Return $\Lambda_\mathcal{X} \cdot \mathsf{wit}_{x_i}^{x-x_i}$ if $\mathsf{add} = 1$. If instead $\mathsf{add} = -1$, return $(\Lambda_{\mathcal{X}'}^{-1} \cdot \mathsf{wit}_{x_i})^{(x-x_i)^{-1}}$. In the last two cases in addition return $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0)$.

Scheme 1: q-SDH-based accumulator in the Type-3 setting.

Theorem 1. *If the q-SDH assumption holds, then Scheme 1 is collision-free.*

Proof 1. *Assume that $\mathcal{A}$ is an adversary against the collision freeness of the accumulator. We show that this adversary can be transformed into an efficient adversary $\mathcal{B}$ against the q-SDH assumption. We perform a proof by reduction in the following way:*

- *When $\mathcal{B}$ is started on a q-SDH instance $\left(\mathsf{BG}, \left(g_1^{s^i}\right)_{i\in[q]}, g_2^s\right)$, set $\mathsf{pk}_\Lambda \leftarrow \left(\mathsf{BG}, \left(g_1^{s^i}\right)_{i\in[q]}, g_2^s\right)$ and start $\mathcal{A}$ on pk_Λ. The oracles for $\mathcal{A}$ are simulated by forwarding the inputs directly to the corresponding algorithms with $(\emptyset, \mathsf{pk}_\Lambda)$ as argument for the keys.*

$\mathcal{O}^{\mathsf{Eval}}(\mathcal{X})$: *Return* $\mathsf{Eval}((\emptyset, \mathsf{pk}_\Lambda), \mathcal{X})$.
$\mathcal{O}^{\mathsf{WitCreate}}(\Lambda_\mathcal{X}, \mathsf{aux}, x)$: *Return* $\mathsf{WitCreate}((\emptyset, \mathsf{pk}_\Lambda), \mathcal{X})$.
$\mathcal{O}^{\mathsf{Add}}(\Lambda_\mathcal{X}, \mathsf{aux}, x)$: *Return* $\mathsf{Add}((\emptyset, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
$\mathcal{O}^{\mathsf{Delete}}(\Lambda_\mathcal{X}, \mathsf{aux}, x)$: *Return* $\mathsf{Delete}((\emptyset, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
$\mathcal{O}^{\mathsf{WitUpdate}}(\mathsf{wit}_{x'}, \mathsf{aux}, x)$: *Return* $\mathsf{WitUpdate}((\emptyset, \mathsf{pk}_\Lambda), \mathsf{wit}_{x'}, \mathsf{aux}, x)$.

- *At some point $\mathcal{A}$ outputs a set $\mathcal{X}^\star$, an element $x^\star \notin \mathcal{X}^\star$, a witness $\mathsf{wit}^\star_{x^\star}$ for $x^\star$, and the randomizer $r^\star$, such that for $\Lambda_{\mathcal{X}^\star}, \mathsf{aux} \leftarrow \mathsf{Eval}((\emptyset, \mathsf{pk}_\Lambda), \mathcal{X}^\star)$, the verification relation $e(\Lambda_{\mathcal{X}^\star}, g_2) = e(\mathsf{wit}^\star_{x^\star}, g_2^{x^\star} \cdot g_2^s)$ holds. Now, compute $h(X) = \prod_{x \in \mathcal{X}^\star}(x+X)$ and $\phi(X)$ such that $h(X) = \phi(X)(x^\star+X)+d$, which exists since $x^\star \notin \mathcal{X}^\star$. Then compute $g_1^{r^\star \phi(s)}$ by expanding the polynomial $\phi(X)$ and the $g_1^{s^i}$ stored in pk_Λ. Then, $\mathcal{B}$ outputs*

$$\left(\mathsf{wit}^\star_{x^\star} \cdot \left(g_1^{r^\star \phi(s)}\right)^{-1}\right)^{\frac{1}{r^\star d}} = \left(g_1^{\frac{r^\star h(s)}{x^\star+s}} \cdot g_1^{\frac{-r^\star(h(s)-d)}{x^\star+s}}\right)^{\frac{1}{r^\star d}}$$
$$= \left(g_1^{\frac{r^\star d}{x^\star+s}}\right)^{\frac{1}{r^\star d}} = g_1^{\frac{1}{x^\star+s}}$$

and $x^\star$ as solution to the q-SDH problem instance.

Hence, $\mathcal{B}$ succeeds with the same probability as $\mathcal{A}$. □

Remark 13. *Note that for support of arbitrary accumulation domains, the accumulator requires a suitable hash function mapping to $\mathbb{Z}_p^*$. For the MPC-based accumulators that we will define later, it is clear that the hash function can be evaluated in public. For simplicity, we omit the hash function in our discussion.*

2.5 UC Secure Accumulators

Only recently, Baldimtsi et al. [BCY20] formalized the security of accumulators in the UC framework. Interestingly, they showed, that any correct and collision-free standard accumulator is automatically UC secure. We, however, want to note, that their definitions of accumulators are slightly different then the framework by Derler et al. (which we are using). Hence, we adapt the ideal functionality $\mathcal{F}_{\text{Acc}}$ from [BCY20] to match our setting: First our ideal functionality $\mathcal{F}_{\text{Acc}}$ consists of two more sub-functionalities. This is due to a separation of the algorithms responsible for the evaluation, addition, and deletion. Secondly, our $\mathcal{F}_{\text{Acc}}$ is simplified to our purpose, whereas $\mathcal{F}_{\text{Acc}}$ from Baldimtsi et al. is in their words "an entire menu of functionalities covering all different types of accumulators". Thirdly, we added identity checks to sub-functionalities (where necessary) to be consistent with the given definitions of accumulators.

The resulting ideal functionality is depicted in Functionality 1 in Appendix C. Note that the ideal functionality has up to three parties. First, the party which holds the set $\mathcal{X}$ is the accumulator manager $\mathcal{AM}$, responsible for the algorithms $\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Add}$ and Delete. The second party $\mathcal{H}$ owns a witness and

is interested in keeping it updated and for this reason, performs the algorithm WitUpdate. The last party $\mathcal{V}$ can be seen as an external party. $\mathcal{V}$ is only able to use Verify to check the membership of an element in the accumulated set.

In the following theorem we adapt the proof from [BCY20] to our setting:

Theorem 2. *If* $\Pi_{Acc} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify}, \mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate})$ *is a correct and collision-free dynamic accumulator with deterministic* Verify*, then* Π_{Acc} *UC emulates* $\mathcal{F}_{Acc}$*.*

Proof 2. *We will proceed by contraposition. Assume to the contrary that* Π_{Acc} *does not UC emulate* $\mathcal{F}_{Acc}$*, i.e., there exists an environment* $\mathcal{E}$*, for all simulators* $\mathcal{S}$ *such that* $\mathcal{E}$ *can distinguish between the distributions of the random variables* $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ *and* $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ *with non-negligible probability. Since the last statement holds for all simulators, we can choose one. We want a simulator* $\mathcal{S}$ *that interacts with the ideal functionality* $\mathcal{F}_{Acc}$ *in a way such that their distribution can not be distinguished by any environment from the real world, except when it violates either correctness or collision-freeness. Such a simulator would be a contradiction to our assumption and thereby prove the theorem.*

Consider a simulator $\mathcal{S}$ *that uses the standard corruption model from [Can01]. Further,* $\mathcal{S}$ *interacts with the environment* $\mathcal{E}$ *by forwarding any input to the real adversary* $\mathcal{A}$ *and conversely forwarding any output from* $\mathcal{A}$ *directly to* $\mathcal{E}$*. When* $\mathcal{S}$ *receives the request (GEN,sid) from* $\mathcal{F}_{Acc}$*, it replies with the actual accumulator algorithms. By construction of* $\mathcal{S}$*, the only differences to the real world that are visible for the environment* $\mathcal{E}$ *are the following instances where* $\mathcal{F}_{Acc}$ *returns* $\perp$*: (i)* WitCreate*: 4., (ii)* Verify*: 1.b, (iii)* Add*: 6. and (iv)* WitUpdate *: 3. The occurrence of one of the above cases would immediately imply a violation of the classical definition. More concretely, if* Verify *1.b would return* $\perp$*, then the collision-freeness would be violated. In the other instances, correctness would not be given any more.* □

As a direct consequence of Theorems 1 and 2, the accumulator from Scheme 1 is also secure in the UC model of [BCY20] since it is correct and collision-free:

Corollary 3. *Scheme 1 emulates* $\mathcal{F}_{Acc}$ *in the UC model.*

3 Multi-Party Public-Key Accumulators

With the building blocks in place, we are now able to go into the details of our construction. We first present the formal notion of (threshold) secret-shared accumulators, their ideal functionality, and then present our constructions.

For the syntax of the MPC-based accumulator, which we dub *(threshold) secret-shared accumulator*, we use the bracket notation $\langle s \rangle$ from Section 2.2 to denote a secret shared value. If we want to explicitly highlight the different shares, we write $\langle s \rangle = (s_1, \ldots, s_n)$, where the share s_i belongs to a party P_i. We base the definition on the framework of Derler et al. [DHS15], where our algorithms behave in the same way, but instead of taking an optional secret trapdoor, the algorithms are given shares of the secret as input. Consequently, Gen outputs

shares of the secret trapdoor instead of the secret key. The static version of the accumulator is defined as follows:

Definition 5 (Static (Threshold) Secret-Shared Accumulator)**.** *Let us assume that we have a (threshold) secret sharing-scheme. A static (threshold) secret-shared accumulator for $n \in \mathbb{N}$ parties $P_1, \ldots, P_n$ is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify})$ *which are defined as follows:*

$\mathsf{Gen}(1^\kappa, q)$: *This algorithm takes a security parameter κ and a parameter q. If $q \neq \infty$, then q is an upper bound on the number of elements to be accumulated. It returns a key pair $(\mathsf{sk}_\Lambda^i, \mathsf{pk}_\Lambda)$ to each party P_i such that $\mathsf{sk}_\Lambda = \mathsf{Open}(\mathsf{sk}_\Lambda^1, \ldots, \mathsf{sk}_\Lambda^n)$, denoted by $\langle \mathsf{sk}_\Lambda \rangle$. We assume that the accumulator public key pk_Λ implicitly defines the accumulation domain D_Λ.*

$\mathsf{Eval}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \mathcal{X})$: *This algorithm takes a secret-shared private key $\langle \mathsf{sk}_\Lambda \rangle$ a public key pk_Λ and a set $\mathcal{X}$ to be accumulated and returns an accumulator $\Lambda_\mathcal{X}$ together with some auxiliary information* aux *to every party P_i.*

$\mathsf{WitCreate}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$: *This algorithm takes a secret-shared private key $\langle \mathsf{sk}_\Lambda \rangle$ a public key pk_Λ, an accumulator $\Lambda_\mathcal{X}$, auxiliary information* aux *and a value x. It returns $\perp$, if $x \notin \mathcal{X}$, and a witness wit_x for x otherwise to every party P_i.*

$\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)$: *This algorithm takes a public key pk_Λ, an accumulator $\Lambda_\mathcal{X}$, a witness wit_x and a value x. It returns* true *if wit_x is a witness for $x \in \mathcal{X}$ and* false *otherwise.*

In analogy to the non-interactive case, dynamic accumulators provide additional algorithms to add elements to the accumulator and remove elements from it, respectively, and update already existing witnesses accordingly. These algorithms are defined as follows:

Definition 6 (Dynamic (Threshold) Secret-Shared Accumulator)**.** *A dynamic (threshold) secret-shared accumulator is a static (threshold) secret-shared accumulator with an additional tuple of PPT algorithms* $(\mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate})$ *which are defined as follows:*

$\mathsf{Add}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$: *This algorithm takes a secret-shared private key $\langle \mathsf{sk}_\Lambda \rangle$ a public key pk_Λ, an accumulator $\Lambda_\mathcal{X}$, auxiliary information* aux*, as well as an element x to be added. If $x \in \mathcal{X}$, it returns $\perp$ to every party P_i. Otherwise, it returns the updated accumulator $\Lambda_{\mathcal{X}'}$ with $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$ and updated auxiliary information* aux' *to every party P_i.*

$\mathsf{Delete}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)$: *This algorithm takes a secret-shared private key $\langle \mathsf{sk}_\Lambda \rangle$ a public key pk_Λ, an accumulator $\Lambda_\mathcal{X}$, auxiliary information* aux*, as well as an element x to be added. If $x \notin \mathcal{X}$, it returns $\perp$ to every party P_i. Otherwise, it returns the updated accumulator $\Lambda_{\mathcal{X}'}$ with $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$ and updated auxiliary information* aux' *to every party P_i.*

$\mathsf{WitUpdate}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)$: *This algorithm takes a secret-shared private key* $\langle \mathsf{sk}_\Lambda \rangle$ *a public key* pk_Λ*, a witness* wit_{x_i} *to be updated, auxiliary information* aux *and an element* x *which was added to/deleted from the accumulator, where* aux *indicates addition or deletion. It returns an updated witness* wit'_{x_i} *on success and* $\perp$ *otherwise to every party* P_i*.*

Correctness and collision-freeness naturally translate from the non-interactive accumulators to the (threshold) secret-shared ones. The work of Baldimtsi et al. also introduced the property creation-correctness. Informally speaking, creation-correctness allows the generation of witnesses during addition. In the above definitions, we see that adding an element to the accumulator and creating a witness are two separate algorithms. Therefore, the notion of creation-correctness does not immediately apply to our accumulators.

For our case, the ideal functionality for (threshold) secret-shared accumulators, dubbed $\mathcal{F}_{\text{MPC-Acc}}$ is more interesting. $\mathcal{F}_{\text{MPC-Acc}}$ is very similar to $\mathcal{F}_{\text{Acc}}$ and is depicted in Functionality 2 in Appendix C. The only difference in describing the ideal functionality for accumulators in the MPC setting arises from the fact that we now have not only one accumulator manager but n, denoted by $\mathcal{AM}_1, \ldots, \mathcal{AM}_n$. More concretely, whenever a sub-functionality of $\mathcal{F}_{\text{MPC-Acc}}$ – that makes use of the secret key – gets a request from a manager identity $\mathcal{AM}_i$, it now also gets a participation message from the other managers identities A_j for $j \neq i$. Furthermore, the accumulator managers take the role of the witness holder. The party $\mathcal{V}$, however, stays unchanged.

3.1 Dynamic (Threshold) Secret-Shared Accumulator from the q-SDH Assumption

For the generation of public parameters Gen, we can rely on already established methods to produce ECDSA key pairs and exponentiations with secret exponents, respectively. These methods can directly be applied to the accumulators. Taking the q-SDH accumulator as an example, the first step is to sample the secret scalar $s \in \mathbb{Z}_p$. Intuitively, each party samples its own share s_i and the secret trapdoor s would then be $s = \mathsf{Open}(s_1, \ldots, s_n)$. The next step, the calculation of the basis elements g^{s^j} for $j = 1, \ldots, q$, is optional, but can be performed to provide public parameters, that are useful even to parties without knowledge of s. All of these elements can be computed using $\mathsf{Exp}_{\mathbb{G}}$ and the secret-shared s, respectively its powers. For the accumulator evaluation, Eval, the parties first sample their shares of r. Then, they jointly compute shares of $r \cdot f(s)$ using their shares of r and s. The so-obtained exponent and $\mathsf{Exp}_{\mathbb{G}}$ produce the final result.

For witness creation, WitCreate, it gets more interesting. Of course, one could simply run Eval again with one element removed from the set. In this case, we can do better, though. The difference between the accumulator and a witness is that in the latter, one factor of the polynomial is canceled. Since s is available, it is thus possible to cancel this factor without recomputing the polynomial from the start. Indeed, to compute the witness for an element x, we can compute $(s + x)^{-1}$ and then apply that inverse using $\mathsf{Exp}_{\mathbb{G}}$ to the accumulator to get the

witness. Note though, that before the parties perform this step, they need to check if x is actually contained in $\mathcal{X}$. Otherwise, they would produce a membership witness for a non-member. In that case, the verification would check whether $f(s)(s+x)^{-1}(s+x)$ matches $f(s)$, which of course also holds even if $s+x$ is not a factor of $f(s)$. In contrast, when performing Eval with only the publicly available information, this issue does not occur since there the witness will not verify. Add and Delete can be implemented in a similar manner. When adding an element to the accumulator, the polynomial is extended by one factor. Removal of an element requires that one factor is canceled. Both operations can be performed by first computing the factor using the shares of s and then running $\mathsf{Exp}_{\mathbb{G}}$.

Now, we present the MPC version of the q-SDH accumulator in Scheme 2 following the intuition outlined above. Note, that the algorithm for WitUpdate is unlikely to be faster than its non-MPC version from Scheme 1. Indeed, the non-MPC version requires only exponentiations in $\mathbb{G}_1$ and a multiplication without the knowledge of the secret trapdoor. We provide the version using the trapdoor for completeness but will use the non-MPC version of the algorithm in practical implementations. Note further that we let Gen choose the bilinear group BG, but this group can already be fixed a priori.

Theorem 3. *Scheme 2 UC emulates $\mathcal{F}_{Acc\text{-}MPC}$ in the $\mathcal{F}_{ABB+}$-hybrid model.*

Proof 3. *At this point, we make use of the UC model. Informally speaking, accumulators are UC secure, and SPDZ, Shamir secret sharing, and the derived operations UC emulate $\mathcal{F}_{ABB+}$. Therefore, according to the universal composition theorem, the use of these MPC protocols in the accumulator Scheme 2 can be done without losing UC security. For a better understanding, we begin by showing the desired accumulator properties for Scheme 2.*

The proof of the correctness follows directly from the correctness proof from Scheme 1 for the case where the secret key is known. Collision-freeness is also derived from the non-interactive q-SDH accumulator. (It is true that now each party has a share of the trapdoor, but without the other shares no party can create a valid witness.) Since Verify *is obviously deterministic, Scheme 2 fulfills all necessary assumption of Theorem 2. After applying Theorem 2, we get a simulator $\mathcal{S}_{Acc}$ interacting with the ideal functionality $\mathcal{F}_{Acc}$. Since we now also have to simulate the non-interactive sub-protocols, we have to extend $\mathcal{S}_{Acc}$. We construct $\mathcal{S}_{Acc\text{-}MPC}$ by building upon $\mathcal{S}_{Acc}$ and in addition internally simulate $\mathcal{F}_{ABB+}$. As described in Section 2.2, the MPC protocols used in the above algorithms are all secure in the UC model. Since we do not open any secret-shared values besides uniformly random elements and the output or values that can be immediately derived from the output, the algorithms are secure due to the universal composition theorem.* □

Remark 14. *In* Gen *of Scheme 2 we explicitly do not compute $h_i \leftarrow g_1^{s^i}$. Hence, using* Eval *without access to s is not possible. But, on the positive side, the public parameters are significantly smaller and so is the runtime of the* Gen *algorithm. If, however, these values are needed to support a non-secret-shared* Eval*, one can modify* Gen *to also compute the following values: $\langle t_1 \rangle \leftarrow \langle s \rangle$, $\langle t_i \rangle \leftarrow \langle t_{i-1} \rangle \cdot \langle s \rangle$,*

$\underline{\mathsf{Gen}(1^\kappa, q)}$: $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BGen}(n)$. Compute $\langle \mathsf{sk}_\Lambda \rangle \leftarrow \mathsf{sRand}(\mathbb{Z}_p^*)$. Compute $h \leftarrow \mathsf{Open}(g_2^{\langle \mathsf{sk}_\Lambda \rangle})$. Return $\mathsf{pk}_\Lambda \leftarrow (\mathsf{BG}, h)$.

$\underline{\mathsf{Eval}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as (BG, h) and $\mathcal{X}$ as subset of $\mathbb{Z}_p^*$. Choose $\langle r \rangle \leftarrow \mathsf{sRand}(\mathbb{Z}_p^*)$. Compute $\langle q \rangle \leftarrow \prod_{x \in \mathcal{X}} (x + \langle \mathsf{sk}_\Lambda \rangle) \in \mathbb{Z}_p^*$ and $\langle t \rangle \leftarrow \langle q \rangle \cdot \langle r \rangle$. The algorithm returns $\Lambda_\mathcal{X} \leftarrow \mathsf{Open}(g_1^{\langle t \rangle})$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0, \mathcal{X})$.

$\underline{\mathsf{WitCreate}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Returns $\perp$ if $x \notin \mathcal{X}$. Otherwise, $\langle z \rangle \leftarrow \langle (x + \langle \mathsf{sk}_\Lambda \rangle)^{-1} \rangle$. Return $\mathsf{wit}_x \leftarrow \mathsf{Open}(\Lambda_\mathcal{X}^{\langle z \rangle})$.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Parse pk_Λ as (BG, h). If $e(\Lambda_\mathcal{X}, g_2) = e(\mathsf{wit}_x, g_2^x \cdot h)$ holds, return 1, otherwise return 0.

$\underline{\mathsf{Add}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Returns $\perp$ if $x \in \mathcal{X}$. Otherwise set $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$. Return $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^x \cdot \mathsf{Open}(\Lambda_\mathcal{X}^{\langle \mathsf{sk}_\Lambda \rangle})$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 1, \mathcal{X}')$.

$\underline{\mathsf{Delete}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: If $x \notin \mathcal{X}$, return $\perp$. Otherwise set $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$, and compute $\langle y \rangle \leftarrow \langle (x + \langle \mathsf{sk}_\Lambda \rangle)^{-1} \rangle$. Return $\Lambda_{\mathcal{X}'} \leftarrow \mathsf{Open}(\Lambda_\mathcal{X}^{\langle y \rangle})$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow -1, \mathcal{X}')$.

$\underline{\mathsf{WitUpdate}((\langle \mathsf{sk}_\Lambda \rangle, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)}$: Parse aux as $(\mathsf{add}, \mathcal{X})$. Return $\perp$ if $\mathsf{add} = 0$ or $x_i \notin \mathcal{X}$. In case $\mathsf{add} = 1$, return $\mathsf{wit}_{x_i} \leftarrow \mathsf{wit}_{x_i}^x \cdot \mathsf{Open}(\mathsf{wit}_{x_i}^{\langle \mathsf{sk}_\Lambda \rangle})$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0, \mathcal{X})$. If instead $\mathsf{add} = -1$, it compute $\langle y \rangle \leftarrow \langle (x + \langle \mathsf{sk}_\Lambda \rangle)^{-1} \rangle$. Return $\mathsf{wit}_{x_i} \leftarrow \mathsf{Open}(\mathsf{wit}_{x_i}^{\langle y \rangle})$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0, \mathcal{X})$.

Scheme 2: MPC-q-SDH: Dynamic (threshold) secret-shared accumulator from q-SDH for $n \geq 2$ parties.

and $h_i \leftarrow \mathsf{Open}(g_1^{\langle t_i \rangle})$ *for* $i = 1, \ldots, t$. *This opens up the possibilities to trade an efficient* Eval *computation with an inefficient* Gen *step, which could be precomputed before the actual accumulator is created. Updates to this accumulator then still profit from the efficiency of the secret shared trapdoor. Additionally,* q *gives an upper bound on the size of the accumulated sets, and thus needs to be considered in the selection of the curves even though the powers of* g_1 *are not placed in the public key.*

3.2 SPDZ vs. Shamir Secret Sharing

In this section, we want to compare two MPC protocols on which our MPC-q-SDH Accumulator can be based on, namely SPDZ and Shamir secret sharing. Both protocols allow us to keep shares of the secret trapdoor and improve performance compared to the keyless q-SDH Accumulator. However, in relying on these protocols for security, the trust assumptions of the MPC-q-SDH Accumulator also have to include the underlying protocols' trust-assumptions.

SPDZ is a full-threshold dishonest-majority protocol that protects against $n-1$ corrupted parties. Therefore, an honest party will always detect malicious behavior. However, full-threshold schemes are not robust; if one party fails to supply its shares, the computation always fails.

On the contrary, Shamir secret sharing is an honest-majority threshold protocol. It is more robust than SPDZ since it allows $k \leq \frac{n-1}{2}$ corrupted parties while still being capable of providing correct results. This also means, if some parties ($\mathrm{k} \leq \frac{n-1}{2}$) fail to provide their shares, the other parties can still compute the correct results without them. Thus, no accumulator manager on its own is a single point of failure. However, if more than k parties are corrupted, the adversaries can reconstruct the secret trapdoor and, therefore, compromise the security of our MPC-q-SDH Accumulator.

4 Implementation and Performance Evaluation

We implemented the proposed dynamic (threshold) secret-shared accumulator from q-SDH and evaluated it against small to large sets.[1] Our primary implementations are based on SPDZ with OT-based preprocessing and Shamir secret sharing in the MP-SPDZ [Kel20][2] framework. However, to demonstrate the usability of our accumulator, we additionally build an implementation in the malicious security setting with dishonest-majority based on the FRESCO framework. We discuss the benchmarks for the MP-SPDZ implementation in this section. For a discussion of the FRESCO benchmarks we refer the reader to Appendix E.

Remark 15. *We want to note, that in our benchmarks we test the performance of the MPC variant of* WitUpdate *from Scheme 2, even though in practice the non-MPC variant from scheme Scheme 1 should be used.*

MP-SPDZ implements the SPDZ protocol with various extensions [DPS+12; KOS15; KOS16; CDE+18], as well as semi-honest and malicious variants of Shamir secret sharing [CDM00; CGH+18; LN17]. For pairing and elliptic curve group operations, we rely on relic[3] and integrate $\mathsf{Exp}_{\mathbb{G}}$, Output-$\mathbb{G}$, and the corresponding operations to update the MAC described in [SA19] into MP-SPDZ. We use the pairing friendly BLS12-381 curve [BLS02], which provides around 120 bit of security following recent estimates [BD19]. We want to note, that our implementation can easily be adapted to support other pairing libraries, as well. For completeness, we also implemented the q-SDH accumulator from Scheme 1 and a Merkle-tree accumulator (cf. Appendix B) using SHA-256. This enables us to compare the performance in cases where the secret trapdoors are available in the MPC case and when they are not. In Table 9.1, we present the numbers for various sizes of accumulated sets.

The evaluation of the MPC protocols was performed on a cluster with a Xeon E5-4669v4 CPU, where each party was assigned only 1 core. The hosts were

[1] The source code is available at https://github.com/IAIK/MPC-Accumulator.
[2] https://github.com/data61/MP-SPDZ
[3] https://github.com/relic-toolkit/relic

Table 9.1: Performance of the accumulator algorithms without access to the secret trapdoors. Time in milliseconds averaged over 100 executions.

Accu.	$\lvert\mathcal{X}\rvert$	Gen	Eval	WitCreate	Add	WitUpdate	Delete	WitUpdate
Scheme 1	2^{10}	649	1 117	1 116	1 116	0.6	1 120	0.7
	2^{14}	9 062	116 031	115 870	115 575	0.6	116 154	0.7
Merkle-Tree	2^{10}	–	1.12	0.05[a]	1.12	0.05[a]	1.12	0.05[a]
	2^{14}	–	15.53	0.83[a]	15.53	0.83[a]	15.53	0.83[a]

[a] Assuming that the full Merkle-tree is known as auxiliary data. If not, the tree has to be rebuilt, which adds the Eval-time.

Table 9.2: Number of Beaver triples, shared random values, and opening rounds required by MPC-q-SDH.

	Gen	Eval	WitCreate	Add	WitUpdate	Delete	WitUpdate
Beaver triples	0	$\lvert\mathcal{X}\rvert$	1	0	0	1	1
Random values	1	1	1	0	0	1	1
Opening rounds	1	$\lceil\log_2(\lvert\mathcal{X}\rvert+1)\rceil+1$	3	1	1	3	3

[a] Note, semi-honest Shamir secret sharing does not require Beaver triples.

connected via a 1 Gbit/s LAN network, and an average round-trip time of < 1 ms. For the WAN setting, a network with a round-trip time of 100 ms and a bandwidth of 100 Mbit/s was simulated. We provide benchmarks for both preprocessing and online phases of the MPC protocols, where the cost of the preprocessing phase is determined by the number of shared multiplications, whereas the performance of the online phase is proportional to the multiplicative depth of the circuit and the number of openings.

4.1 Evaluation of MPC-q-SDH

In the offline phase of the implemented MPC protocols, the required Beaver triples [Bea91] for shared multiplication and the pre-shared random values are generated. A shared inverse operation requires one multiplication and one shared random value. In Table 9.2, we list the number of triples required for each operation for the MPC-q-SDH accumulator. Except for Eval they require a constant number of multiplications and inverse operations and, therefore, a constant number of Beaver triples and shared random elements. In Eval, the number of required Beaver triples is determined by $\lvert\mathcal{X}\rvert$. Furthermore, Table 9.2 lists the number of opening rounds (including openings in multiplications, excluding MAC-checks) of the online phase of the MPC-q-SDH accumulator allowing one to calculate the number of communication rounds for different sharing schemes.

As discussed in Remark 14, Gen is not producing the public parameters h_i. If Eval without MPC is desired, the time and communication of Eval for the respective set sizes should be added to the time and communication of Gen to obtain an estimate of its performance.

Table 9.3: Offline phase performance of different steps of the MPC-q-SDH accumulator with access to the secret trapdoor based on MP-SPDZ. Time in milliseconds.

	Operation	$\lvert\mathcal{X}\rvert$	LAN setting $n=$ 2	3	4	5	WAN setting 2	3	4	5
	BaseOTs	2^{10}, 2^{14}	0.03	0.08	0.14	0.23	0.14	0.31	0.56	0.84
Semi-Honest	Inverse	2^{10}, 2^{14}	0.78	1.72	3.06	4.03	209.9	227.5	322.8	331.0
	Gen	2^{10}, 2^{14}	0.44	1.21	1.76	3.01	207.7	223.6	325.9	332.0
	Eval	2^{10}	189	397	706	959	4 695	8 215	13 680	25 725
		2^{14}	4000	8 308	14 380	17 928	55 542	109 720	214 585	356 330
Malicious	Inverse	2^{10}, 2^{14}	4.34	7.93	11.5	15.3	840.5	1 262	1 538	1 914
	Gen	2^{10}, 2^{14}	2.56	4.23	6.80	9.32	841.3	1 235	1 540	1 856
	Eval	2^{10}	1 601	2 849	4 345	6 227	25 737	45 254	87 328	141 181
		2^{14}	31 099	62 978	89 132	145 574	412 747	682 033	1 364 660	2 236 860

Dishonest-Majority based on SPDZ. Table 9.3 compares the offline performance of the MPC-q-SDH accumulator based on SPDZ in different settings. We give both timings for the accumulation of $|\mathcal{X}|$ elements in Eval and the necessary pre-computation for a single inversion, which is used in several other operations (e.g., WitCreate). Additionally we also give the time for pre-computing a single random element, which is required to generate the authenticated share of the secret-key in Eval. Further note that batching the generation of many triples together like for the Eval phase is more efficient in practice than producing a single triple and as these triples are not dependent on the input, all parties can continuously generate triples in the background for later use in the online phase.

In Table 9.4, we present the online performance of our MPC-q-SDH accumulator based on SPDZ for different set sizes, parties, security settings, and network settings. It can clearly be seen, that – except for the Eval operation – the runtime of each operation is independent of the set size. In other words, after an initial accumulation of a given set, every other operation has constant time. In comparison, the runtime of the non-MPC accumulators without access to the secret trapdoor, as depicted in Table 9.1, depends on the size of the accumulated set. Our MPC-accumulator outperforms the non-MPC q-SDH accumulators the larger the accumulated set gets. In the LAN setting MPC-q-SDH's Eval is faster than the non-MPC version for all benchmarked players, even in the WAN settings it outperforms the non-MPC version in the two player case. For 2^{14} elements, it is even faster for all benchmarked players in all settings, including the WAN setting. In any case, the witnesses have constant size contrary to the $\log_2(|\mathcal{X}|)$ sized witnesses of the Merkle-tree accumulator.

The numbers for the evaluation of the online phase in the WAN setting are also presented in Table 9.4. The overhead that can be observed compared to the LAN setting is influenced by the communication cost. Since our implementation implements all multiplications in Eval in a depth-optimized tree-like fashion, the overhead from switching to a WAN setting is not too severe.

Table 9.4: Online phase performance of the MPC-q-SDH accumulator with access to the secret trapdoor based on SPDZ implemented in MP-SPDZ, for both the LAN and WAN settings with n parties. Time in milliseconds averaged over 50 executions.

		Semi-Honest								Malicious							
		LAN setting				WAN setting				LAN setting				WAN setting			
Operation	$\lvert\mathfrak{X}\rvert$	$n=$ 2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5
Gen	2^{10}	4	4	7	19	53	110	170	219	11	13	25	37	169	278	395	505
	2^{14}	4	4	9	20	56	111	172	220	11	13	28	48	179	280	396	506
Eval	2^{10}	3	13	58	231	635	1 277	1 916	2 558	10	17	50	131	966	1 327	1 669	1 995
	2^{14}	26	47	117	315	949	1 948	3 166	4 571	89	94	174	225	1 297	1 979	2 830	3 872
WitCreate	2^{10}	2	2	32	39	168	320	482	645	5	10	35	75	372	606	823	1 050
	2^{14}	2	2	28	51	168	320	473	638	5	6	28	80	365	606	835	1 052
Add	2^{10}	2	2	8	17	47	107	166	213	5	5	17	31	170	273	388	499
	2^{14}	2	2	5	14	50	108	170	214	5	5	17	42	173	271	383	491
$\mathsf{WitUpdate}_{\mathrm{Add}}$	2^{10}	2	2	5	30	60	108	154	214	5	7	12	34	159	276	390	495
	2^{14}	2	2	3	20	60	107	152	217	5	6	10	54	154	275	390	500
Delete	2^{10}	2	2	21	58	156	319	488	639	5	10	47	78	379	598	818	1 034
	2^{14}	2	2	23	55	158	318	489	642	5	6	38	87	385	603	822	1 033
$\mathsf{WitUpdate}_{\mathrm{Delete}}$	2^{10}	2	2	52	47	165	320	475	643	5	10	26	100	374	604	828	1 044
	2^{14}	2	4	43	57	162	323	475	639	5	10	35	74	365	599	827	1 048

On the first look, one can observe an irregularity in our benchmarks. More specifically, notice that for four or more parties, the maliciously secure evaluation of the Eval online phase is consistently faster than the semi-honest evaluation of the same phase. However, this is a direct consequence of a difference in how MP-SPDZ handles the communication in those security models, where communication is handled in a non-synchronized send-to-all approach in the malicious setting and a synchronized broadcast approach in the semi-honest setting. The synchronization in the latter case scales worse for more parties and, therefore, introduces some additional delays.

Finally, Table 9.5 depicts the size of the communication between the parties for both offline and online phases. The communication of Eval has to account for a number of multiplications dependent on $\mathfrak{X}$ and therefore scales linearly with its size. As we already observed for the runtime of MPC-q-SDH, also the communication of WitCreate, Add, Delete and WitUpdate is independent of the size of the accumulated set, and additionally less than 200 kB for all algorithms. Combined with the analysis of the runtime, we conclude that the performance of the operations that might be performed multiple times per accumulator is very efficient in both runtime and communication. When compared to the performance of the non-MPC accumulators in Table 9.1, we see that the performance of operations that benefit from access to the secret trapdoor are multiple orders of magnitude faster in the MPC accumulators and, in the LAN setting, even come close to the performance of the standard Merkle-tree accumulator, for both the semi-honest and malicious variant.

Table 9.5: Communication cost (in kB per party) of the MPC-q-SDH accumulator with access to the secret trapdoor based on SPDZ implemented in MP-SPDZ.

Operations	$\lvert\mathcal{X}\rvert$	Semi-Honest Offline[a]	Semi-Honest Online	Malicious Offline[a]	Malicious Online
Gen	2^{10}, 2^{14}	20	0.10	86	0.24
Eval	2^{10}	12 571	66	79 549	66
	2^{14}	200 823	1 049	1 271 484	1 049
WitCreate, Delete, $\mathsf{WitUpdate}_{\mathrm{Delete}}$	2^{10}, 2^{14}	33	0.15	164	0.37
Add, $\mathsf{WitUpdate}_{\mathrm{Add}}$	2^{10}, 2^{14}	4	0.05	4	0.14

[a] Includes BaseOTs for a new connection

Table 9.6: Offline phase performance of different steps of the MPC-q-SDH accumulator with access to the secret trapdoor in the semi-honest (SH) and malicious threshold setting implemented in MP-SPDZ. Time in milliseconds.

	Operation	$\lvert\mathcal{X}\rvert$	LAN setting $n=3$	LAN $n=4$	LAN $n=5$	WAN setting 3	WAN 4	WAN 5
SH	Inverse	2^{10}, 2^{14}	6	9	14	473	585	998
Malicious	Inverse	2^{10}, 2^{14}	7	9	17	1 036	1 231	2 136
	Gen	2^{10}, 2^{14}	6	11	17	1 008	1 256	2 233
	Eval	2^{10}	20	29	48	1 232	2 089	2 629
		2^{14}	218	245	510	3 431	8 130	8 519

Honest-Majority Threshold Sharing based on Shamir Secret Sharing. In this section, we discuss the benchmarks of our implementation based on Shamir secret sharing. MP-SPDZ implements semi-honest Shamir secret sharing based on [CDM00] and a maliciously secure variant following [LN17][1]. In Table 9.6, we present the offline phase runtime, in Table 9.7 we show the runtime of the online phase, and in Table 9.8 we depict the size of the communication between the parties for the 3-party case.

The most expensive part of the SPDZ offline phase is creating the Beaver triples required for the Eval operation. As Table 9.6 shows, this step is several orders of magnitudes cheaper in the Shamir-based implementation. This is especially true in the semi-honest setting, in which no Beaver triples are required in the Shamir-based implementation. The offline runtime of the other operations is similar to the SPDZ-based implementations.

The Shamir-based implementation's online runtime is slightly cheaper than the runtime of the SPDZ-based implementation, except for the Eval operation.

[1] A newer version of MP-SPDZ now implements maliciously secure Shamir secret sharing following [CGH+18].

Table 9.7: Online phase performance of the MPC-q-SDH accumulator with access to the secret trapdoor in the threshold setting implemented in MP-SPDZ, for both the LAN and WAN settings with n parties. Time in milliseconds averaged over 50 executions.

		Semi-Honest						Malicious					
		LAN setting			WAN setting			LAN setting			WAN setting		
Operation	$\lvert\mathcal{X}\rvert$	$n=$ 3	4	5	3	4	5	3	4	5	3	4	5
Gen	2^{10}	5	5	7	109	111	118	7	7	14	112	119	228
	2^{14}	5	5	7	110	111	120	7	7	14	113	120	230
Eval	2^{10}	5	6	9	1 278	1 314	2 474	9	9	16	1 285	1 422	2 578
	2^{14}	33	40	77	1 788	2 776	3 831	80	84	161	2 018	3 938	4 636
WitCreate	2^{10}	2	2	3	317	319	440	3	3	5	324	336	648
	2^{14}	2	2	3	318	321	443	3	3	5	323	332	642
Add	2^{10}	2	2	3	109	108	114	3	3	5	109	111	215
	2^{14}	2	2	3	107	108	113	3	3	5	110	112	220
$\mathsf{WitUpdate}_{\mathrm{Add}}$	2^{10}	2	2	3	107	107	112	3	3	5	107	112	217
	2^{14}	2	2	3	107	108	114	3	3	5	108	114	220
Delete	2^{10}	2	2	3	320	321	438	3	3	5	320	332	642
	2^{14}	2	2	3	317	321	439	3	3	5	321	331	642
$\mathsf{WitUpdate}_{\mathrm{Delete}}$	2^{10}	2	2	3	320	321	441	3	3	5	321	333	647
	2^{14}	2	2	3	316	320	441	3	3	5	320	332	645

However, the difference in runtime of the Eval operation is also not significant, especially when considering the trade for the much cheaper offline phase.

Similar behavior can be seen for the communication cost, as depicted in Table 9.8. Offline communication is several orders of magnitude smaller in the Shamir-based implementation than in SPDZ, while online communication is similar to the SPDZ based version. Only the Eval operation requires about twice as much online communication in the Shamir-based implementation. To summarize, our honest-majority threshold implementation based on Shamir secret sharing provides much better offline phase performance, with similar online performance compared to our dishonest majority full-threshold implementation.

4.2 Further Improvement

The maliciously secure MPC protocols we use in this work delay the MAC check to the output phase after executing the Open subroutine. This means, it is possible for intermediate results to be wrong due to tampering of an attacker; however, since honest parties only reveal randomized values during the openings in a multiplication, no information about secret values can be gained by attackers.

Similar to threshold signature schemes [GG18; DKL+19; DOK+20], the protocols can be optimized by skipping the MAC checks at the end of WitCreate, Add, Delete, and WitUpdate and use the Verify step of the accumulator to check for correctness instead. The only feasible attack on this optimization is to pro-

Table 9.8: Communication cost (in kB per party) of the MPC-q-SDH accumulator in the 3-party threshold setting implemented in MP-SPDZ.

Operation	$\lvert\mathfrak{X}\rvert$	Semi-Honest Offline	Semi-Honest Online	Malicious Offline	Malicious Online
Gen	2^{10}, 2^{14}	0.26	0.20	0.65	0.20
Eval	2^{10}	0.26	66	459	131
	2^{14}	0.26	1 049	7 340	2 097
WitCreate, Delete, $\mathsf{WitUpdate}_{\mathrm{Delete}}$	2^{10}, 2^{14}	0.26	0.23	1.1	0.3
Add, $\mathsf{WitUpdate}_{\mathrm{Add}}$	2^{10}, 2^{14}	0	0.11	0	0.11

duce invalid accumulators/witnesses without leaking information on the secret trapdoor; however, false output values can be detected during verification. Therefore, we can execute the semi-honest online phase and call Verify at the end, while still protecting against malicious parties. This trades the extra round of communication in the MAC check for an evaluation of a bilinear pairing (≈ 10 ms on our benchmark platform) which results in a further speedup, especially in the WAN-setting.

5 Applications

5.1 Credential Revocation in Distributed Credential Systems

As first application of MPC-based accumulators, we focus on distributed credential systems [GGM14], and in particular, on the implementation in Sovrin [KL16]. In general, anonymous credentials provide a mechanism for making identity assertions while maintaining privacy, yet, in classical, non-distributed systems require a trusted credential issuer. This central issuer, however, is both a single point of failure and a target for compromise and can make it challenging to deploy such a system. In a distributed credential system, on the other hand, this trusted credential issuer is eliminated, e.g., by using distributed ledgers.

We shortly recall how Sovrin implements revocation. When issuing a credential, every user gets a unique revocation identifier i_R. All valid revocation IDs are accumulated using a q-SDH accumulator which is published. Additionally, the users obtains a witness certifying membership of its i_R in the accumulator. Whenever a user shows their credential, they have to prove that they know this witness for their i_R with respect to the published accumulator. When a new user joins, the accumulator has to be updated. Consequently, all the witnesses have to be updated as well, as otherwise they would no longer be able to provide a valid proof. Similar, in the case that a user is revoked and thus removed from the accumulator, all other users have to update their witnesses accordingly. Also, the verifiers always have to check for updated accumulators.

Now, recall that the q-SDH accumulator supports all required operations without needing access to the trapdoor. Hence, all operations can be performed and, especially, the users can update their witnesses on their own if the corresponding i_Rs are published on the ledger. While functionality-wise all operations are supported, performance-wise a large number of users becomes an issue. With potentially millions to billions of users, adding and deleting members from the accumulator becomes increasingly expensive (cf. Table 9.1). Hence, at a certain size, having access to the trapdoor would be beneficial. But, on the other side, generating membership witnesses for non-members would then become possible.

The latter is also an issue during the setup of the system. Trusting one third party to generate the public parameters of the accumulator might be undesired in a distributed system as in this case. The special structure of the Sovrin ecosystem with their semi-trusted foundation members, however, naturally fits to our multi-party accumulator. First, the foundation members can setup the public parameters in a distributed manner. Secondly, as all of them have shares of the trapdoor, they can also run the updates of the accumulator using the MPC-q-SDH-accumulator. Additionally, using a threshold secret sharing scheme can add robustness against foundation members failing to provide their shares for computations. The change to this accumulator is completely transparent to the clients and verifiers and no changes are required there. Furthermore, the Verify step of the MPC-q-SDH-accumulator is equal to the Verify operation of the non-MPC q-SDH-accumulator. Therefore, the same efficient zero-knowledge proofs [ACN13] can be used to prove knowledge of a witness without revealing it. These proofs are significantly more efficient then proving witnesses of a Merkle-Tree-accumulator, even when SNARK-friendly hash functions (e.g., Poseidon [GKK+19]) are used.

5.2 Privacy-Preserving Certificate-Transparency Logs

We finally look at the application of accumulators in the CT ecosystem. Certificate Authorities request the inclusion of certificates in the log whenever they sign a new certificate. Once the certificate was included in the log, auditors can check the consistency of this log. Additionally, TLS clients also verify whether all certificates that they obtain were actually logged, thereby ensuring that log servers do not hand out promises of certificate inclusion without following through. Technically, the CT log is realized as a Merkle-tree accumulator containing all certificates. As certificates need to be added continuously, it is made dynamic by simply recalculating the root hash and all the proofs. Functionality wise, dynamic accumulators would perfectly fit this use-case. However, their real-world performance without secret trapdoors is not good enough – recalculating hash trees is just more efficient. Knowledge of the secret trapdoors would however be catastrophic for this application, as the guarantees of the whole system break down: log servers could produce witnesses for any certificate they get queried on, even if it was never submitted to the log servers for inclusion.

In the CT ecosystem, the clients need to contact the log servers for the inclusion proof, and therefore verifying certificates has negative privacy implications, as this query reveals the browsing behavior of the client to the log server. Based on

previous work by Lueks and Goldberg [LG15], Kales et al. [KOR19] proposed to rethink retrieval of the inclusion proofs by employing multi-server private information retrieval (PIR) to query the proofs. To further improve performance, the accumulator is split into sub-accumulators based on, e.g., time periods. All sub-accumulators are then accumulated in a top-level accumulator. Consequently, the witnesses with respect to the sub-accumulator stay constant and can be embedded in the server's certificate and only the membership-proofs of the sub-accumulators need to be updated when new certificates are added to the log. Only these top-level proofs have to be queried using PIR, thus greatly improving the overall performance, as smaller databases are more efficient to query.

However, one drawback of this solution is the increase in certificate size if one were to include this static membership witness for the sub-accumulator in the certificate itself. Kales et al. [KOR19] propose to build sub-accumulators per hour, which would result in sub-accumulators that hold about 2^{16} certificates. A Merkle-tree membership proof for these sub-accumulators is 512 bytes in size when using SHA-256. In contrast, a membership proof for the q-SDH accumulator is only 48 bytes in size (with the curve used in our implementation). A typical DER-encoded X509 certificate using RSA-2048 as used in TLS is about 1-2 KB in size, meaning inclusion of the Merkle-tree sub-accumulator membership proof would increase the certificate size by $25 - 50\%$, whereas the q-SDH sub-accumulator membership proof only increases the size by $2.5 - 5\%$.

We can now leverage the fact that their solution already requires two non-colluding servers for the multi-server PIR. These servers hold copies of the Merkle-tree accumulator and answer private membership queries for the top-level accumulator. Switching the used accumulators to our MPC-q-SDH accumulator would give the benefit of small, constant size membership proofs, while still being performant enough to accumulate and produce witnesses for all elements of a sub-accumulator in one hour.

6 Conclusion

In this work, we introduced dynamic (threshold) secret-shared accumulators which remove the need of a trusted third party for public-key accumulators. By replacing the trusted party with a distributed setup algorithm, we achieved even more: since shares of the secret trapdoor are now available, the otherwise expensive algorithms can also be implemented as MPC protocol making use of the trapdoor. Thereby we obtained – especially in the bilinear groups setting – an efficient accumulator even for large sizes of accumulated sets.

Since our constructions are generic in a sense, improvements in the underlying MPC protocols and their implementations will directly translate to our accumulators.

Acknowledgments. This work was supported by EU's Horizon 2020 project under grant agreement n°825225 (Safe-DEED) and n°871473 (KRAKEN), and EU's Horizon 2020 ECSEL Joint Undertaking grant agreement n°783119 (SECREDAS),

and 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.

A Notation and Assumptions

Notation. Notation-wise, let $[n] := \{1, \ldots, n\}$ for $n \in \mathbb{N}$. For an algorithm $\mathcal{A}$, we write $\mathcal{A}(\cdots; r)$ to make the random coins explicit. We say that an algorithm is efficient if it runs in probabilistic polynomial time (PPT).

Assumptions. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of prime order p. A *pairing* $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a map that is *bilinear* (i.e., for all $g_1, g_1' \in \mathbb{G}_1$ and $g_2, g_2' \in \mathbb{G}_2$, we have $e(g_1 \cdot g_1', g_2) = e(g_1, g_2) \cdot e(g_1', g_2)$ and $e(g_1, g_2 \cdot g_2') = e(g_1, g_2) \cdot e(g_1, g_2')$, *non-degenerate* (i.e., for generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, we have that $e(g_1, g_2) \in G_T$ is a generator), and *efficiently computable*. Let BGen be a PPT algorithm that, on input of a security parameter n, outputs $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BGen}(n)$ for generators g_1 and g_2 of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $\Theta(n)$-bit prime p. If $\mathbb{G}_1 = \mathbb{G}_2$ the pairing is of Type-1 and if $\mathbb{G}_1 \neq \mathbb{G}_2$ and no non-trivial efficiently computable homomorphism $\mathbb{G}_2 \to \mathbb{G}_1$ exists, then it is of Type-3.

We recall the q-SDH assumption for Type-3 bilinear groups [BB08].

Definition 7 (q-SDH assumption)**.** *For $q > 0$, we define the advantage of an adversary $\mathcal{A}$ as*

$$\mathbf{Adv}^{\mathsf{q\text{-}SDH}}_{\mathsf{BGen},\mathcal{A}}(n) = \Pr\left[x \xleftarrow{R} \mathbb{Z}_p, (c, y) \leftarrow A\left(\mathsf{BG}, \left(g_1^{x^i}\right)_{i\in[q]}, g_2^x\right) : y = g_1^{(x+c)^{-1}}\right].$$

The q-SDH assumption holds if $\mathbf{Adv}^{\mathsf{q\text{-}SDH}}_{\mathsf{BGen},A}$ is a negligible function in the security parameter n for all PPT adversaries $\mathcal{A}$.

In the Type-1 setting, the assumption can be simplified to only providing the powers of g_1 to the adversary, since $g_1 = g_2$.

B Merkle-tree Accumulator

In Scheme 3, we cast the Merkle-tree accumulator in the framework of [DHS15] as done in [DRS18; KOR19]. In practical instantiations, the requirement that Eval only works on sets of a size that is a power of 2 can be dropped. It is always possible to repeat the last element until the tree is of the correct size. Correctness can easily be verified. We restate the well-known fact that this accumulator is collision-free.

Lemma 1. *If $\{H_k\}_{k\in\mathsf{K}^\kappa}$ is a family of collision-resistant hash functions, the static accumulator in Scheme 3 is collision-free.*

C Ideal Functionalities

In this section, we list the ideal functionalities used in the proofs in this paper. In Functionality 1 we present the ideal functionality $\mathcal{F}_{\text{Acc}}$, in Functionality 2 we present $\mathcal{F}_{\text{MPC-Acc}}$.

$\underline{\mathsf{Gen}(1^\kappa, t)}$: Fix a family of hash functions $\{H_k\}_{k\in\mathsf{K}^\kappa}$ with $H_k\colon \{0,1\}^* \to \{0,1\}^\kappa \ \forall\ k \in \mathsf{K}^\kappa$. Choose $k \xleftarrow{R} \mathsf{K}^\kappa$ and return $(\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda) \leftarrow (\emptyset, H_k)$.

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as H_k and $\mathcal{X}$ as $(x_0, \ldots, x_{n-1})$. If $\nexists\ k \in \mathbb{N}$ so that $n = 2^k$ return $\perp$. Otherwise, let $\ell_{u,v}$ refer to the u-th leaf (the leftmost leaf is indexed by 0) in the v-th layer (the root is indexed by 0) of a perfect binary tree. Return $\Lambda_\mathcal{X} \leftarrow \ell_{0,0}$ and $\mathsf{aux} \leftarrow ((\ell_{u,v})_{u\in[n/2^{k-v}]})_{v\in[k]}$, where

$$\ell_{u,v} \leftarrow \begin{cases} H_k(\ell_{2u,v+1} || \ell_{2u+1,v+1}) & \text{if } v < k, \text{ and} \\ H_k(x_i) & \text{if } v = k. \end{cases}$$

$\underline{\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x_i)}$: Parse aux as $((\ell_{u,v})_{u\in[n/2^{k-v}]})_{v\in[k]}$ and return wit_{x_i} where

$$\mathsf{wit}_{x_i} \leftarrow (\ell_{\lfloor i/2^v \rfloor + \eta, k-v})_{0\le v\le k}, \eta = \begin{cases} 1 & \text{if } \lfloor i/2^v \rfloor \pmod 2 = 0 \\ -1 & \text{otherwise.} \end{cases}$$

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_{x_i}, x_i)}$: Parse pk_Λ as H_k, $\Lambda_\mathcal{X}$ as $\ell_{0,0}$, set $\ell_{i,k} \leftarrow H_k(x_i)$. Recursively check for all $0 < v < k$ whether the following holds and return 1 if so. Otherwise return 0.

$$\ell_{\lfloor i/2^{v+1} \rfloor, k-(v+1)} = \begin{cases} H_k(\ell_{\lfloor i/2^v \rfloor, k-v} || \ell_{\lfloor i/2^v \rfloor + 1, k-v}) & \text{if } 2 | \lfloor i/2^v \rfloor \\ H_k(\ell_{\lfloor i/2^v \rfloor - 1, k-v} || \ell_{\lfloor i/2^v \rfloor, k-v}) & \text{otherwise.} \end{cases}$$

Scheme 3: Merkle-tree accumulator.

GEN: On input (gen, sid) from $\mathcal{AM}$ the functionality does the following:
1. If this is not the first gen command, or if sid does not encode the identity of $\mathcal{AM}$, ignore this command. Otherwise, continue.
2. $t \leftarrow 0$.
3. Initialize an empty list A (keeps track of all accumulator states).
4. Initialize map S, and set $S[0] \leftarrow \emptyset$ (maps operation counters to current accumulated sets).
5. Send (gen, sid) to $\mathcal{S}$.
6. Get $(algorithms, sid, (\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify}, \mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate}))$ from $\mathcal{S}$. Their expected input output behavior is described in Definition 3. All of them should be polynomial-time and Verify should be deterministic.
7. Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$.
8. Store sk, pk; add $\Lambda_\emptyset \leftarrow \emptyset$ to A.
9. Send $(algorithms, sid, (\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify}, \mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate}))$ to $\mathcal{AM}$.

EVAL: On input $(eval, sid, \mathcal{X})$ from $\mathcal{AM}$ the functionality does the following:
1. If sid does not encode the identity of $\mathcal{AM}$, or if $\mathcal{X} \not\subset \mathsf{D}_\Lambda$, ignore this command. Otherwise, continue.
2. $t \leftarrow t + 1$, and $S[t] \leftarrow \mathcal{X}$.
3. Run $(\Lambda_\mathcal{X}, \mathsf{aux}) \leftarrow \mathsf{Eval}((\mathsf{sk}, \mathsf{pk}), \mathcal{X})$.
4. Store aux; add $\Lambda_\mathcal{X}$ to A.
5. Send $(eval, sid, \Lambda_\mathcal{X}, \mathcal{X})$ to $\mathcal{AM}$.

WITCREATE: On input $(witcreate, sid, x)$ from $\mathcal{AM}$ the functionality does the following:
1. If sid does not encode the identity of $\mathcal{AM}$, or if $x \notin \mathsf{D}_\Lambda$, ignore this command. Otherwise, continue.
2. Run $w \leftarrow \mathsf{WitCreate}((\mathsf{sk}, \mathsf{pk}), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
3. If $x \notin S[t]$, send $\perp$ to $\mathcal{AM}$ and halt. Otherwise, continue.
4. If $\mathsf{Verify}(\mathsf{pk}, \Lambda_\mathcal{X}, w, x) = 1$ continue. Otherwise, send $\perp$ to $\mathcal{AM}$ and halt.
5. Send $(witness, sid, x, w)$ to $\mathcal{AM}$.

VERIFY: On input $(verify, sid, \Lambda, \mathsf{Verify}', x, w)$ from party $\mathcal{V}$ the functionality does the following:
1. If $\mathsf{Verify}' = \mathsf{Verify} \wedge \Lambda \in A$:
   (a) $t \leftarrow$ largest t such that $S[t]$ corresponds to Λ.
   (b) If $\mathcal{AM}$ not corrupted $\wedge\ x \notin S[t] \wedge \mathsf{Verify}(\mathsf{pk}, \Lambda, x, w) = 1$, send $\perp$ to $\mathcal{P}$. Otherwise, continue.
   (c) $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \Lambda, w, x)$

   Otherwise, set $b = \mathsf{Verify}'(\mathsf{pk}, \Lambda, w, x)$.
2. Send $(verified, sid, \Lambda, \mathsf{Verify}', x, w, b)$ to $\mathcal{V}$.

ADD: On input (add, sid, x) from $\mathcal{AM}$ the functionality does the following:
1. If sid does not encode the identity of $\mathcal{AM}$, or if $x \notin D_\Lambda$, ignore this command. Otherwise, continue.
2. If $x \in S[t]$ send $\perp$ to $\mathcal{AM}$ and halt. Otherwise, continue.
3. $t \leftarrow t + 1$, and $S[t] \leftarrow S[t-1]$.
4. Run $(\Lambda_{\mathcal{X}'}, \mathsf{aux}') \leftarrow \mathsf{Add}((\mathsf{sk}, \mathsf{pk}), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
5. Run $w \leftarrow \mathsf{WitCreate}((\mathsf{sk}, \mathsf{pk}), \Lambda_{\mathcal{X}'}, \mathsf{aux}', x)$.
6. If $\mathsf{Verify}(\mathsf{pk}, \Lambda_{\mathcal{X}'}, w, x) = 0$, send $\perp$ to $\mathcal{AM}$ and halt. Otherwise, continue.
7. Store aux'; add x to $S[t]$ and $\Lambda_{\mathcal{X}'}$ to A.
8. Send $(added, sid, \Lambda_{\mathcal{X}'}, x)$ to $\mathcal{AM}$.

DELETE: On input $(delete, sid, x)$ from $\mathcal{AM}$ the functionality does the following:
1. If sid does not encode the identity of $\mathcal{AM}$, or if $x \notin D_\Lambda$, ignore this command. Otherwise, continue.
2. If $x \notin S[t]$ send $\perp$ to $\mathcal{AM}$ and halt. Otherwise, continue.
3. $t \leftarrow t + 1$, and $S[t] \leftarrow S[t-1]$.
4. Run $(\Lambda_{\mathcal{X}'}, \mathsf{aux}') \leftarrow \mathsf{Delete}((\mathsf{sk}, \mathsf{pk}), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
5. Store aux'; remove x from $S[t]$ and add $\Lambda_{\mathcal{X}'}$ to A.
6. Send $(deleted, sid, \Lambda_{\mathcal{X}'}, x)$ to $\mathcal{AM}$.

WITUPDATE: On input $(witupdate, sid, \Lambda_{\mathcal{X}_{\text{old}}}, \Lambda_{\mathcal{X}_{\text{new}}}, x, w_{\text{old}})$ from a party $\mathcal{H}$, the functionality does the following:
1. If $\Lambda_{\mathcal{X}_{\text{old}}} \notin A \vee \Lambda_{\mathcal{X}_{\text{new}}} \notin A$, send $\perp$ to $\mathcal{H}$ and halt. Otherwise continue.
2. Run $w_{\text{new}} \leftarrow \mathsf{WitUpdate}((\mathsf{sk}, \mathsf{pk}), w_{\text{old}}, \mathsf{aux}, x)$.
3. If $\mathsf{Verify}(\mathsf{pk}, \Lambda_{\mathcal{X}_{\text{old}}}, w_{\text{old}}, x) = 1 \wedge x \in S[t] \wedge \mathsf{Verify}(\mathsf{pk}, \Lambda_{\mathcal{X}_{\text{new}}}, w_{\text{new}}, x) = 0$, send $\perp$ to $\mathcal{V}$ and halt. Otherwise, continue.
4. Send $(updatedwit, sid, \Lambda_{\mathcal{X}_{\text{old}}}, \Lambda_{\mathcal{X}_{\text{new}}}, x, w_{\text{old}}, w_{\text{new}})$ to $\mathcal{H}$.

Functionality 1: Ideal Functionality $\mathcal{F}_{\text{Acc}}$ for dynamic accumulators

GEN: On input (gen, sid_i) from all parties $\mathcal{AM}_i$, the functionality does the following:

1. If this is not the first gen command, or if for any $i \in [n]$, sid_i does not encode the identity of $\mathcal{AM}_i$, ignore this command. Otherwise, continue.
2. $t \leftarrow 0$.
3. Initialize an empty list A (keeps track of all accumulator states).
4. Initialize map S, and set $S[0] \leftarrow \emptyset$ (maps operation counters to current accumulated sets).
5. Send $(gen, sid_i)_{i\in[n]}$ to $\mathcal{S}$.
6. Get $(algorithms, (sid_1, \ldots, sid_n), (\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify}, \mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate}))$ from $\mathcal{S}$. Their expected input output behavior is described in Definition 6. All of them should be polynomial-time and Verify should be deterministic.
7. Run $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$. Store sk, pk; add $\Lambda_\emptyset \leftarrow \emptyset$ to A.
8. Send $(algorithms, sid_i, (\mathsf{Gen}, \mathsf{Eval}, \mathsf{WitCreate}, \mathsf{Verify}, \mathsf{Add}, \mathsf{Delete}, \mathsf{WitUpdate}))$ to $\mathcal{AM}_i$, for all $i = 1, \ldots, n$.

EVAL: On input $(eval, sid_k, \mathcal{X})$ from $\mathcal{AM}_k$ and $(eval, sid_j, ?)$ from all other parties $\mathcal{AM}_j$, for $j \neq k$, the functionality does the following:

1. If for any $i \in [n]$, sid_i does not encode the identity of $\mathcal{AM}_i$, or if $\mathcal{X} \not\subset \mathsf{D}_\Lambda$, ignore this command. Otherwise, continue.
2. $t \leftarrow t + 1$, and $S[t] \leftarrow \mathcal{X}$.
3. Run $(\Lambda_\mathcal{X}, \mathsf{aux}) \leftarrow \mathsf{Eval}((\mathsf{sk}, \mathsf{pk}), \mathcal{X})$. Store aux; add $\Lambda_\mathcal{X}$ to A.
4. Send $(eval, sid_i, \Lambda_\mathcal{X}, \mathcal{X})$ to $\mathcal{AM}_i$, for all $i = 1, \ldots, n$.

WITCREATE: On input $(witcreate, sid_k, x)$ from $\mathcal{AM}_k$ and $(witcreate, sid_j, ?)$ from all parties $\mathcal{AM}_j$, for $j \neq k$, the functionality does the following:

1. If for any $i \in [n]$, sid_i does not encode the identity of $\mathcal{AM}_i$, or if $x \notin \mathsf{D}_\Lambda$, ignore this command. Otherwise, continue.
2. Run $w \leftarrow \mathsf{WitCreate}((\mathsf{sk}, \mathsf{pk}), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
3. If $x \notin S[t]$, send $\perp$ to all $\mathcal{AM}_i$ and halt. Otherwise, continue.
4. If $\mathsf{Verify}(\mathsf{pk}, \Lambda_\mathcal{X}, w, x) = 1$ continue. Otherwise, send $\perp$ to all $\mathcal{AM}$ and halt.
5. Send $(witness, sid_i, x, w)$ to $\mathcal{AM}_i$, for all $i = 1, \ldots, n$.

VERIFY: On input $(verify, sid, \Lambda, \mathsf{Verify}', x, w)$ from party $\mathcal{V}$ the functionality does the following:

1. If $\mathsf{Verify}' = \mathsf{Verify} \wedge \Lambda \in A$:
   (a) $t \leftarrow$ largest t such that $S[t]$ corresponds to Λ.
   (b) If at least one $\mathcal{AM}_i$ is not corrupted $\wedge\ x \notin S[t] \wedge \mathsf{Verify}(\mathsf{pk}, \Lambda, x, w) = 1$, send $\perp$ to $\mathcal{V}$. Otherwise, continue.
   (c) $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \Lambda, w, x)$
   Otherwise, set $b = \mathsf{Verify}'(\mathsf{pk}, \Lambda, w, x)$.
2. Send $(verified, sid, \Lambda, \mathsf{Verify}', x, w, b)$ to $\mathcal{V}$.

ADD: On input (add, sid_k, x) from $\mathcal{AM}_k$ and $(add, sid_j, ?)$ from all parties $\mathcal{AM}_j$, for $j \neq k$, the functionality does the following:

1. If for any $i \in [n]$, sid_i does not encode the identity of $\mathcal{AM}_i$, or if $x \notin D_\Lambda$, ignore this command. Otherwise, continue.
2. If $x \in S[t]$ send $\perp$ to all $\mathcal{AM}_i$ and halt. Otherwise, continue.
3. $t \leftarrow t + 1$, and $S[t] \leftarrow S[t-1]$.
4. Run $(\Lambda_{\mathcal{X}'}, \mathsf{aux}') \leftarrow \mathsf{Add}((\mathsf{sk}, \mathsf{pk}), \Lambda_\mathcal{X}, \mathsf{aux}, x)$. Run $w \leftarrow \mathsf{WitCreate}((\mathsf{sk}, \mathsf{pk}), \Lambda_{\mathcal{X}'}, \mathsf{aux}', x)$.
5. If $\mathsf{Verify}(\mathsf{pk}, \Lambda_{\mathcal{X}'}, w, x) = 0$, send $\perp$ to all $\mathcal{AM}_i$ and halt. Otherwise, continue.
6. Store aux'; add x to $S[t]$ and $\Lambda_{\mathcal{X}'}$ to A.
7. Send $(added, sid_i, \Lambda_{\mathcal{X}'}, x)$ to $\mathcal{AM}_i$, for all $i = 1, \ldots, n$.

DELETE: On input $(delete, sid_k, x)$ from $\mathcal{AM}_k$ and $(delete, sid_j, ?)$ from all parties $\mathcal{AM}_j$, for $j \neq k$, the functionality does the following:

1. If for any $i \in [n]$, sid_i does not encode the identity of $\mathcal{AM}_i$, or if $x \notin D_\Lambda$, ignore this command. Otherwise, continue.
2. If $x \notin S[t]$ send $\perp$ to all $\mathcal{AM}_i$ and halt. Otherwise, continue.
3. $t \leftarrow t + 1$, and $S[t] \leftarrow S[t-1]$.
4. Run $(\Lambda_{\mathcal{X}'}, \mathsf{aux}') \leftarrow \mathsf{Delete}((\mathsf{sk}, \mathsf{pk}), \Lambda_\mathcal{X}, \mathsf{aux}, x)$.
5. Store aux'; remove x from $S[t]$ and add $\Lambda_{\mathcal{X}'}$ to A.
6. Send $(deleted, sid_i, \Lambda_{\mathcal{X}'}, x)$ to $\mathcal{AM}_i$, for all $i = 1, \ldots, n$.

WITUPDATE: On input $(witupdate, sid_k, \Lambda_{\mathcal{X}_{\text{old}}}, \Lambda_{\mathcal{X}_{\text{new}}}, x, w_{\text{old}})$ from $\mathcal{AM}_k$ and $(witupdate, sid_j, ?, ?, ?, ?)$ from all parties $\mathcal{AM}_j$, for $j \neq k$, the functionality does the following:

1. If for any $i \in [n]$, sid_i does not encode the identity of $\mathcal{AM}_i$, or if $x \notin D_\Lambda$, ignore this command. Otherwise, continue.
2. If $\Lambda_{\mathcal{X}_{\text{old}}} \notin A \vee \Lambda_{\mathcal{X}_{\text{new}}} \notin A$, send $\perp$ to all $\mathcal{AM}_i$ and halt. Otherwise continue.
3. Run $w_{\text{new}} \leftarrow \mathsf{WitUpdate}((\mathsf{sk}, \mathsf{pk}), w_{\text{old}}, \mathsf{aux}, x)$.
4. If $\mathsf{Verify}(\mathsf{pk}, \Lambda_{\mathcal{X}_{\text{old}}}, w_{\text{old}}, x) = 1 \wedge x \in S[t] \wedge \mathsf{Verify}(\mathsf{pk}, \Lambda_{\mathcal{X}_{\text{new}}}, w_{\text{new}}, x) = 0$, send $\perp$ to $\mathcal{V}$ and halt. Otherwise, continue.
5. Send $(updatedwit, sid_i, \Lambda_{\mathcal{X}_{\text{old}}}, \Lambda_{\mathcal{X}_{\text{new}}}, x, w_{\text{old}}, w_{\text{new}})$ to $\mathcal{AM}_i$, for all $i = 1, \ldots, n$.

Functionality 2: Ideal Functionality $\mathcal{F}_{\text{Acc-MPC}}$

D Strong-RSA Accumulator

We recall the strong RSA assumption [BP97].

Definition 8 (Strong RSA assumption)**.** *Given two appropriately chosen primes p and q such that $N = pq$ has bit-length n, then it holds for all PPT adversaries $\mathcal{A}$ that*

$$\Pr\left[u \xleftarrow{R} \mathbb{Z}_N^*, (v, w) \leftarrow \mathcal{A}(u, N) \colon v^w \equiv u \pmod{N}\right]$$

is negligible in the security parameter n.

Major lines of work investigated accumulators in the hidden order groups, i.e., RSA-based, and the known order groups, i.e., discrete logarithm-based, setting. The first collision-free RSA-based accumulator is due to Baric and Pfitzmann [BP97]. The accumulator in this construction consists of a generator raised to the product of all elements of the set. Then witnesses essentially consist of the same value skipping the respective elements in the product. Thereby, the witness can easily be verified by raising the power of the witness to the element and checking if the result matches the accumulator. We recall the RSA-based accumulator in Scheme 4.

Note that, whenever the factorization of N is available, the Chinese remainder theorem can be used to speed up the computations. For WitCreate and WitUpdate we can use the factorization to compute inverses $\mod (p-1)\cdot(q-1)$. Deletion of values from the accumulator is not possible if the factorization is unknown.

Correctness can easily be verified and collision freeness follows from the strong RSA assumption:

Theorem 4 ([BP97])**.** *If the strong RSA assumption holds, Scheme 4 is collision-free.*

Again, from Theorems 2 and 4, it follows that Scheme 4 is also secure in the UC model:

Corollary 4. *Scheme 4 emulates $\mathcal{F}_{Acc}$ in the UC model.*

D.1 Dynamic (Threshold) Secret-Shared Accumulator From the Strong RSA Assumption

For our dynamic (threshold) secret-shared accumulator from the strong RSA assumption, observe that the two main operations that have to be performed in the context of an MPC protocol are the sampling of the secret prime factors as well as the computation of the inverses of the public elements in the exponent. Both operations are also performed during RSA key generation with the sole difference that in this case, the public exponent is inverted. Therefore, we can make use of any protocol for distributed RSA key generation. In particular, our construction makes use of the state-of-the-art protocol by Frederiksen et al. [FLO+18] for the two-party case.

$\underline{\mathsf{Gen}(1^\kappa, t)}$: Choose an RSA modulus $N = p \cdot q$ with two large safe primes p, q, and let g be a random quadratic residue $\mod N$. Set $\mathsf{sk}_\Lambda \leftarrow \emptyset$ and $\mathsf{pk}_\Lambda \leftarrow (N, g)$.

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as (N, g) and let $\mathcal{X} \subset \mathbb{P}$. Return $\Lambda_\mathcal{X} \leftarrow g^{\prod_{x \in \mathcal{X}} x} \mod N$ and $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0, \mathcal{X})$.

$\underline{\mathsf{WitCreate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: If $x \notin \mathcal{X}$, return $\perp$. If $\mathsf{sk}_\Lambda \neq \emptyset$, return $\mathsf{wit}_x \leftarrow \Lambda_\mathcal{X}^{x^{-1}} \mod N$, otherwise return $\mathsf{wit}_x \leftarrow g^{\prod_{x' \in \mathcal{X} \setminus \{x\}} x'} \mod N$.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Parse pk_Λ as (N, g). If $\mathsf{wit}_x^x = \Lambda_\mathcal{X} \mod N$ holds, return 1, otherwise return 0.

$\underline{\mathsf{Add}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g) and aux as $\mathcal{X}$. If $x \in \mathcal{X}$, return $\perp$. Set $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$, $\mathsf{aux}' \leftarrow (\mathcal{X}', \mathsf{add} \leftarrow 1)$, and $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^x \mod N$. Return $\Lambda_{\mathcal{X}'}$ and aux'.

$\underline{\mathsf{Delete}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g) and aux as $\mathcal{X}$. If $x \notin \mathcal{X}$, return $\perp$. If $\mathsf{sk}_\Lambda \neq \emptyset$, set $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$, $\mathsf{aux}' \leftarrow (\mathcal{X}', \mathsf{add} \leftarrow -1)$, and $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^{x^{-1}} \mod N$. Otherwise, compute $(\Lambda_{\mathcal{X}'}, \mathsf{aux}') \leftarrow \mathsf{Eval}((\emptyset, \mathsf{pk}_\Lambda), \mathcal{X} \setminus \{x\})$ with $\mathsf{add} \leftarrow -1$ in aux'. Return $\Lambda_{\mathcal{X}'}$ and aux'.

$\underline{\mathsf{WitUpdate}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g) and aux as $(\mathcal{X}, \mathsf{add})$. If $\mathsf{add} = 0$, return $\perp$. Return $\mathsf{wit}_{x_i}^x \mod N$ if $\mathsf{add} = 1$. If instead $\mathsf{add} = -1$ and $\mathsf{sk}_\Lambda \neq \emptyset$, then return $\mathsf{wit}_{x_i}^{x^{-1}} \mod N$. Otherwise, compute $a, b \in \mathbb{Z}$ such that $ax_i + bx = 1$ and return $\mathsf{wit}_{x_i}^b \cdot \Lambda_\mathcal{X}^a \mod N$. In the last two cases in addition return $\mathsf{aux} \leftarrow (\mathsf{add} \leftarrow 0)$.

Scheme 4: Strong RSA-based accumulator.

In the following, we will recap the structure of this UC secure two-party protocol and highlight the most essential parts of the protocol. The key generation in the malicious setting consists of the following four phases:

Candidate Generation: Both parties P_1 and P_2 choose random shares $p_1 \in \mathbb{N}$ respectively $p_2 \in \mathbb{N}$ and commit to it. Based on maliciously secure OT, they do a secure trial division of $p = p_1 + p_2$ with public threshold $B_1 \in \mathbb{N}$.

Construct Modulus: Given shared candidate primes $p = p_1 + p_2, q = q_1 + q_2$ the parties compute $N = pq$ by a custom version of the Gilboa protocol [Gil99]. The candidate modulus N is send to both parties.

Verify Modulus: This phase consists of three steps:

1. Second trial division with threshold $B_2 > B_1$.
2. Secure biprimality test.

3. Proof of honesty that checks the commitments and whether $\gcd(e, \phi(N)) = 1$, where e is the public exponent.

Construct Keys: Computes the shares of $d = d_1 + d_2$ such that $e(d_1 + d_2) \equiv 1 \mod \phi(N)$ (uses additional output from the proof of honesty).

We will denote this protocol by Π_{RSA} and by $\mathcal{F}_{\text{RSA}}$ its ideal functionality. At a later point in the RSA-based accumulator, we need to invert an element $x \in \mathbb{P}$ in $\mathbb{Z}_N^*$. Since neither P_1 nor P_2 knows the order $\phi(N)$ of this ring, we employ parts of the MPC protocol. For this task, we will perform the 3rd step of **Verify Modulus** with $e = x$, and then immediately run **Construct Keys**. The output of this sub-protocol, which we will denote by $\mathsf{Invert}_{\phi(N)}(x)$, is a secret-shared value $\langle y \rangle = y_1 + y_2$, where y_i is the share of party i, s.t. $x(y_1 + y_2) \equiv 1 \mod \phi(N)$.

$\underline{\mathsf{Gen}(1^\kappa, t)}$: Generate an RSA modulus $N = (p_1 + p_2) \cdot (q_1 + q_2)$ via the protocol Π_{RSA}, where the party P_i receives (p_i, q_i) for $i = 1, 2$. Further, let g be a random quadratic residue $\mod N$. Set $\mathsf{pk}_\Lambda \leftarrow (N, g)$.

$\underline{\mathsf{Eval}((\mathsf{sk}_\Lambda, \mathsf{pk}_\Lambda), \mathcal{X})}$: Parse pk_Λ as (N, g) and $\mathcal{X} \subset \mathbb{P}$. Return $\Lambda_\mathcal{X} \leftarrow g^{\prod_{x \in \mathcal{X}} x} \mod N$ and $\mathsf{aux} \leftarrow \mathcal{X}$.

$\underline{\mathsf{WitCreate}(((p_1, q_1), (p_2, q_2), \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: If $x \notin \mathcal{X}$, return $\perp$. Compute $\langle y \rangle \leftarrow \mathsf{Invert}_{\phi(N)}(x)$ and return $\mathsf{wit}_x \leftarrow \mathsf{Open}(\Lambda_\mathcal{X}^{\langle y \rangle} \mod N)$.

$\underline{\mathsf{Verify}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{wit}_x, x)}$: Parse pk_Λ as (N, g). If $\mathsf{wit}_x^x = \Lambda_\mathcal{X} \mod N$ holds, return 1, otherwise return 0.

$\underline{\mathsf{Add}(\mathsf{pk}_\Lambda, \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g) and aux as $\mathcal{X}$. If $x \in \mathcal{X}$, return $\perp$. Set $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$, $\mathsf{aux}' \leftarrow (\mathcal{X}', \text{add} \leftarrow 1)$, and $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^x \mod N$. Return $\Lambda_{\mathcal{X}'}$ and aux'.

$\underline{\mathsf{Delete}(((p_1, q_1), (p_2, q_2), \mathsf{pk}_\Lambda), \Lambda_\mathcal{X}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g) and aux as $\mathcal{X}$. If $x \notin \mathcal{X}$, return $\perp$. Set $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$, $\mathsf{aux}' \leftarrow (\mathcal{X}', \text{add} \leftarrow -1)$, and $\langle y \rangle \leftarrow \mathsf{Invert}_{\phi(N)}(x)$. Return $\Lambda_{\mathcal{X}'} \leftarrow \mathsf{Open}(\Lambda_\mathcal{X}^{\langle y \rangle} \mod N)$ and aux'.

$\underline{\mathsf{WitUpdate}(((p_1, q_1), (p_2, q_2), \mathsf{pk}_\Lambda), \mathsf{wit}_{x_i}, \mathsf{aux}, x)}$: Parse pk_Λ as (N, g) and aux as $(\mathcal{X}, \text{add})$. Return $\perp$ if $\text{add} = 0 \vee x_i \notin \mathcal{X}$. Return $\mathsf{wit}_{x_i}^x \mod N$ if $\text{add} = 1$. If instead $\text{add} = -1$ and compute $\langle y \rangle \leftarrow \mathsf{Invert}_{\phi(N)}(x)$, then return $\mathsf{Open}(\mathsf{wit}_{x_i}^{\langle y \rangle} \mod N)$. In the last two cases in addition return $\mathsf{aux} \leftarrow (\text{add} \leftarrow 0)$.

Scheme 5: MPC-RSA: Dynamic (threshold) secret-shared accumulator based on RSA for two parties.

However, we have to note that the used distributed RSA key generation protocols may leak a small amount of secret information. We refer to the full version of [FLO+18] for a detailed discussion of this leakage. Since our protocol

may invert multiple elements, the parties might need to add bounds on the maximum number of operations to prevent leakage of the secret key.

The last open point during Gen is the sampling of the quadratic non-residue mod N. An option here is to simply sample g at random from $\mathbb{Z}_N^*$ and checking whether the Jacobi symbol satisfies $(\frac{g}{N}) = -1$. Despite its definition as the product of the Legendre symbols of the prime factors of N, the Jacobi symbol can be computed efficiently without knowledge of the prime factors using an algorithm analogous to the Euclidean algorithm. Hence, we perform this step outside of the MPC protocol once N was generated.

In the security analysis of Scheme 5, we can reuse the arguments of Theorem 3. Therefore, we will omit the proof of the following theorem. We, however, note, that the theorem only holds provided that the parties keep track of the number of potentially leaked bits and abort if this number gets too large.

Theorem 5. *Scheme 5 UC emulates $\mathcal{F}_{Acc\text{-}MPC}$ in the $\mathcal{F}_{ABB+}$,$\mathcal{F}_{RSA}$-hybrid model.*

E FRESCO Benchmarks

In this section, we discuss the benchmarks of our q-SDH implementation in the maliciously secure dishonest-majority setting in the FRESCO framework. FRESCO is a Java framework facilitating fast prototyping of MPC-based applications and protocols. FRESCO implements the SPDZ protocol and various extensions [DPS+12; KOS15; KOS16; CDE+18]. For pairing and elliptic curve group operations, we rely on the ECCelerate library[1] and integrate $\mathsf{Exp}_{\mathbb{G}}$, Output-$\mathbb{G}$, and the corresponding $\text{MACCheck}_{\text{ECC}}$ algorithms of [SA19] into FRESCO. As a pairing-friendly curve, a 400-bit Barreto-Naehrig curve [BN05] is used, which provides around 100 bit of security following recent estimates [BD19].

The offline phase performance of our FRESCO implementation can be seen in Table 9.9. A comparison to our MP-SPDZ implementation (Table 9.3) shows that the offline phase in FRESCO is not yet as optimized as the one of MP-SPDZ (as an example, the BaseOTs used in FRESCO are not using elliptic curve arithmetic) and that the performance of the latter is more indicative of an optimized implementation. Further note that batching the generation of many triples together like for the Eval phase is more efficient in practice than producing a single triple and as these triples are not dependent on the input, all parties can continuously generate triples in the background to fill a triple-buffer for use in the online phase.

In Table 9.10, we present the online phase performance of our q-SDH implementation in the FRESCO framework. Since the FRESCO implementation does not support a depth-optimized tree-like multiplication, the Eval operation scales worse with the number of parties. Compared to our MP-SPDZ implementation (Table 9.4) it is, therefore, much slower, especially in the WAN setting.

[1] https://jce.iaik.tugraz.at/sic/Products/Core_Crypto_Toolkits/ECCelerate

Table 9.9: Offline phase performance of different steps of the MPC-q-SDH accumulator with access to the secret trapdoor implemented in FRESCO. Time in seconds.

Operation	$\lvert\mathcal{X}\rvert$	LAN setting $n =$ 2	3	4	5	WAN setting 2	3	4	5
BaseOTs	2^{10}	21.5	60.3	104.6	148.9	76.5	215.0	364.2	504.3
	2^{14}	21.5	60.3	104.6	148.9	76.5	215.0	364.2	504.3
Eval	2^{10}	54.4	154.0	265.0	409.1	95.7	283.1	465.2	683.3
	2^{14}	825.8	2 259.5	4 048.8	6 150.9	1 449.8	3 587.1	6 578.4	10 056.3
Inverse	2^{10}	1.3	15.3	16.8	19.7	3.2	68.2	70.4	72.8
	2^{10}	1.3	15.3	16.8	19.7	3.2	68.2	70.4	72.8

Finally, we present the communication cost of our q-SDH implementation in the FRESCO framework in Table 9.11. Again, a comparison to our MP-SPDZ implementation (Table 9.5) shows, that the FRESCO framework requires more communication to achieve the same result. While some of this overhead can be attributed to the different choice of elliptic curve, the rest is inherent to the implementation of the framework.

Table 9.10: Online phase performance of the MPC-q-SDH accumulator with access to the secret trapdoor implemented in FRESCO, for both the LAN and WAN settings with n parties. Time in milliseconds averaged over 50 executions.

Operation	$\lvert\mathcal{X}\rvert$	LAN setting $n=$ 2	3	4	5	WAN setting 2	3	4	5
Gen	2^{10}	78	106	301	772	604	1 124	1 399	1 684
	2^{14}	75	110	246	766	608	1 118	1 401	1 673
Eval	2^{10}	133	354	4 062	14 193	52 316	56 802	58 728	60 043
	2^{14}	1 389	4 362	61 222	196 563	828 522	892 293	918 445	935 274
WitCreate	2^{10}	40	84	279	906	1 109	1 858	2 156	2 444
	2^{14}	41	98	267	992	1 120	1 853	2 152	2 442
Add	2^{10}	43	78	231	646	574	1 088	1 373	1 650
	2^{14}	41	74	225	706	571	1 087	1 363	1 642
$\mathsf{WitUpdate}_{\mathrm{Add}}$	2^{10}	40	72	259	745	569	1 094	1 372	1 649
	2^{14}	40	85	213	732	564	1 088	1 369	1 644
Delete	2^{10}	47	80	299	958	1 075	1 849	2 150	2 454
	2^{14}	42	92	297	857	1 071	1 847	2 152	2 446
$\mathsf{WitUpdate}_{\mathrm{Delete}}$	2^{10}	40	82	258	977	1 071	1 852	2 151	2 447
	2^{14}	38	38	294	880	1 077	1 852	2 149	2 441

Table 9.11: Communication cost (in MiB per party) of the MPC-q-SDH accumulator with access to the secret trapdoor implemented in FRESCO.

Operations	$\lvert\mathcal{X}\rvert$	Offline phase[a]	Online Phase[b]
Gen	2^{10}	0.297	0.103
	2^{14}	0.297	0.103
Eval	2^{10}	220.971	0.176
	2^{14}	3 530.148	3.164
WitCreate, Delete, $\mathsf{WitUpdate}_{\mathrm{Delete}}$	2^{10}	0.634	0.041
	2^{14}	0.634	0.041
Add, $\mathsf{WitUpdate}_{\mathrm{Add}}$	2^{10}	0.297	0.039
	2^{14}	0.297	0.039

[a] Includes BaseOTs for a new connection
[b] Includes the setup of a fresh MAC for each share of the secret trapdoor

References

[ABL+17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. "A Subversion-Resistant SNARK." In: *ASIACRYPT (3)*. Vol. 10626. LNCS. Springer, 2017, pp. 3–33.

[ACN13] Tolga Acar, Sherman S. M. Chow, and Lan Nguyen. "Accumulators and U-Prove Revocation." In: *Financial Cryptography*. Vol. 7859. LNCS. Springer, 2013, pp. 189–196.

[ARS20] Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. "Lift-and-Shift: Obtaining Simulation Extractable Subversion and Updatable SNARKs Generically." In: *CCS*. ACM, 2020, pp. 1987–2005.

[ATS+09] Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. "Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems." In: *CT-RSA*. Vol. 5473. LNCS. Springer, 2009, pp. 295–308.

[BB08] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups." In: *J. Cryptology* 21.2 (2008), pp. 149–177.

[BB89] Judit Bar-Ilan and Donald Beaver. "Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction." In: *PODC*. ACM, 1989, pp. 201–209.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains." In: *CRYPTO (1)*. Vol. 11692. LNCS. Springer, 2019, pp. 561–586.

[BBH+19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup." In: *CRYPTO (3)*. Vol. 11694. LNCS. Springer, 2019, pp. 701–732.

[BCY20] Foteini Baldimtsi, Ran Canetti, and Sophia Yakoubov. "Universally Composable Accumulators." In: *CT-RSA*. Vol. 12006. LNCS. Springer, 2020, pp. 638–666.

[BD19] Razvan Barbulescu and Sylvain Duquesne. "Updating Key Size Estimations for Pairings." In: *J. Cryptology* 32.4 (2019), pp. 1298–1336.

[Bea91] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization." In: *CRYPTO*. Vol. 576. LNCS. Springer, 1991, pp. 420–432.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. "NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion." In: *ASIACRYPT (2)*. Vol. 10032. LNCS. 2016, pp. 777–804.

[BL11] Ernie Brickell and Jiangtao Li. “Enhanced privacy ID from bilinear pairing for hardware authentication and attestation.” In: *IJIPSI* 1.1 (2011), pp. 3–33.

[BL12] Ernie Brickell and Jiangtao Li. “Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities.” In: *IEEE Trans. Dependable Sec. Comput.* 9.3 (2012), pp. 345–360.

[BLS02] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. “Constructing Elliptic Curves with Prescribed Embedding Degrees.” In: *SCN*. Vol. 2576. LNCS. Springer, 2002, pp. 257–267.

[BM93] Josh Cohen Benaloh and Michael de Mare. “One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract).” In: *EUROCRYPT*. Vol. 765. LNCS. Springer, 1993, pp. 274–285.

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. “Pairing-Friendly Elliptic Curves of Prime Order.” In: *SAC*. Vol. 3897. LNCS. Springer, 2005, pp. 319–331.

[BP97] Niko Baric and Birgit Pfitzmann. “Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees.” In: *EUROCRYPT*. Vol. 1233. LNCS. Springer, 1997, pp. 480–494.

[BS01] Emmanuel Bresson and Jacques Stern. “Efficient Revocation in Group Signatures.” In: *PKC*. Vol. 1992. LNCS. Springer, 2001, pp. 190–206.

[Can01] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols.” In: *FOCS*. IEEE, 2001, pp. 136–145.

[CCD+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and Abhi Shelat. “Multiparty Generation of an RSA Modulus.” In: *CRYPTO (3)*. Vol. 12172. LNCS. Springer, 2020, pp. 64–93.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. “SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^{k} for Dishonest Majority.” In: *CRYPTO (2)*. Vol. 10992. LNCS. Springer, 2018, pp. 769–798.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. “General Secure Multi-party Computation from any Linear Secret-Sharing Scheme.” In: *EUROCRYPT*. Vol. 1807. LNCS. Springer, 2000, pp. 316–334.

[CGG+17] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. “Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services.” In: *ACM CCS*. ACM, 2017, pp. 229–243.

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. "Fast Large-Scale Honest-Majority MPC for Malicious Adversaries." In: *CRYPTO (3)*. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 34–64.

[CHI+20] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthuramakrishnan Venkitasubramaniam, and Ruihan Wang. "Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority." In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 374.

[CL02] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials." In: *CRYPTO*. Vol. 2442. LNCS. Springer, 2002, pp. 61–76.

[DG20] Samuel Dobson and Steven D. Galbraith. "Trustless Groups of Unknown Order with Hyperelliptic Curves." In: *IACR ePrint* 2020 (2020), p. 196.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. "Revisiting Cryptographic Accumulators, Additional Properties and Relations to Other Primitives." In: *CT-RSA*. Vol. 9048. LNCS. Springer, 2015, pp. 127–144.

[DK01] Ivan Damgård and Maciej Koprowski. "Practical Threshold RSA Signatures without a Trusted Dealer." In: *EUROCRYPT*. Vol. 2045. LNCS. Springer, 2001, pp. 152–165.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits." In: *ESORICS*. Vol. 8134. LNCS. Springer, 2013, pp. 1–18.

[DKL+19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. "Threshold ECDSA from ECDSA Assumptions: The Multiparty Case." In: *IEEE S&P*. IEEE, 2019, pp. 1051–1066.

[DN03] Ivan Damgård and Jesper Buus Nielsen. "Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption." In: *CRYPTO*. Vol. 2729. LNCS. Springer, 2003, pp. 247–264.

[DOK+20] Anders P. K. Dalskov, Claudio Orlandi, Marcel Keller, Kris Shrishak, and Haya Shulman. "Securing DNSSEC Keys via Threshold ECDSA from Generic MPC." In: *ESORICS (2)*. Vol. 12309. LNCS. Springer, 2020, pp. 654–673.

[DPS+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption." In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 643–662.

[DRS18] David Derler, Sebastian Ramacher, and Daniel Slamanig. "Post-Quantum Zero-Knowledge Proofs for Accumulators with Applications to Ring Signatures from Symmetric-Key Primitives." In: *PQCrypto*. Vol. 10786. LNCS. Springer, 2018, pp. 419–440.

[FHM11] Chun-I Fan, Ruei-Hau Hsu, and Mark Manulis. "Group Signature with Constant Revocation Costs for Signers and Verifiers." In: *CANS*. Vol. 7092. LNCS. Springer, 2011, pp. 214–233.

[FLO+18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. "Fast Distributed RSA Key Generation for Semi-honest and Malicious Adversaries." In: *CRYPTO (2)*. Vol. 10992. LNCS. Springer, 2018, pp. 331–361.

[Fuc18] Georg Fuchsbauer. "Subversion-Zero-Knowledge SNARKs." In: *PKC (1)*. Vol. 10769. LNCS. Springer, 2018, pp. 315–347.

[GG18] Rosario Gennaro and Steven Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup." In: *ACM CCS*. ACM, 2018, pp. 1179–1194.

[GGM14] Christina Garman, Matthew Green, and Ian Miers. "Decentralized Anonymous Credentials." In: *NDSS*. The Internet Society, 2014.

[Gil99] Niv Gilboa. "Two Party RSA Key Generation." In: *CRYPTO*. Vol. 1666. LNCS. Springer, 1999, pp. 116–129.

[GKK+19] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. "Starkad and Poseidon: New Hash Functions for Zero Knowledge Proof Systems." In: *IACR ePrint* 2019 (2019), p. 458.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. "Updatable and Universal Common Reference Strings with Applications to zk-SNARKs." In: *CRYPTO (3)*. Vol. 10993. LNCS. Springer, 2018, pp. 698–728.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. "Pairings for cryptographers." In: *Discret. Appl. Math.* 156.16 (2008), pp. 3113–3121.

[HM00] Safuat Hamdy and Bodo Möller. "Security of Cryptosystems Based on Class Groups of Imaginary Quadratic Orders." In: *ASIACRYPT*. Vol. 1976. LNCS. Springer, 2000, pp. 234–247.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation." In: *CCS*. ACM, 2020, pp. 1575–1590.

[KL16] Dmitry Khovratovich and Jason Law. *Sovrin: digitial signatures in the blockchain area*. 2016. URL: https://sovrin.org/wp-content/uploads/AnonCred-RWC.pdf.

[KOR19] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. "Revisiting User Privacy for Certificate Transparency." In: *EuroS&P*. IEEE, 2019, pp. 432–447.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "Actively Secure OT Extension with Optimal Overhead." In: *CRYPTO (1)*. Vol. 9215. LNCS. Springer, 2015, pp. 724–741.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer." In: *ACM CCS*. ACM, 2016, pp. 830–842.

[Lau14] Ben Laurie. "Certificate Transparency." In: *ACM Queue* 12.8 (2014), pp. 10–19.

[LG15] Wouter Lueks and Ian Goldberg. "Sublinear Scaling for Multi-Client Private Information Retrieval." In: *Financial Cryptography*. Vol. 8975. LNCS. Springer, 2015, pp. 168–186.

[Lip12] Helger Lipmaa. "Secure Accumulators from Euclidean Rings without Trusted Setup." In: *ACNS*. Vol. 7341. LNCS. Springer, 2012, pp. 224–240.

[LLN+16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. "Zero-Knowledge Arguments for Lattice-Based Accumulators: Logarithmic-Size Ring Signatures and Group Signatures Without Trapdoors." In: *EUROCRYPT (2)*. Vol. 9666. LNCS. Springer, 2016, pp. 1–31.

[LN17] Yehuda Lindell and Ariel Nof. "A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority." In: *CCS*. ACM, 2017, pp. 259–276.

[LN18] Yehuda Lindell and Ariel Nof. "Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody." In: *ACM CCS*. ACM, 2018, pp. 1837–1854.

[Mer89] Ralph C. Merkle. "A Certified Digital Signature." In: *CRYPTO*. Vol. 435. LNCS. Springer, 1989, pp. 218–238.

[MGG+13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. "Zerocoin: Anonymous Distributed E-Cash from Bitcoin." In: *IEEE S&P*. IEEE, 2013, pp. 397–411.

[NFH+09] Toru Nakanishi, Hiroki Fujii, Yuta Hira, and Nobuo Funabiki. "Revocable Group Signature Schemes with Constant Costs for Signing and Verifying." In: *PKC*. Vol. 5443. LNCS. Springer, 2009, pp. 463–480.

[Ngu05] Lan Nguyen. "Accumulators from Bilinear Pairings and Applications." In: *CT-RSA*. Vol. 3376. LNCS. Springer, 2005, pp. 275–292.

[NKH+05] Toru Nakanishi, Fumiaki Kubooka, Naoto Hamada, and Nobuo Funabiki. "Group Signature Schemes with Membership Revocation for Large Groups." In: *ACISP*. Vol. 3574. LNCS. Springer, 2005, pp. 443–454.

[NS04] Toru Nakanishi and Yuji Sugiyama. "A Group Signature Scheme with Efficient Membership Revocation for Reasonable Groups." In: *ACISP*. Vol. 3108. LNCS. Springer, 2004, pp. 336–347.

[OSV20] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. "Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption." In: *CT-RSA*. Vol. 12006. LNCS. Springer, 2020, pp. 254–283.

[PTT08] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. "Authenticated hash tables." In: *ACM CCS*. ACM, 2008, pp. 437–448.

[SA19] Nigel P. Smart and Younes Talibi Alaoui. "Distributing Any Elliptic Curve Based Protocol." In: *IMACC*. Vol. 11929. LNCS. Springer, 2019, pp. 342–366.

[San99] Tomas Sander. "Efficient Accumulators without Trapdoor Extended Abstracts." In: *ICICS*. Vol. 1726. LNCS. Springer, 1999, pp. 252–262.

[Sha79] Adi Shamir. "How to Share a Secret." In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[SSU16] Daniel Slamanig, Raphael Spreitzer, and Thomas Unterluggauer. "Linking-Based Revocation for Group Signatures: A Pragmatic Approach for Efficient Revocation Checks." In: *Mycrypt*. Vol. 10311. LNCS. Springer, 2016, pp. 364–388.

[Ver16] Eric R. Verheul. "Practical backward unlinkable revocation in FIDO, German e-ID, Idemix and U-Prove." In: *IACR ePrint* 2016 (2016), p. 217.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. "SecureNN: 3-Party Secure Computation for Neural Network Training." In: *PoPETs* 2019.3 (2019), pp. 26–49.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature