Practical Secure Computation in the Client-Server Model

Vos, J.V.

DOI
10.4233/uuid:89791fc2-8d5e-4644-b4c9-810e4a69aa67

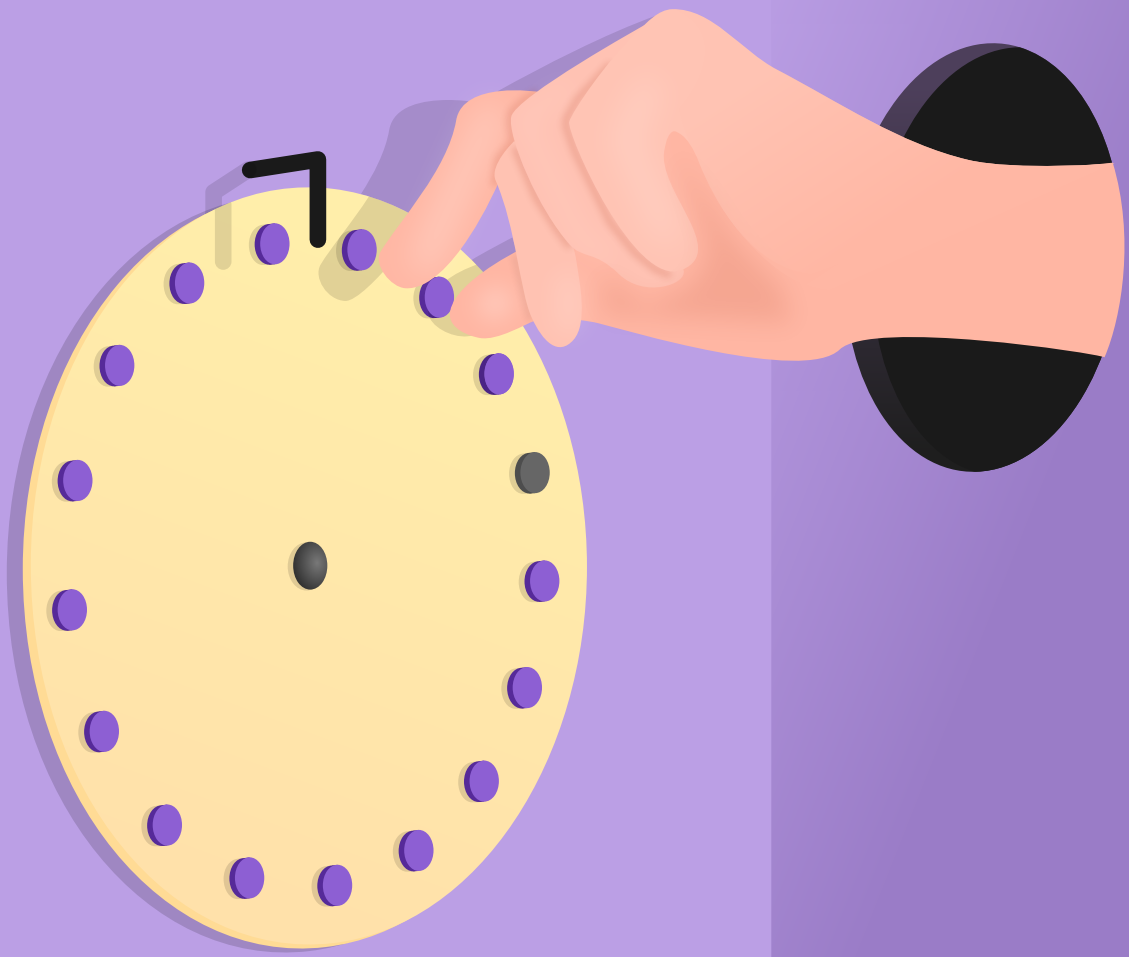
Publication date
2025

Document Version
Final published version

Citation (APA)
Vos, J. V. (2025). *Practical Secure Computation in the Client-Server Model*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:89791fc2-8d5e-4644-b4c9-810e4a69aa67

PRACTICAL SECURE COMPUTATION

IN THE CLIENT-SERVER MODEL



Jelle Vos

Practical Secure Computation in the Client-Server Model

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus Prof.dr.ir. T.H.J.J. van der Hagen
Chair of the Board for Doctorates
to be defended publicly on
Tuesday 17, June 2025 at 10:00 o'clock

by

Jelle VOS
Master of Science in Computer Science
Delft University of Technology, the Netherlands
born in Delft, the Netherlands

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus	chairperson
Dr. Z. Erkin	Delft University of Technology, promotor
Prof.dr. M. Conti	Delft University of Technology, University of Padua, promotor

Independent members:

Prof.dr.ir. R.L. Lagendijk	Delft University of Technology
Prof.dr. M.E. van Dijk	Vrije Universiteit
Dr.ir. L.A.M. Schoenmakers	Eindhoven University of Technology
Dr. S. Samardjiska	Radboud Universiteit
Dr.ir. E. Makri	Universiteit Leiden
Prof.dr. M.T.J. Spaan	Delft University of Technology, reserve member

Cover design: Jelle Vos, inspired by De Correspondent books. The shape of the hand was modeled after the pinch finger gesture designed by 588ku from pngtree.com/freepng/pinching-finger-gesture-cartoon-illustration_4719807.html.

ISBN: 978-94-6518-075-5

Contents

Summary

To compute something securely is to do so in a way that does not reveal (some of) the inputs, intermediate values, or outputs, to certain predetermined parties. For example, a hospital might outsource the computation of patients' medical analyses to the cloud without the cloud provider being able to extract sensitive medical information. Researchers have proposed cryptographic protocols capable of performing secure computation for any function imaginable. However, there are still technical obstacles that hinder practical deployments of secure computation protocols. In this thesis, we identify four such impracticalities and propose techniques to address them.

A crucial building block that forms the basis of all protocols in this thesis is homomorphic encryption. Like 'regular' encryption it protects the encrypted values from being seen. However, it also allows one to perform computations on these encrypted values, without decrypting them. We distinguish between partially-homomorphic encryption schemes, which allow for some specific computations to be performed, and fully-homomorphic encryption schemes, which can perform any computation imaginable.

In the first part of this thesis, we focus on secure multi-party computation, which are protocols between multiple parties who each contribute an input to the computation. An example is a private set intersection, in which each party contributes a private set and the output reveals the elements they have in common. Such protocols may be used to identify fraud in patients who collect certain medicine from multiple hospitals. Current protocols often require the parties to perform many interactions or they require all the parties to communicate with each other. In this part, we address these two impracticalities by designing and analyzing private set operation protocols based on partially-homomorphic encryption. Given that our protocols only require few rounds of communication between each party and one central party, they are in line with the client-server model, which models the architecture of conventional, non-secure computations.

In the second part, we focus on two impracticalities regarding fully-homomorphic encryption, which may be used to securely outsource computations from lightweight clients to powerful servers. Firstly, it is currently computationally expensive to use fully-homomorphic encryption to compute complex functions. Secondly, doing so requires expert knowledge. In this part, we provide algorithms that allow non-experts to securely perform complicated computations using standard programming primitives, and to do so more efficiently than before. We implement these algorithms in a new secure computation compiler called oraqle, and they are also being implemented in Google's HEIR compiler.

Before wrapping up the thesis, we provide initial results on an extension of the oraqle compiler that is capable of generating secure multi-party computation protocols automatically. We show that this extension can generate one of the protocols that we designed by hand in the first part of this thesis. We conclude that the techniques that we propose are promising, but we cannot solve the impracticalities in all cases: depending on the computation being performed, fully-homomorphic encryption techniques remain computationally expensive. We propose several future directions that may further reduce the computational cost of secure computation.

Samenvatting

Het op een veilige manier uitvoeren van berekeningen houdt in dat de ingevoerde data, tussentijdse berekeningen, en de uitkomsten van de berekeningen geheim blijven voor een groep aangewezen partijen. Een voorbeeld hiervan is een ziekenhuis dat haar berekeningen uitbesteedt aan een *cloud provider*, maar op zo'n manier dat de cloud provider niet bij de gevoelige medische data kan. Onderzoekers hebben al technieken ontwikkeld om elke mogelijke functie op een veilige manier te berekenen door middel van cryptografische protocollen, alleen hebben deze technieken nog tekortkomingen die het gebruik in de praktijk belemmeren. In dit proefschrift benoemen wij vier van deze tekortkomingen en ontwikkelen wij technieken om deze te verhelpen.

Aan de basis van alle protocollen die wij ontwikkelen in dit proefschrift ligt een bestaande bouwsteen genaamd homomorfische versleuteling. Deze vorm van versleuteling voorkomt dat versleutelde data zichtbaar is, en tegelijk staat het computers toe om berekeningen uit te voeren op deze versleutelde data zonder deze te ontsleutelen. Gedeeltelijke homomorfische versleuteling laat men specifieke berekeningen uitvoeren, terwijl volledige homomorfische versleuteling in staat is elke mogelijke berekening uit te voeren.

In het eerste deel van dit proefschrift ligt de focus op protocollen waarin meerdere partijen een invoer kunnen bijdragen. Een voorbeeld van zo'n *multi-party computation* protocol is een *private set intersection*, waarin elke partij een geheime verzameling invoert. De uitvoer van dit protocol is de doorsnede van al deze verzamelingen (de elementen die overeenkwamen tussen alle verzamelingen). Zo'n protocol kan bijvoorbeeld gebruikt worden om fraude in ziekenhuizen tegen te gaan door patiënten op te sporen die bepaalde medicijnen opvragen van meerdere ziekenhuizen. De huidige protocollen vereisen vaak dat de partijen veel en allemaal met elkaar communiceren. In dit deel ontwikkelen en analyseren wij protocollen voor *private set operations* gebaseerd op gedeeltelijke homomorfische versleuteling waarvoor dit niet geldt. Omdat partijen in deze protocollen minder communiceren, en slechts met één centrale partij, passen deze protocollen in het client-servermodel dat overeenkomt met de architectuur van veel huidige systemen voor conventionele (niet-veilige) berekeningen.

In het tweede deel ligt de focus op de bruikbaarheid van volledige homomorfische versleuteling, wat gebruikt kan worden voor het uitbesteden van berekeningen van zwakke client-computers aan krachtige servers. Op dit moment kost het veel computerkracht om complexe berekeningen uit te voeren met dit soort versleuteling en vereist het de nodige expertise. Daarom ontwikkelen wij algoritmes die het mogelijk maken om zonder deze expertise alsnog ingewikkelde berekeningen uit te voeren, en zelfs efficiënter dan voorheen. Wij hebben deze algoritmes geïmplementeerd in een nieuwe veilige berekenings-*compiler* genaamd oraqle, en ze worden ook in Google's HEIR compiler geïntegreerd.

Aan het eind van dit proefschrift behandelen wij een uitbreiding van de oraqle compiler die in staat is om multi-party protocollen automatisch te genereren. Deze uitbreiding is zelfs in staat één van de protocollen te genereren die wij in het eerste deel van dit proefschrift met de hand ontwierpen. Onze conclusie is dat deze technieken veelbelovend zijn, maar nog niet alle praktische problemen oplossen: Afhankelijk van de beoogde berekening kunnen volledig homomorfische versleutelingstechnieken nog behoorlijk duur zijn om uit te voeren. Om deze reden stellen wij enkele toekomstige richtingen voor die de kosten van veilige berekeningen zouden kunnen drukken.

Introduction

Secure computation is the process of evaluating a function on secret inputs. Consider, for example, a computer model that can predict medical conditions on a patient's medical scan such as a photo, an X-ray, or a CT scan. Now consider that the patient does not wish for the model's operators to see their medical information, but they do intend to learn the model's prediction. In other words, the patient wants to evaluate the computer model (a function) on an input they intend to keep hidden: The patient wants to *securely compute* the model's output.

It seems paradoxical that one can decide how a computer should handle inputs that it cannot see in plain. Indeed, for a long time, it was unknown how to securely compute any possible function until Yao [Yao82] provided a general method in 1982. This method is based on one-way functions; functions that are easy to compute in one direction but hard to invert. Since then, many more secure computation techniques have been proposed, and they have even been applied in practice in several use cases. For example, Apple [App21] and Google [WPY19] use secure computation to check if a user's password is compromised, without leaking the password to their servers.

Yao's work described above can be classified as a means of realizing secure *multi-party* computation (MPC). These secure computation techniques involve at least two parties, say Alex and Blake, who collaborate to compute some function $f(A, B)$ that incorporates both their inputs. In this function, A represents Alex' inputs and B represents Blake's.

Until recently, it was unknown whether it is also possible to securely outsource the computation of a function on one party's inputs to another party, ideally without requiring multiple interactions between the two parties. In this setting, called secure *outsourced* computation (SOC), Alex would perform minimal work, while Blake computes $f(A)$ on Alex' behalf. In 2009, Gentry [Gen09] found a theoretical solution to this problem in the form of a fully homomorphic encryption (FHE) scheme. Such a scheme allows any function computation to be securely outsourced by letting Alex encrypt their inputs, enabling Blake to compute the function using homomorphisms (see Sec. 7.2.3) and returning the encrypted result.

At this point, we know how to theoretically securely compute any function $f(\dots)$. However, the challenge that remains is to do so efficiently, or more generally, in a way that is *practical*. By efficient, we mean that these techniques may be such that one is willing to pay the price of using them. In the real world, such a price may be a combination of the computational cost, costs relating to the use of the communication network, and the total run time of a secure computation protocol. By practical, we mean that these techniques may be deployed in the real world without facing significant technical obstacles. This thesis considers several of the issues that currently make secure computation impractical and proposes initial steps around these issues. Specifically, we consider four impracticalities:

1. The high number of interactions required in secure computation protocols.
2. The fact that many secure computation protocols require all parties to communicate with each other.
3. The complexity of expressing desired computations using arithmetic operations by hand, which requires expert knowledge.
4. The high computational cost of fully homomorphic encryption.

We proceed to explain these impracticalities in more detail.

Practical secure computation & existing techniques

Given that we know how to compute any function securely, it is perhaps surprising that secure computation techniques are not yet widely deployed. After all, there are plenty of use cases:

- Researchers could compute statistics for medical studies without revealing patients' personal medical information.
- Users could query large language models from lightweight devices without revealing their queries to the server.
- Countries could detect impending collisions between secret satellites without revealing the location of other satellites.
- Organizations could run collaborative analyses to detect cyber threats without revealing their customers' information.
- Security agencies could prevent conflicting investigations without revealing information about other investigations.

The reason why we have not seen secure computation applied to the these use cases in the real world is that there are still challenges that make these techniques impractical to deploy. We discuss these impracticalities in more detail for two contexts: secure multi-party computation and secure outsourced computation. We also briefly explain how existing techniques relate to these impracticalities (we provide detailed overviews of prior work with respect to specific sub-problems in later chapters of this thesis).

Impracticalities in secure multi-party computation

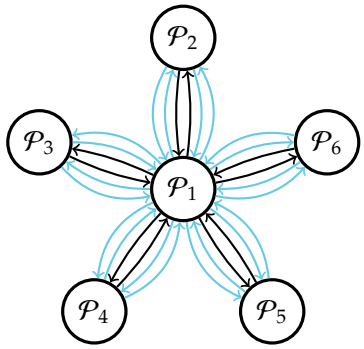
In the context of MPC, one of these issues was already noticed in 1990, namely that many secure computation protocols require a large degree of interaction (i.e. a large number of communication rounds) between the parties involved:

> " For many concrete computations, the resulting number of rounds would be prohibitive; in distributed computation, the number of rounds is generally the most valuable resource. "
>
> — Beaver, Micali, and Rogaway, 1990 [BMR90]

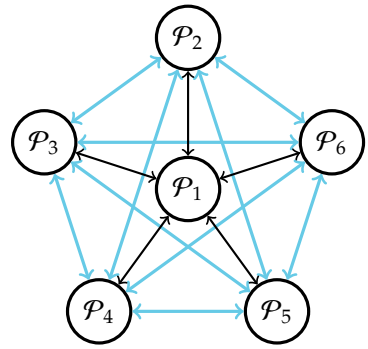
The reason that interactions are so expensive is twofold. First, interaction requires synchronization, and any party's asynchrony may lead to other parties idling. Secondly (and more importantly), every interaction causes delays between the time that a message is sent and when it is received. The only way to reduce this time is to speed up communication lines or increase their bandwidth, but they are inevitably constrained by the speed of light: even in a vacuum, each interaction in a protocol between Europe and the USA will take at least 20 milliseconds.

In other words, our first major impracticality is 1: High interactivity. Figure 1a presents an example in which multiple parties have to interact in many rounds with central party $\mathcal{P}_1$. Unfortunately, many multi-party computation techniques require a large degree of interaction, as we will discuss later in more detail.

Another impracticality originates from the fact that MPC techniques do not always follow the same architecture as conventional distributed computing tasks. Specifically, conventional distributed applications such as private messaging, e-commerce, or online video games, typically follow the client-server model, in which computations are centralized around one or more powerful servers acting in unison. The clients in such a system deploy lightweight computer systems that strictly communicate with a server. This is in stark contrast to information-theoretic MPC, which requires parties to communicate with all or at least a sizeable subset of other parties. Figure 1b illustrates that, whereas the client-server model implies a star topology, MPC techniques often assume a full-mesh topology. As a result, clients must be significantly more powerful and available, and existing infrastructure built around the client-server must be overhauled. In other words, a second major impracticality is 2: Full-mesh topology.



(a) Impracticality #1, many MPC techniques require a large number of interactions.



(b) Impracticality #2, many MPC techniques require a full-mesh topology.

Figure 1: Impracticalities originating in secure multi-party computation techniques.

Next, we briefly underline *why* current techniques suffer from these two impracticalities. While there are many techniques for secure multi-party computation, the term 'generic MPC' has become synonymous with information-theoretic MPC. In these techniques, parties split their secrets into shares that do not reveal anything about the secrets when viewed in isolation. They send these shares to multiple other parties, who can then perform computations between shares, obtaining new shares. Typically, secret shares encode plaintext messages in the ring $\mathbb{Z}_q$. A simple

secret sharing scheme would be to split such a message m_1 into n random shares $x_1, \ldots, x_n$ in a way that $x_1 + \cdots + x_n = m_1$. Say that we also have another message m_2 that is split into shares $y_1, \ldots, y_n$, then we can perform additions between them by adding the individual shares:

$$(x_1 + y_1) + \cdots + (x_n + y_n) = m_1 + m_2 \ .$$

It turns out that performing multiplications in this scheme requires at least one interaction between all the involved parties. As such, these techniques suffer both from 1: High interactivity and 2: Full-mesh topology.

Other generic MPC approaches circumvent some of these impracticalities by relying on cryptographic hardness assumptions. For example, a protocol that requires a full-mesh topology may be transformed into a protocol in the star topology by using symmetric-key cryptography to route all communication confidentially through the central server. However, this creates a bottleneck, and while it solves 2: Full-mesh topology, it does not solve 1: High interactivity. On the other hand, there are secure computation techniques in the form of homomorphic secret sharing that realize two-round protocols for all functions [BGI16; Boy+18], but they require expensive asymmetric-key cryptography and their efficiency strongly depends on the computed function. Other techniques for MPC include function secret sharing [BGI15], multi-party garbling [BMR90], and threshold partially-homomorphic cryptosystems [CDN01]. These techniques either require many parties to communicate with each other, or they require a large number of interactions (a high round complexity) to evaluate complex functions.

In other words, current generic techniques typically suffer from the aforementioned impracticalities, or they are inefficient. That said, for specific functionalities, it may be possible to find a practical trade-off between these three properties.

Impracticalities in secure outsourced computation

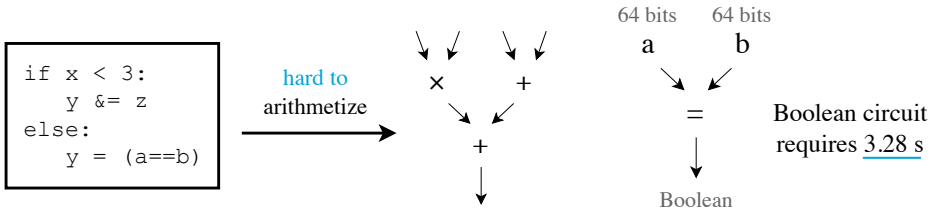
Secure multi-party computation techniques do not generally translate to secure outsourced computation techniques: since they typically involve the efforts of multiple parties, the efforts cannot be easily centralized to one powerful server. For this reason, low-round homomorphic secret sharing schemes are not suitable in the context of SOC. Of course, one may still distribute these computations across multiple servers, but this is at odds with the client-server model. Moreover, distributing the computation across two or three servers typically involves a *non-collusion* assumption that is hard to realize in practice.

A more suitable secure outsourced computation technique comes in the form of homomorphic encryption, which is an encryption scheme that permits one or more homomorphisms between the ciphertext and plaintext spaces. In other words, a client could outsource a plaintext computation to a server by encrypting it, sending it to the server, who applies a certain computation to the ciphertext domain to reflect the intended computation in the plaintext domain. A fully-homomorphic encryption scheme allows *any* computation to be outsourced in this fashion, which addresses both impracticalities 1: High interactivity and 2: Full-mesh topology. This makes it 'the Holy Grail of cryptography':

> " The idea of fully homomorphic encryption schemes was first proposed by Rivest, Adleman, and Dertouzos in the late 1970s, but remained a mirage for three decades, the never-to-be-found Holy Grail of cryptography. At least until 2008, when Craig Gentry announced a new approach to the construction of fully homomorphic cryptosystems. "
>
> — Micciancio, 2010 [Mic10]

Since their invention, fully-homomorphic encryption (FHE) schemes have seen significant developments in virtually all aspects, such as computational efficiency, key sizes, and user-friendly implementations. Unfortunately, FHE still suffers from impracticalities. First, while writing programs in conventional programming languages is widely practiced with the necessary tools for automation and support, it is notoriously hard to design cryptographic protocols for secure computation tasks that are correct, secure, *and* efficient. This is not a task that is easily tackled by a non-expert, because current FHE schemes require programs to be expressed in terms of additions and multiplications (and automorphisms) in some algebraic structure. There are already major efforts to automate this process called *arithmetization* in the context of FHE in the form of homomorphic encryption compilers (see Sec. 6.2 for an overview of these compilers), but many compilers leave arithmetization to the user, while the ones that do perform automatic arithmetization only consider a fraction of the parameter space. We summarize this impracticality as 3: Arithmetization, as portrayed in Figure 2a.

The second and most straightforward impracticality is that FHE is computationally expensive: non-linear operations take orders of magnitude longer than equivalent non-secure operations. For example, as shown in Figure 2b, a homomorphic encryption scheme computing an equality check between two sequences of 64 bits requires 3.28 seconds to compute (see Table 6.2), whereas a modern computer can perform such an equality check on plaintext values in the order of nanoseconds. We denote this with 4: Compute-intensive.



(a) Impracticality #3, arithmetization beyond Boolean or LUT circuits requires expert knowledge.

(b) Impracticality #4, evaluating HE circuits is computationally expensive.

Figure 2: Impracticalities originating in secure outsourced computation techniques.

The reason why 3: Arithmetization in general is a hard problem, is that these homomorphic encryption schemes tend to have a plaintext algebra of the form $\mathbb{Z}_q$, where q is some natural number, or some polynomial ring. The existing techniques for synthesizing arithmetic circuits, which are comprised of additions and multiplications, are based on Boolean circuits, because these are used for hardware

design. Operations such as comparisons (i.e. $x < y$) have no formal definition in these plaintext algebras and potential arithmetizations strongly depend on the algebra. For this reason, homomorphic encryption compilers typically only perform Boolean circuit synthesis. Alternatively, compilers generate LUT circuits consisting of additions and look-up tables (LUTs), which can be conveniently computed using homomorphic encryption schemes such as TFHE [Chi+18]. However, these look-up tables can only be applied to one element at a time, whereas other homomorphic encryption schemes such as BGV [BGV12] allow the efficient evaluation of arithmetic circuits on many elements in parallel.

The homomorphic encryption schemes that are currently in use by homomorphic encryption compilers are based on (variants of) the learning with errors assumption [Reg05]. A newly-encrypted 'fresh' ciphertext in these schemes contains two parts: uniform randomness and a noisy linear combination of this randomness as determined by the secret key. The reason why non-linear operations on such a ciphertext are expensive is that they cause the ciphertext to grow in the number of parts, requiring an expensive process called 'relinearization' to reduce the number of parts back to two.

Another computationally-expensive operation in these schemes is called re-encryption or bootstrapping, which is necessary due to the fact that ciphertexts are noisy. After performing homomorphic operations, and especially after performing multiplications, the noise may grow and eventually overwrite the encrypted plaintext, hindering successful decryption. Before the ciphertext reaches this point, one may use a bootstrapping key to non-interactively refresh the noise level in the ciphertext. Bootstrapping and relinearization operations are the main factors contributing to impracticality 4: Compute-intensive, and both are a consequence of homomorphic multiplications.

An alternative to FHE that requires a single interaction (thereby optimally solving impracticality 1: High interactivity) is called functional encryption [SW05]. Like any other encryption schemes, functional encryption allows a party to encrypt some plaintext x. However, their decryption differs in that they require a function-specific decryption key sk_f that allows the decrypter to non-interactively obtain $f(x)$ without learning anything else about x. Some functional encryption schemes can securely compute any function [Gol+13; GVW13], however these schemes internally use similar constructions to the aforementioned fully-homomorphic encryption schemes based on the learning with errors assumption. As a result, they suffer from the same impracticalities. Other schemes like the scheme by Garg et al. [Gar+13] rely on hard-to-realize primitives like multilinear maps.

Research questions

We started this introduction with a rather imprecise definition of secure computation. To properly define the aim of this thesis, we first refine this definition. In its broadest form, secure computation can be defined as any computation that involves operands that must remain confidential. In this thesis, we consider a stricter definition of what makes a computation secure: we require the computation to remain confidential in the presence of (augmented) honest-but-curious colluding

parties. Such parties faithfully execute their role in a protocol, but they attempt to learn whatever additional information they can (see Sec. 1.2.3). In other words, due to the collusion-resistance requirement, we do not consider secure computation protocols in which a trusted third party performs the computation under the promise that they will indeed keep secrets secret. Similarly, we do not consider secure enclaves, in which a party must place their trust in hardware and software to protect secret keys.[1]

Within this scope, the goal of this thesis is to address the four major impracticalities mentioned above. We answer the following research question:

> How can we make collusion-resistant confidential computation between honest-but-curious parties more practical?

For secure multi-party computation, the open problem is to design efficient protocols in the client-server model that require few interactions for operations that are currently impractical. In other words, we aim to address the problems related to communication. For secure outsourced computation, the open problem is that high-level functions are currently impractical to evaluate: this requires expert knowledge and current approaches lead to inefficient circuits.

Addressing impracticalities in MPC

A general rule in secure computation is that the simpler the function, the more practical and efficient it is to compute. For example, there are practical protocols for computing a summation between multiple values, such as Prio [CB17]. However, more complex functions quickly become impractical to evaluate. In this thesis, we aim to address the aforementioned impracticalities for functions with potential real world applications that are more complex than summations.

One such class of functions, is the class of set operations such as intersections, unions, and membership queries. This class has been studied for more than 20 years under the name multi-party private set operations (MPSO) [KS05]. We present a detailed overview of multi-party private set intersection protocols in Chapter 1, and other private set operations in Chapters 3 & 4. These protocols use a wide variety of multi-party computation techniques. However, there is still a gap in the literature when it comes to designing efficient protocols in the client-server model that require few interactions.

In Part A of this thesis, we aim to fill this gap by proposing low-round homomorphic-encryption based protocols. Using homomorphic encryption allows us to address impracticality 2: Full-mesh topology. We answer the following sub-question:

> What are computationally-efficient, low-round, collusion-resistant multi-party private set operation protocols between honest-but-curious parties in the client-server model?

As such, we limit our scope from multi-party computation to multi-party private set operations.

[1] A secure enclave can be seen as a non-colluding key holder, where collusion entails extracting the key or other hidden data from the enclave.

Addressing impracticalities in SOC

As mentioned previously, homomorphic encryption compilers have already made it significantly easier for non-experts to evaluate high-level circuits using homomorphic encryption. However, these compilers typically leave arithmetization to the user. We aim to democratize the use of homomorphic encryption by proposing algorithms for the arithmetization of several high-level operations, so that homomorphic encryption becomes practical for non-experts, too.

An impracticality that has arguably received more attention from the research community is impracticality 4: Compute-intensive. Since Gentry's initial work [Gen09], the computational cost of FHE has been reduced significantly [GMT23]. Research still continues to reduce the computational costs of individual homomorphic encryption operations, for example, through specialized hardware [Gee+23] or new software implementation techniques [Bel+24]. However, further reductions can also be achieved by eliminating homomorphic operations in the first place. We aim to do so by generating arithmetic circuits that require fewer or less expensive homomorphic encryption operations. For example, by considering a different plaintext space or computational model, and by exploiting implicit parallelization.

In short, our aim in Part B is to address impracticalities 3: Arithmetization and 4: Compute-intensive by proposing methods for generating arithmetic circuits that are computationally cheaper to evaluate than current proposals. We answer the following sub-question:

> Given a high-level circuit, how can one efficiently generate an arithmetic circuit that is computationally cheap to evaluate using fully homomorphic encryption?

As such, we limit the scope from secure outsourced computation in general, to homomorphic encryption-based secure outsourced computation.

Addressing impracticalities in general

Ideally, our work does not only address the impracticalities of MPC and SOC in isolation, but also together. After all, even when multiple parties contribute inputs (MPC), it is in the spirit of the client-server model to outsource most of these computations to a server (SOC). While a significant amount of research effort has gone into tools for generic MPC protocols and homomorphic encryption compilers, we aim to unify these tools. This is the aim for the last part of this thesis, titled 'The road ahead'. In this part, we provide initial answers to the following sub-question:

> Given a high-level circuit, a description of parties' knowledge, and a secure computation technique, how can one generate a corresponding efficient secure multi-party computation protocol?

Detailed thesis outline

Table 1 provides an overview of the parts in this thesis. The columns indicate whether a part concerns secure outsourced computation or secure multi-party

computation. The rows indicate whether we consider protocols that were manually designed and optimized, or whether we provide algorithms for doing so automatically. We continue by outlining each chapter in the order in which they appear in this thesis: this thesis is made up of eight chapters that are adaptations from existing research article. Some of these articles have been published, whereas others have not yet been accepted or have only been made available as a pre-print. An exception is Chapter 8, which has not been made available before. We note that, because the adapted articles are independent, notation between chapters may sometimes be inconsistent. Moreover, while we took care to minimize repetition, some concepts may be introduced multiple times. The end of this thesis is marked by the discussion chapter.

Table 1: Overview of the parts in this thesis. Our third part titled 'The road ahead' combines insights from both Part A and Part B.

		Computation	
		Outsourced	Multi-party
Generation	Manual	-	Part A
	Automatic	Part B	The road ahead

Part A: Efficient MPSO protocols in the star topology

In Part A, we analyze and propose secure multi-party computation protocols for set operations, such as multi-party private set intersection (MPSI) protocols computing $f(X_1, \ldots, X_n) = X_1 \cap \cdots \cap X_n$, set membership queries computing $f(x, X_1, \ldots, X_n) = x \in (X_1 \cup \cdots \cup X_n)$, and multi-party private set union (MPSU) protocols computing $f(X_1, \ldots, X_n) = X_1 \cup \cdots \cup X_n$, where $X_1, \ldots, X_n$ are all sets and x is a single element.

Chapter 1: SoK: Collusion-resistant MPSI in the semi-honest model

Published in IEEE Symposium on Security and Privacy 2024

We provide a systematization for multi-party private set intersection protocols that resist passive attacks from a given number of colluding parties.

Chapter 2: On the insecurity of Bloom filter-based PSI

Submitted to the IACR Theory of Cryptography Conference 2025

We demonstrate that while small Bloom filters are useful data structures for constructing low-round and efficient private set intersection protocols, they are unsuitable for achieving confidentiality. We show that previous security proofs were incorrect, showing that it is impossible to achieve a tight security proof, and by proposing practical attacks. We provide several mitigations.

Chapter 3: Fast MPSO in the star topology from secure ANDs and ORs

Revision of a pre-print made available on the IACR Cryptology ePrint Archive in 2022

We provide a collection of efficient protocols for multi-party private set intersections and unions, based on the decisional Diffie-Hellman assumption.

Chapter 4: Privacy-preserving membership queries for federated anomaly detection

Published in Proceedings on Privacy Enhancing Technologies 2024

We introduce the concept of repeated membership queries, allowing a party to query membership of single element in static sets held by multiple other parties, without revealing the query nor the contents of the sets. Along the way we propose a new abstract primitive akin to oblivious key-value stores called encrypted membership query filters (EMQFs) that may be of independent interest.

Part B: Automatic generation of HE circuits

In Part B, we consider that designing efficient secure outsourced computation protocols currently requires expert knowledge. We address this problem by proposing algorithms that generate efficient circuits of primitive operations for high-level tasks.

Chapter 5: Depth-aware arithmetization of common primitives in prime fields

Submitted to USENIX Security 2025

We propose the concept of *depth-aware* arithmetization, which translates high-level circuit specifications into efficient arithmetic circuits, while at the same time exploring the trade-off between the number of multiplications and squarings in an arithmetic circuit and the number of multiplications on any path through the circuit.

Chapter 6: A depth-aware secure computation compiler

Published in 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography

We propose the oraqle compiler, which implements depth-aware arithmetization. By implementing several heuristics and integrating previous work for parameter selection and the evaluation of homomorphic encryption circuits, it allows one to efficiently generate homomorphic encryption circuits.

Chapter 7: Efficient circuits for permuting and mapping packed values across leveled homomorphic ciphertexts

Published in European Symposium on Research in Computer Security 2022

We propose an algorithm based on graph coloring that generates efficient homomorphic encryption circuits for permuting elements across multiple packed ciphertexts. We also extend it to perform mappings of any kind.

The road ahead

The final part in this thesis aims at unifying the ideas proposed in parts A & B.

Chapter 8: Optimal automated protocol design for HE-based MPSI

In the final technical chapter of this thesis, which has not been published before, we demonstrate a future direction for practical secure computation in the client-server model: the automatic realization and optimization of HE-based MPC protocols. In this chapter, we provide extensions to the oraqle compiler that allow it to generate multi-party private set intersection protocols that achieve competitive performance with those that were manually designed.

References

[App21] Apple Inc. *Password Monitoring*. https://support.apple.com/guide/security/password-monitoring-sec78e79fc3b/web. Accessed: 2024-10-04. 2021.

[Bel+24] Mariya Georgieva Belorgey et al. "Revisiting Key Decomposition Techniques for FHE: Simpler, Faster and More Generic". In: *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part I*. Ed. by Kai-Min Chung and Yu Sasaki. Vol. 15484. Lecture Notes in Computer Science. Springer, 2024, pp. 176–207. DOI: 10.1007/978-981-96-0875-1_6. URL: https://doi.org/10.1007/978-981-96-0875-1%5C_6.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 337–367. DOI: 10.1007/978-3-662-46803-6_12. URL: https://doi.org/10.1007/978-3-662-46803-6%5C_12.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Breaking the Circuit Size Barrier for Secure Computation Under DDH". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, 2016, pp. 509–539. DOI: 10.1007/978-3-662-53018-4_19. URL: https://doi.org/10.1007/978-3-662-53018-4%5C_19.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. Ed. by Shafi Goldwasser. ACM, 2012, pp. 309–325. DOI: 10.1145/2090236.2090262. URL: https://doi.org/10.1145/2090236.2090262.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The Round Complexity of Secure Protocols (Extended Abstract)". In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. Ed. by Harriet Ortiz. ACM, 1990, pp. 503–513. DOI: 10.1145/100216.100287. URL: https://doi.org/10.1145/100216.100287.

[Boy+18] Elette Boyle et al. "Foundations of Homomorphic Secret Sharing". In: *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*. Ed. by Anna R. Karlin. Vol. 94. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 21:1–21:21. DOI: 10.4230/LIPICS.ITCS.2018.21. URL: https://doi.org/10.4230/LIPIcs.ITCS.2018.21.

[CB17] Henry Corrigan-Gibbs and Dan Boneh. "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics". In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. Ed. by Aditya Akella and Jon Howell. USENIX Association, 2017, pp. 259–282. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. "Multiparty Computation from Threshold Homomorphic Encryption". In: *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*. Ed. by Birgit Pfitzmann. Vol. 2045. Lecture Notes in Computer Science. Springer, 2001, pp. 280–299. DOI: 10.1007/3-540-44987-6_18. URL: https://doi.org/10.1007/3-540-44987-6%5C_18.

[Chi+18] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption over the Torus". In: *IACR Cryptol. ePrint Arch.* (2018), p. 421. URL: https://eprint.iacr.org/2018/421.

[Gar+13] Sanjam Garg et al. "Attribute-Based Encryption for Circuits from Multilinear Maps". In: *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8043. Lecture Notes in Computer Science. Springer, 2013, pp. 479–499. DOI: 10.1007/978-3-642-40084-1_27. URL: https://doi.org/10.1007/978-3-642-40084-1%5C_27.

[Gee+23] Robin Geelen et al. “BASALISC: Programmable Hardware Accelerator for BGV Fully Homomorphic Encryption”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023.4 (2023), pp. 32–57. DOI: 10.46586/TCHES.V2023.I4.32-57. URL: https://doi.org/10.46586/tches.v2023.i4.32-57.

[Gen09] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[GMT23] Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. “SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks”. In: *Proc. Priv. Enhancing Technol.* 2023.3 (2023), pp. 154–172. DOI: 10.56553/POPETS-2023-0075. URL: https://doi.org/10.56553/popets-2023-0075.

[Gol+13] Shafi Goldwasser et al. “Reusable garbled circuits and succinct functional encryption”. In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM, 2013, pp. 555–564. DOI: 10.1145/2488608.2488678. URL: https://doi.org/10.1145/2488608.2488678.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. “Attribute-based encryption for circuits”. In: *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM, 2013, pp. 545–554. DOI: 10.1145/2488608.2488677. URL: https://doi.org/10.1145/2488608.2488677.

[KS05] Lea Kissner and Dawn Xiaodong Song. “Privacy-Preserving Set Operations”. In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 241–257. DOI: 10.1007/11535218_15. URL: https://doi.org/10.1007/11535218%5C_15.

[Mic10] Daniele Micciancio. “A first glimpse of cryptography's Holy Grail”. In: *Commun. ACM* 53.3 (2010), p. 96. DOI: 10.1145/1666420.1666445. URL: https://doi.org/10.1145/1666420.1666445.

[Reg05] Oded Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. Ed. by Harold N. Gabow and Ronald Fagin. ACM, 2005, pp. 84–93. DOI: 10.1145/1060590.1060603. URL: https://doi.org/10.1145/1060590.1060603.

[SW05] Amit Sahai and Brent Waters. "Fuzzy Identity-Based Encryption". In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 457–473. DOI: 10.1007/11426639_27. URL: https://doi.org/10.1007/11426639%5C_27.

[WPY19] Amanda Walker, Sarvar Patel, and Moti Yung. *Helping organizations do more without collecting more data*. https://security.googleblog.com/2019/06/helping-organizations-do-more-without-collecting-more-data.html. Accessed: 2024-10-04. June 2019.

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38. URL: https://doi.org/10.1109/SFCS.1982.38.

Part A

Efficient MPSO protocols in the star topology

Chapter 1

SoK: Collusion-resistant Multi-party Private Set Intersections in the Semi-honest Model

Multi-party private set intersection protocols have been around for more than 20 years. As a result, there are many different protocols with different characteristics. This chapter provides an overview of these protocols and their characteristics: It proposes a systematization that captures all collusion-resistant MPSI protocols. In doing so, it also discusses to what degree they are hindered by impracticalities 1: High interactivity and 2: Full-mesh topology.

Our main conclusions are that older protocols are still competitive with more recent protocols, at least with regards to their asymptotic complexity. More generally, almost all of the protocols that we analyzed excel in specific contexts. However, with the current lack of practical use cases for MPSI, it is not certain what context is realistic in practice, and therefore, if there is a specific protocol that we can consider to be the state of the art.

This chapter is an adaptation of the work with the same title that has been published in IEEE Symposium on Security and Privacy 2024, authored by Jelle Vos, Mauro Conti, and Zekeriya Erkin.

1.1 Introduction

In 2004, Freedman et al. [FNP04] proposed the first custom protocol for privately computing the intersection between multiple sets of data. Since then, private set intersection protocols have received a great deal of attention from the research community. Most notably, many concretely efficient two-party protocols have been proposed, which currently perform intersections over sets of 1 million elements in less than 6 seconds over a 100 MBit/s communication channel [KBM22]. However, when protocols must scale to an arbitrary number of parties, performance degrades rapidly [Kol+17]. In this work, we systemize and summarize the current body of literature on such multi-party private set intersections (MPSI), and identify opportunities for future work.

More formally, an MPSI protocol solves the problem where n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with respective private sets $X_1, \ldots, X_n$, want to confidentially compute the intersection $X_1 \cap \cdots \cap X_n$. They do so in the presence of at most t adversaries that may collude with one another. In this work, as for many previous works [Kol+17; HV17; Bay+22], one party $\mathcal{P}_1$ learns the intersection. We refer to this party as the leader, and the other parties $\mathcal{P}_2, \ldots, \mathcal{P}_n$ as assistants.

Motivating applications

One motivating application of *multi-party* private set intersections is that of finding a set of suitable meeting dates between multiple private calendars. Here, X_i would be a set of dates at which party $\mathcal{P}_i$ is available. At the end of the protocol, the leader $\mathcal{P}_1$ receives the set of meeting dates at which all parties are available. Several other applications have been mentioned in previous works. We highlight several below.

Miyaji & Nishida [MN15] and Kolesnikov et al. [Kol+17] mention an application where multiple shop owners or digital services want to launch a collective promotion campaign. To do so, these shops must identify their mutual set of customers, without violating the privacy of the other customers.

Kissner & Song [KS05], Inbar et al. [IOP18] and Kolesnikov et al. [Kol+17] mention cyber security applications. They consider a problem between several organizations who want to catch an intruder in a common network. The idea is that each organization keeps a list of suspicious IP addresses, and by computing the intersection, they narrow them down. Since IP addresses reveal personal information, it is important that the other IP addresses remain private. Similarly, Ghosh et al. [GN19] mention that MPSI can be applied to detect botnets.

Wang et al. [WBU21] discuss an application where an investigative agency needs to narrow down a list of suspects by cooperating with multiple other agencies. Since none of the agencies can share plain data with each other, they engage in an MPSI protocol to only consider relevant suspects.

Freedman et al. [FNP04] and Li & Wu [LW07] also mention that MPSI protocols may be used in online recommendation services, dating services, medical databases, and data mining in general. Kissner & Song discuss further applications, including detecting fraudulent sales from pharmacies, enforcing no-fly lists privately, combining the results of multiple surveys, and governments checking if their ill citizens are actually receiving aid [KS05]. Finally, Poddar et al. [Pod+21] discuss the application of MPSI for private database join operations.

We note that while PSI protocols are called *private* set intersections, this does not necessary imply that no personal data is being revealed. Instead, such protocols only ensure that the intersection is computed confidentially. If the input sets contain data that may never be revealed, the final intersection can still violate data privacy requirements. In other words, MPSI protocols do not necessarily alleviate every service that requires an intersection from possible privacy violations.

Focus of this paper

This paper restricts itself to protocols that do not require any external parties to collaborate. Moreover, we focus on protocols in the semi-honest model, and only mention whether extensions exist to the malicious model. The reasons are as follows:

A trivial way to tackle the MPSI problem is by selecting some trusted third party, who receives all private sets, computes the intersection, and sends the result to the leader. While such a protocol is extremely efficient, the third party sees all the private input sets. To prevent this, some works provide a slightly stronger notion of security assuming that the third party does not collude with any of the other parties, i.e. $t = 1$. As long as this assumption holds, the private information

stays confidential, even from this third party. These protocols are often referred to as server-aided protocols. Note however, that this non-colluding assumption, like the trusted third party assumption, severely restricts the capabilities of an adversary. Moreover, such a protocol may not be trivially altered to allow for $t > 1$. It is for this reason that we consider only MPSI protocols with no external parties, and that support colluding parties.

When it comes to the security model, we only consider semi-honest adversaries. Not only is the semi-honest model often a crucial intermediate step to the malicious model, but typically the main insight behind a custom protocol does not change in the malicious model. We see this in the underlying set encodings used in MPSI protocols, such as polynomial roots, bitsets, garbled Bloom filters, and oblivious key-value stores. These encodings shape the protocol more so than the choice of cryptographic primitives.

State of MPSI

At present, there exist tens of works on MPSI protocols using a variety of underlying encodings, cryptographic primitives, and network topologies. However, many of these works only consider a small amount previous works in their comparison. Many older receive fewer attention in recent works as they are considered irrelevant due to performance comparisons, while this does not consider all possible points of comparison.

At the moment, the fastest works when the number of elements grows large are based on oblivious key-value stores and vector oblivious linear evaluations. However, these works require several interactions between each pair of parties and a large bandwidth cost. It remains an open question to analyze how these protocols behave for different network conditions. Some alternatives only require assistants to communicate with the leader and require a lower total bandwidth. They are typically based on partially homomorphic encryption, at the cost of more computation. The cheapest protocols computation-wise are based on secret sharing, but they suffer from a large bandwidth cost. No current works are comparable in performance and practicality when $n > 2$, compared to the setting with $n = 2$: there is still a demand for more efficient protocols with regards to communication, computation, and the degree of interaction.

Contributions and outline of this paper

The remainder of this paper is structured as follows. In Section 1.2, we formalize the requirements for semi-honest MPSI protocols. In Section 1.3, we go over some preliminaries that form the foundations of these protocols. Then, in Section 1.4 we provide three high-level constructions that fit all protocols, and discuss possible set representations for each of them. After that, in Section 1.5, we provide a comprehensive overview of currently unbroken MPSI protocols based on these set representations. In Section 1.6, we highlight the problems in works that have been broken and analyze common security issues. Finally, in Section 1.7 we present a theoretical performance comparison of the most recent MPSI protocols in each category, and in Sections 1.8 and 1.9 we discuss the results, identify remaining

research directions, and conclude this work. Appendices 1.A and 1.B contain derivations of the performance aspects of the considered MPSI protocols.

1.2 Formal problem description

An MPSI protocol π is a multi-party protocol that computes the intersection between private sets $X_1, \ldots, X_n$ with overwhelming probability:

$$X_1 \cap \cdots \cap X_n \approx \pi(X_1, \ldots, X_n) \,. \tag{1.1}$$

These sets are all subsets of the universe of possible elements $\mathcal{U}$. Moreover, we establish an upper bound k, so $|X_i| \leq k$ for all $i = 1, \ldots, n$. Each set X_i is held by a party $\mathcal{P}_i$ that is semi-honest. In other words, the party faithfully follows the protocol description, but tries to learn as much as possible in the process. For the protocols studied in this work, we consider it sufficient for only the leader $\mathcal{P}_1$ to receive the computed intersection. We present the ideal functionality of such a protocol below.

Definition 1 (MPSI protocol)**.** An MPSI protocol correctly and privately computes the following ideal functionality:

$$f(X_1, \ldots, X_n) = (X_1 \cap \cdots \cap X_n, \Lambda, \ldots, \Lambda) \,,$$

where Λ is the empty string. The upper bound k may be provided as auxiliary information.

Remark 1. Some works consider a case where all parties receive the result. In the semi-honest model it is trivial to transform a protocol where only the leader learns the result to one where all parties do: the leader simply forwards its result to all parties. It is not straightforward to do so in the malicious model [GHL22].

Note that MPSI protocols typically require the elements in the input sets to be mapped to some cryptographic object. For example, the protocol by Kissner & Song [KS05] maps elements to some group to be encrypted as Paillier ciphertexts. As a result, one must select cryptographic parameters that ensure the group is large enough to contain all distinct elements from $\mathcal{U}$.

Remark 2. One way to circumvent choosing larger parameters when $\mathcal{U}$ is too large is to hash elements of the input sets onto the cryptographic object, thereby supporting a larger universe at the risk of collisions. The chance of collisions is negligible for a sufficiently large cryptographic object.

1.2.1 Correctness requirements

In (1.1) and in the remainder of this paper we write $\approx$ to denote that this result could be an approximation. In fact, due to the properties of cryptographic protocols, all schemes considered in this paper are in some way approximations, albeit that they can be arbitrarily accurate. For example, the scheme by Kolesnikov et al. [Kol+17] achieves a failure probability of $2^{-\lambda}$, where λ is a customizable statistical security parameter. Typically, $\lambda = 40$, and in the remainder of this paper we will consider any probability lower than 2^{-40} to be negligible:

Definition 2 (Exact MPSI protocol)**.** The probability that the output of an exact MPSI protocol does not equal the actual intersection does not exceed 2^{-40}:

$$\Pr\left[X_1 \cap \cdots \cap X_n \neq \pi(X_1, \ldots, X_n)\right] \leq 2^{-40} \ .$$

We say that an MPSI protocol is *approximate* if the chosen parameters cause it to suffer from a higher probability of being wrong than 2^{-40}. We measure the accuracy of approximate protocols more precisely by determining the probability ε that a random element x is wrongly claimed to be included or excluded in the computed intersection. Note that in some protocols, x can only be falsely included and never be falsely excluded. We refer to those protocols as satisfying *superset correctness*. We refer to protocols that only false exclude elements of the intersection as satisfying *subset correctness*.

1.2.2 Privacy in the semi-honest model

Since this work considers protocols in the semi-honest model and since the output of an MPSI protocol is deterministic, one can prove that a protocol is privacy-preserving if it is simulatable [Lin17]. The strongest notion of privacy for semi-honest MPSI protocols is called *size-hiding*, which means that the size of each party's set stays secret:

Definition 3 (Size-hiding MPSI)**.** For each party $\mathcal{P}_i$ in MPSI protocol π, there exists a simulator $\mathcal{S}_i$ that generates an indistinguishable view for all possible combinations of inputs. The simulator receives its input X_i and the intersection $X_1 \cap \cdots \cap X_n$. It must hold that:

$$\begin{aligned} &\{\mathcal{S}_i(1^\lambda, X_i, X_1 \cap \cdots \cap X_n)\}_{X_1,\ldots,X_n \subseteq \mathcal{U}} \overset{c}{\equiv} \\ &\{\mathsf{view}_i^\pi(X_1, \ldots, X_n, \lambda)\}_{X_1,\ldots,X_n \subseteq \mathcal{U}} \ . \end{aligned} \tag{1.2}$$

Note that here we provide the assistants with the intersection as well, because even though they are not required to output it, they are allowed to learn this information.

If the size of each party's set is not private, we have the following definition:

Definition 4 (Size-revealing MPSI)**.** For each party $\mathcal{P}_i$ in MPSI protocol π, there exists a simulator $\mathcal{S}_i$ that generates an indistinguishable view for all possible combinations of inputs. The simulator receives its input X_i, set sizes $|X_1|, \ldots, |X_n|$, and the intersection $X_1 \cap \cdots \cap X_n$. It must hold that:

$$\begin{aligned} &\{\mathcal{S}_i(1^\lambda, X_i, |X_1|, \ldots, |X_n|, X_1 \cap \cdots \cap X_n)\}_{X_1,\ldots,X_n \subseteq \mathcal{U}} \overset{c}{\equiv} \\ &\{\mathsf{view}_i^\pi(X_1, \ldots, X_n, \lambda)\}_{X_1,\ldots,X_n \subseteq \mathcal{U}} \ . \end{aligned} \tag{1.3}$$

Remark 3. Size-revealing MPSI protocols can typically be turned into size-hiding MPSI protocols by letting users submit dummy elements in their input sets to always pad their set to size k. These can either be repetitions of other elements in the set or random elements which have a negligible probability of appearing in the resulting intersection.

We consider the privacy definitions above in the presence of $1 < t < n$ colluding parties. These colluding parties are still semi-honest but they share their views with one another to learn as much private information as possible. When we say that a protocol has a collusion resistance of t, this means that even t colluding adversaries cannot break the claimed notion of privacy.

There are protocols that achieve statistical ($\overset{s}{\equiv}$) rather than computational ($\overset{c}{\equiv}$) indistinguishability in (1.2) or (1.3). These protocols are information-theoretically secure; they do not rely on any computational hardness assumptions. Note that such protocols can only achieve a collusion threshold $t < \frac{n}{2}$ [BGW88]. In the best case, computationally-secure protocols can withstand $t = n - 1$ colluding parties.

1.2.3 Privacy in the *augmented* semi-honest model

Some MPSI protocols consider the *augmented* semi-honest model, which allows an adversary to choose a different input than its actual input immediately before running the protocol [HL10]. While this may seem like a stronger form of security, it is not necessarily compatible with the semi-honest model. That is, a protocol proven to be secure in the augmented semi-honest model may not be secure in the semi-honest model. The reason is that the augmented semi-honest model empowers simulators to choose a convenient input for themselves, while a semi-honest simulator must work for every input. Counter-ituitively, this also implies that protocols secure in the malicious model are not necessarily secure in the semi-honest model. On the contrary, protocols in the malicious model *are always* secure in the augmented semi-honest model. For this reason, we include protocols that adopt this model in this work.

Concretely, the benefit of the augmented model to the design of MPSI protocols is that certain precomputations can be done before the start of the protocol. For example, Inbar et al. [IOP18] perform secret sharing ahead of time, after which parties only have to change their local shares during the actual protocol execution.

1.3 Preliminaries

We briefly recall the background for MPSI protocols. In the remainder of this paper, we work over sets of integers. Where an ordering is necessary, we use $1, 2, 3, \ldots$ implicitly. Table 1.1 summarizes the symbols in this work.

1.3.1 Network topologies

The network topology defines the structure of communication channels between the parties involved in a protocol.

In a star topology, each assistant only shares one bidirectional communication channel with the leader, and none with other assistants. Such a topology resembles the ideal world scenario, and thereby provides the minimal number of communication channels necessary to realize an MPSI protocol. The drawback of this topology is that if a malicious leader refuses to communicate, no communication is possible. The number of channels grows as $O(n)$.

In a mesh topology, each party has a bidirectional communication channel with all other parties. As a result, each party has the ability to broadcast. Such a topology has maximum redundancy when malicious parties refuse to communicate. The number of channels grows as $O(n^2)$.

All other topologies sit in between star and mesh topologies. One topology used in earlier MPSI protocols [KS05] combines a star and ring structure where each assistant shares a communication channel with the leader as well as one other assistant in a circular fashion. We refer to this as a 'wheel' topology. Another topology, which is used by Qiu et al. [Qiu+22], resembles a binary tree.

1.3.2 Building blocks for secure MPC

Partially-homomorphic encryption

A partially-homomorphic encryption (PHE) scheme is a collection of methods to securely encrypt and decrypt data, which still allows parties to manipulate the underlying data while it is encrypted. For example, the Paillier cryptosystem [Pai99] allows one to encrypt some integers $x, y \in \mathbb{Z}_{pq}$, and then homomorphically combine the two resulting ciphertexts to compute a ciphertext that encrypts the sum $x + y$. This makes the Paillier cryptosystem additively homomorphic. Another cryptosystem comes in the form of the ElGamal [Gam84] cryptosystem, which is multiplicatively homomorphic in its standard form. It works over any cyclic group $\mathbb{G}$ in which the decisional Diffie-Hellman assumption holds. By encoding values in the exponent, the cryptosystem becomes additive in some sense. Moreover, by using an elliptic curve group for $\mathbb{G}$, all operations become significantly faster than over the integers [VCE22]. MPSI protocols use threshold versions of homomorphic cryptosystems, which require $t + 1$ parties to work together to decrypt a ciphertext.

Secret sharing

Secret sharing is an information-theoretically secure method of storing data among multiple parties, without individual parties being able to access it. To do so, a secret is split up into multiple shares that combine to retrieve the original secret. One example is `XOR`-sharing, which splits a secret x into shares $s_1, \ldots, s_n$ so that $x = s_1 \oplus \cdots \oplus s_n$. Other secret sharing schemes allow for arithmetic operations to be performed over the underlying secrets by manipulating the shares. For example, a simple additive secret sharing scheme with $x = s_1 + \cdots + s_n \pmod{q}$ allows parties to compute the sum of two shared secrets, by adding their respective shares. By interacting with the other parties, it is also possible to privately multiply additive secret shares [BGW88].

Oblivious transfers

An oblivious transfer (OT) is a two-party protocol between a sender and a receiver. In its simplest form, the receiver chooses one of two messages s_0 and s_1 to receive using bit $b \in \mathbb{Z}_2$. The sender sends message s_b as requested. Crucially, the sender may not learn b and the receiver may not learn the other message s_{1-b}. Oblivious transfers are computationally cheap to evaluate in large quantities due to the

existence of OT extensions. OT extensions execute a small amount of full-fledged oblivious transfers in the opposite direction, after which the received data can be used to perform future transfers with minimal computational effort.

Oblivious linear evaluation

An oblivious transfer can be seen as a protocol that privately computes $s_b = s_0 + (s_1 - s_0)b$ for $b \in \mathbb{Z}_2$. In other words, it computes a linear function over b, which is binary. Oblivious linear evaluations (OLEs) extend oblivious transfers to compute linear functions over elements from larger groups $\mathbb{Z}_q$. More precisely, the sender holds values u and v, and the receiver learns $w = ux + v$ for some x of its choosing [Sch+19]. In some cases, a receiver might want to perform a large amount of OLEs with the same input x. There exist custom protocols for this special case called vector oblivious linear evaluations (VOLEs), which are significantly cheaper to evaluate.

Oblivious pseudorandom functions

Another primitive for evaluating functions obliviously is an oblivious pseudorandom function (OPRF). Such a protocol allows a receiver to compute a pseudorandom function over an input x of its choosing, while the sender chooses which key K is used to compute the output [CM20]. Importantly, the receiver does not learn K and the sender does not learn x, nor the output of the PRF. A multi-point OPRF evaluates multiple OPRFs at the same time on the same key K for different inputs [CM20]. It does so more efficiently than it is to execute multiple single point OPRFs. The most efficient schemes currently rely on oblivious transfers.

1.3.3 Common methods in cryptographic protocols

Finally, we explain three common techniques at the core of several MPSI protocols.

Secure `AND` operations

In many MPSI protocols there is a need to compute an `AND` operation between multiple bits $x_1, \ldots, x_n$. While a simple approach is to compute the product, the multiplications are expensive to execute privately. Instead, a typical alternative is to compute some aggregate based on the input bits that is 0 when all bits were 0, and random otherwise. Kolesnikov et al. [Kol+17], for example, achieve this using `XOR`-based secret sharing. Other works [Bay+22; VCE22] privately compute a randomized sum $\mathbf{r}(x_1 + \cdots + x_n)$, where $\mathbf{r}$ is some random number unknown to any set of colluding parties. In both cases, the result of the `AND` operation is 1 when the aggregate equals 0.

Arithmetic on encrypted polynomials

Many MPSI protocols (e.g. [FNP04; KS05; HV17]) rely on arithmetic over private polynomials. One can do so using any method that privately performs arithmetic over integers, such as secret sharing or additively homomorphic encryption.

By working over the coefficients of the polynomial, adding polynomials only requires one to add the coefficients of the two polynomials in element-wise fashion. Multiplying one private and one plaintext polynomial can be done using school-book multiplication [KS05]. Evaluating the polynomial at a plaintext value x requires computing the plaintext powers $x^2, \ldots, x^d$ up to degree d and performing a dot product. Instead of working over the coefficients, Cheon et al. [CJS12] translate the polynomials to a point-value representation, which makes polynomial multiplication computationally cheaper. We discuss this in more detail in Section 1.5.1.

Binning techniques

If an MPSI protocol does not scale linearly with the number of elements k, binning techniques can improve the over-all efficiency by splitting the problem into several small MPSI problems; one for each bin. There are two kinds of binning techniques: those where a bin can have at most one element, and those where bins can have any number of elements, but preferably as small as possible. One popular technique for assigning a bin at most one element is that of Cuckoo hashing [PR01], which populates a set of m bins by repeatedly inserting an element into a bin selected by a hash function. If that bin was occupied, the element is placed into another bin, evicting the element that was previously there and must now be re-inserted. A suitable set of parameters makes this process highly likely to terminate and succeed. If multiple elements may occupy the same bin, one does not have to consider eviction. One technique called balanced allocations [Aza+94] distributes k elements over $m = \frac{k}{\ln \ln k}$ bins by repeatedly selecting two random bins and assigning the element to the most empty bin. The number of elements in each bin is then bounded by $O(\ln \ln k)$ with overwhelming probability.

1.4 Common constructions

All MPSI schemes thus far can be expressed as a series of membership checks. This is possible because the result of a set intersection is always a subset of each party's input. Since privacy-preserving set intersections may not leak any of the intermediate computations, not all constructions are possible. We identify three remaining general constructions that are used to construct MPSI protocols.

In the first construction, the parties represent their sets so that they can be combined, and the resulting representation can be revealed to the leader. In this case, the membership queries can be done in plain text. In the second construction, the resulting representation is queried by the leader, because revealing it would leak information about the input sets. In the third construction, the leader queries each other set separately, then combines the outcomes to only reveal that an element is in the intersection if it was in *all* input sets.

Since these methods rely on ways of encoding sets into new, convenient representations, we introduce the following notation: $\hat{X} \leftarrow \mathsf{Enc}(X)$ is a representation of set X by some encoding Enc. The encoded set can be extracted using $X \leftarrow \mathsf{Dec}(\hat{X})$, but note that this function sometimes requires knowledge of a compact superset

Table 1.1: Summary of the notation in this work

Symbol	Description
n	Number of parties
t	Maximum number of colluding parties
$\mathcal{P}_i, X_i$	The ith party and its input set
k	Maximum set size
$\mathcal{U}$	Universe
$\mathcal{S}$	Simulator
view_i	View of party $\mathcal{P}_i$
$\overset{s}{\equiv}, \overset{c}{\equiv}$	Statistical and computational indistinguishability
λ	Statistical security parameter
$\{\dots\}$	Some unordered (multi)set
$[\dots]$	Some ordered vector
$\uplus$	Multiset sum
$\hat{X}, m$	Set representation of set X and its number of bins
$\mathsf{Enc}, \mathsf{Dec}$	Encodes or decodes a set in a set representation
$\odot$	Homomorphism of the set representation

of X. It is always possible to use the universe $\mathcal{U}$ for this purpose, but this is not efficient when it contains many elements. In the remainder of this section we present the currently used set representations for each general construction in the same order as above.

1.4.1 Private homomorphic set representations

Private homomorphic set representations are set representations that can be homomorphically combined using some operation $\odot$ to compute the set intersection:

$$X_1 \cap \dots \cap X_n \approx \mathsf{Dec}(\mathsf{Enc}(X_1) \odot \dots \odot \mathsf{Enc}(X_n)) \,. \tag{1.4}$$

Crucially, the resulting representation does not reveal anything about the original inputs. In other words, the result is computationally indistinguishable from the actual encoding of the intersection.

$$\mathsf{Enc}(X_1) \odot \dots \odot \mathsf{Enc}(X_n) \overset{c}{\equiv} \mathsf{Enc}(X_1 \cap \dots \cap X_n) \,. \tag{1.5}$$

Given such a private homomorphic set representation, one can design an MPSI protocol given that $\odot$ can be efficiently computed in private. We discuss four such representations.

Bitsets

Bitsets are binary vectors that represent a set X by indicating for each element of the universe $x \in \mathcal{U}$ whether $x \in X$. To do so, there must exist some ordering

over all elements in $\mathcal{U}$. Two bitsets can be homomorphically combined when their orderings agree. The homomorphism $\odot$ is then simply an element-wise `AND` operation.

Example 1. Given $\mathcal{U} = \{1, 2, 3, 4\}$ with ordering $[1, 2, 3, 4]$:

$$\begin{array}{rcl} \{1,3,4\} & \xrightarrow{\text{Enc}} & [1,0,1,1] \\ \cap\ \{1,2,3\} & \xrightarrow{\text{Enc}} & \wedge\ [1,1,1,0] \\ \hline \{1,3\} & \xleftarrow{\text{Dec}} & [1,0,1,0] \end{array}$$

There is a clear drawback to using bitsets for MPSI: the representation requires $|\mathcal{U}|$ bits to encode a set, even if it is empty. On the other hand, bitsets always exactly represent the original set, even when combined. In other words, the condition in (1.5) actually holds under equality.

Hash sets

Another example of private homomorphic set representations comes in the form of hash sets. Like a bitset, a hash set is a binary vector. It has m bits, referred to as bins, initially set to 0, and a hash function $H : \mathcal{U} \mapsto \{1, \ldots, m\}$. To encode set X, one computes the index $H(x)$ for each element $x \in X$ and sets the bin at that index to 1. If two hash sets agree on the hash function, they can also be combined homomorphically using an `AND` operation. Decoding is not straightforward since it requires computing the inverse of a hash function H^{-1}. However, given a superset S of the elements possibly contained in the hash set, one can instead check for each element $s \in S$ whether it is in the hash set.

Example 2. For some common H:

$$\begin{array}{rcl} \{1,3,4\} & \xrightarrow{\text{Enc}} & [1,1,1,0] \\ \cap\ \{1,2,3\} & \xrightarrow{\text{Enc}} & \wedge\ [0,1,1,1] \\ \hline \{1,3\} & \xleftarrow{\text{Dec}} & [0,1,1,0] \end{array}$$

Note that the encoded 1s are not in the same position as in a bitset; the positions are selected by hash function H.

Hash sets can be seen as Bloom filters with only one hash function; see Section 1.4.2. Note, however, that by increasing the number of hash functions, the representation opens up a mechanism for information leakage. Another way of looking at hash sets is as a bitset with Remark 2 applied to it. As such, (1.5) holds under equality.

The hash function H allows hash sets to require only m bits to represent a set, rather than $|\mathcal{U}|$. However, as a consequence, they do not exactly represent the original set. This occurs when H maps multiple elements to the same bin by H. If one of these elements is encoded in the hash set, all those other elements will falsely appear to be in the set. Using (1.7) with $h = 1$, such false positives can happen for each query with probability $\varepsilon \approx 1 - e^{\frac{k}{m}}$, where k is the number of elements encoded in the hash set, assuming that the output of H is statistically uniform. False negatives never occur in hash sets.

Sorted multisets

Blanton & Aguiar [BA16] propose sorted multisets to combine sets and compute their sorted intersection. The key idea is to combine multiple sets $X_1, \ldots, X_n$ into multiset $X_1 \uplus \cdots \uplus X_n$, and then isolate those elements with multiplicity n. By sorting the resulting multiset, there is a straightforward way of identifying those elements that appear n times. By also sorting the input sets, the sorted multiset can be computed without performing a full sort operation. Instead, sets can be combined using a simpler merging operation. To achieve this, Enc simply sorts the set. The homomorphism $\odot$ can then be computed by merging the sets, checking for elements that appear n times, and removing the other elements. This last step is dubbed 'monotonizing' by Poddar et al. [Pod+21]. Note that for a set to be sorted, a partial ordering must exist within $\mathcal{U}$.

A general way of merging two sorted sets is a bitonic sorter [Bat68]. Such a circuit merges two sets of total length $k = 2^p$ in p steps. After that, selecting elements that appear n times can be done by checking equality for all elements that are n places apart. In currently known protocols [BA16; Pod+21] the result then contains the elements of the intersection as well as 0s. To go back to a sorted set, those 0s can be removed using a monotonization circuit [Pod+21]. If the output must be revealed but does not have to be a sorted set without 0s, the output can be sorted [BA16] or shuffled [Pod+21] instead.

Example 3. Computing the homomorphism between two ($n = 2$) sorted multisets $[1,3,4] \leftarrow \mathsf{Enc}(\{1,3,4\})$ and $[1,2,3] \leftarrow \mathsf{Enc}(\{1,2,3\})$ goes as follows:

$$\begin{aligned}
[1,3,4] \odot [1,2,3] &= \mathsf{Mono}(\mathsf{Check}(\mathsf{Merge}([1,3,4],[1,2,3]))), \\
&= \mathsf{Mono}(\mathsf{Check}([1,1,2,3,3,4])) , \\
&= \mathsf{Mono}([1,0,0,3,0]) , \\
&= [1,3] .
\end{aligned}$$

A drawback of this representation is that the homomorphism is more complex to compute than the `AND` operations required to combine two bitsets or hash sets. However, some complexity is reduced by sorting the sets in advance. The benefit of sorted multisets is that they are both compact and exact: the representation's size is similar to the original set.

Polynomial roots

The roots of a polynomial form a multiset, so they can be used to compute intersections. To encode a set S in the roots of a polynomial in $\mathbb{Z}_q$, a mapping must exist from $\mathcal{U}$ to $\mathbb{Z}_q$. The polynomial encoding of S is:

$$\mathsf{Enc}(S) = P(x) = \prod_{s \in S} (x - s) , \tag{1.6}$$

which is a polynomial of degree $|S|$. Checking membership of a single element s is possible by evaluating the polynomial and checking if $P(s) = 0$.

Computing the polynomial that encodes the intersection of two polynomial set encodings requires computing the greatest common divisor, after which the roots

of the resulting polynomial represent the multiset intersection of the original roots. The resulting polynomial does not contain any information other than the roots in the intersection, so this is a private homomorphic set representation [KS05].

One problem with applying the previous approach to MPSI protocols is that the homomorphism requires the parties to privately compute the greatest common divisor of two polynomials. Instead, Kissner & Song [KS05] show that one can still satisfy (1.5) when computing $\hat{X}_1 \odot \hat{X}_2$ as $r_1\hat{X}_1 + r_2\hat{X}_2 = \mathbf{r}\gcd(\hat{X}_1, \hat{X}_2)$, where r_1 and r_2 are random polynomials. To achieve this, we adapt the earlier encoding function, multiplying by some random polynomial r. This randomization also helps reduce false positives, which are now spread uniformly over the range of the coefficients.

In the following example, we demonstrate that addition of two polynomials indeed retains the common roots, even when they are not randomized.

Example 4. Consider the following without randomization:

$$\begin{aligned}&\mathsf{Enc}(\{1,3,4\}) \odot \mathsf{Enc}(\{1,2,3\})\\ &= (x-1)(x-3)(x-4) + (x-1)(x-2)(x-3)\\ &= 2(x-1)(x-3)^2\ .\end{aligned}$$

Indeed, the roots are 1 and 3. Note that the root 3 appears twice, which is technically a false positive.

Polynomials are convenient set representations because they grow linearly with the size of the encoded set k, and the false positives are spread uniformly randomly over the space. A drawback is that polynomial multiplications by default scale quadratically with k.

1.4.2 Leaky homomorphic set representations

Leaky homomorphic set representations are set representations that satisfy (1.1), but for which (1.5) does not necessarily hold. In other words, the result of the homomorphism may leak information about the original sets. As such, these representations cannot be revealed to any of the parties, and instead, need to be privately queried to ensure that the parties only learn the intersection. Consequently, the representation must support an efficient way to perform private membership queries.

Bloom filters

A Bloom filter is a hash set with multiple hash functions $H_1, \ldots, H_h$, see Section 1.4.1. The benefit of using $h > 1$ is that the number of bins m can be lower for the same false positive rate ε. As a consequence however, computing the intersection between two or more Bloom filters through an `AND` operation may leak more information about the original sets than just their intersection. In other words, (1.5) does not hold.

The probability ε that a query falsely returns a positive result is approximately given by [GG10]:

$$\varepsilon \approx \left(1 - \left(1 - \frac{1}{m}\right)^{hk}\right)^h \approx \left(1 - e^{\frac{-hk}{m}}\right)^h \ . \tag{1.7}$$

So the minimal number of bins m for a Bloom filter with at most k elements and a false positive rate of ε is:

$$m = -\frac{k \ln \varepsilon}{\ln^2 2} \ , \quad h = -\log_2 \varepsilon \ . \tag{1.8}$$

The false positive rate after computing $\odot$ is bound by the false positive rate of the original Bloom filters [PSN10]. Instead of using the approximation (1.7), one can also use the upper bound by Goel & Gupta [GG10] to choose parameters as described by Vos et al. [VCE22].

The reason that Bloom filter intersections leak information is that a bin can be set to 1 even if it does not contribute to the intersection. The probability of this happening grows with the number of hash functions h. In the following example we show a situation where the filter of the actual intersection does not match the filter computed using $\odot$.

Example 5. Consider a Bloom filter with $m = 6$ and $h = 2$. The bins are indexed using $0, \ldots, m-1$. The first hash function behaves as $H_1(2) = 2$, $H_1(3) = 1$, $H_1(4) = 1$. The second as $H_2(2) = 3$, $H_2(3) = 2$, $H_2(4) = 5$.

Now, given sets $X_1 = \{2, 3\}$ and $X_2 = \{4\}$, we have:

$$\begin{aligned} \mathsf{Enc}(X_1 \cap X_2) &\neq \mathsf{Enc}(X_1) \wedge \mathsf{Enc}(X_2) \ , \\ \mathsf{Enc}(\emptyset) &\neq [0,1,1,1,0,0] \wedge [0,1,0,0,0,1] \ , \\ [0,0,0,0,0,0] &\neq [0,1,0,0,0,0] \ . \end{aligned}$$

The benefit of Bloom filters is that they scale linearly with the size of the set k. However, the drawback is that the constant factor of $O(k)$ can be large if ε must be small. For example, if $\varepsilon \approx 2^{-40}$ then $m \approx 57.7k$. In other words, it takes almost 58 bins to represent each element with negligible failure rate. A bitset requires $|\mathcal{U}|$ bins and has zero failure rate, so it stands to reason that one would then better choose a bitset representation if $|\mathcal{U}| \leq 57.7k$.

Garbled Bloom filters

In a *garbled* Bloom filter, the bins selected by the hash functions $H_1, \ldots, H_h$ are not set to 1, but to an `XOR`-sharing of some value. This makes a garbled Bloom filter a key-value store (see Section 1.4.3). The bins in such a filter are bitstrings of length λ. Garbled Bloom filters were first proposed by Dong et al. [DCW13].

Let b_i denote the ith bin of a garbled Bloom filter. Kolesnikov et al. [Kol+17] provide the following algorithm to encode a set of key-value pairs in such a filter:

1. Initialize all bins b_i for $i = 1, \ldots, m$ to $\perp$.

2. For each key-value pair (x, y), select indices $I_j = H_j(x)$ for $j = 1, \ldots, h$. For empty bins ($b_{I_j} = \perp$), choose random λ-bit strings so that $b_{I_1} \oplus \cdots \oplus b_{I_h} = y$.

3. Replace empty bins ($b_i = \perp$) with a random λ-bit string.

A simple way of using garbled Bloom filters to encode sets for performing set intersections is to encode the set elements as the keys of the filter, and set the values to $0 \ldots 0$, or some other well-formed value. The filter representing the intersection can be computed using an element-wise `XOR` operation. Only the bins pertaining to elements in the intersection then `XOR` to $0 \ldots 0$. Garbled Bloom filters inherit their *false negative* probability from the *false positive* probability Bloom filters. They also incur a chance of false positives, which occur when randomly chosen bins accidentally `XOR` to $0 \ldots 0$. This happens with probability $2^{-\lambda}$.

In the following example we show why these filters leak information through the homomorphism.

Example 6. Consider a garbled Bloom filter with $m = 5$ bins, $h = 3$ hash functions and $\lambda = 3$. The hash functions behave like $H_1(1) = 0, H_2(1) = 1, H_3(1) = 2$, and $H_1(4) = 2, H_2(4) = 4, H_3(4) = 3$. Then, the set intersection between $\{1, 4\}$ and $\{1\}$ can be computed like:

$$\begin{array}{rcl@{\quad}rcl} & \{1,4\} & \xrightarrow{\text{Enc}} & & [010, 011, 001, 010, 011] \\ \cap & \{1\} & \xrightarrow{\text{Enc}} & \oplus & [001, 100, 101, 110, 110] \\ \hline & \{1\} & \xleftarrow{\text{Dec}} & & [011, 111, 100, 100, 101] \end{array}$$

Notice that the party holding the second set can infer that the other party has 4 in its set with high probability. It can do so because the sum of the bins pertaining to 4 sum to the same value in its own garbled Bloom filter as well as the result: $101 \oplus 110 \oplus 110 = 101 = 100 \oplus 100 \oplus 101$.

The most significant benefit of garbled Bloom filters over regular Bloom filters is their ability to be used as oblivious key-value stores. That is, if the values they encode are statistically random, then it is unknown which keys are encoded into them. Moreover, the number of bins m decides the probability of false negatives rather than false positives.

1.4.3 Aggregatable membership queries

A third construction for MPSI protocols performs membership queries on the encoded sets and combines the results of these queries. As such, no homomorphism is required over the set representation. Instead, we require an aggregation method $\odot$ for which it holds that:

$$\bigwedge_{i=1}^{n} x \in X_n = \mathsf{Reveal}(\mathsf{Query}_x(\hat{X}_1) \odot \cdots \odot \mathsf{Query}_x(\hat{X}_n)) \,, \tag{1.9}$$

where $\hat{X}_i \leftarrow \mathsf{Enc}(X_i)$ conveniently encodes set X_i.

Essentially, these approaches execute two-party protocols between the leader and the assistants, and combine the results. Since this high-level construction

requires that the output of the two-party protocols remains private, all three approaches discussed here output a value linked to the queried element rather than a Boolean. For example, MPSI protocols could return a secret share of 0 for each of the elements in the queried party's set, and randomness otherwise. If so, $\odot$ is simply the reconstruction operation of the secret sharing scheme.

Embedding payloads in polynomial roots

In Section 1.4.1, we saw that polynomials encoding a set in their roots can be homomorphically combined. The same encoding can also be used to return aggregatable query results from a polynomial $p(x)$ by privately computing $rp(x)+P$, where P is the payload and x is the queried element. The result is P only when evaluated over any of the roots of p with overwhelming probability, or randomness otherwise. This method is particularly useful in protocols based on homomorphic encryption, because it allows the leader to locally evaluate the assistant's encrypted polynomial. Consequently, the leader can decide on the payload P. The leader can generate secret shares of zeroes to embedded as payloads without communicating with the assistants.

Oblivious programmable PRFs

Proposed by Kolesnikov et al. [Kol+17], an oblivious programmable PRF (OPPRF) is an oblivious PRF that contains hardcoded values in the form of key-value pairs. In other words, given a secret input x', if x' matches one of the key-value pairs (x, y) so that $x = x'$, then the output is y. Otherwise, the output is randomness. The approach in Section 1.4.3 already satisfies a form of this, where the keys are given by the roots of the polynomial and the values are decided on by the evaluator of the polynomial. OPPRFs, however, are more specific: the sender decides on the hardcoded key-value pairs. Importantly, the receiver is only allowed to perform a limited number of queries.

Kolesnikov et al. [Kol+17] propose three instantiations of OPPRFs. The first design interpolates a polynomial over the hardcoded key-value pairs, the second uses a garbled Bloom filter (see Section 1.4.2), and the third returns a key generated by a pseudo-random function that can be used to decrypt one of a set of encryptions if it pertains to one of the hardcoded keys. The authors also show how to extend the OPPRFs to be queried multiple times using cuckoo filters. The polynomial roots approach in Section 1.4.3 requires the polynomial to be evaluated in private, which is expensive, while these OPPRFs only rely on symmetric primitives.

Oblivious key-value stores

Garimella et al. [Gar+21] describe a primitive similar to OPPRFs called oblivious key-value stores (OKVS), which perform the same functionality but they can be transmitted in plain text. The polynomial-based OPPRF described above is an example of such an OKVS. In fact, this is the most compact OKVS possible, as it takes c polynomial coefficients to hard-code c key-value pairs. Crucially, the hardcoded values must be randomly distributed to hide which keys are encoded

in the data structure. While it is compact, a drawback of this polynomial-based OKVS is that encoding it is computationally expensive.

Garimella et al. [Gar+21] propose an OKVS that is significantly cheaper to encode based on cuckoo hashing. While it is not optimally compact, it is approximately 1.5 to 2.5 times larger than the polynomial-based OKVS [NTY21]. Moreover, by the way this OKVS is constructed, one can guarantee that encoding succeeds with overwhelming probability $(1 - 2^{-40})$. One of the most significant advantages of OKVSs over OPPRFs is that they reduce the communication needed for MPSI protocols in the malicious model.

This primitive originates from the garbled Bloom filters by Dong et al. [DCW13], after which they were applied to two-party protocols under the name of 'PaXoS' [Pin+20]. In parallel, the field of structured linear functions studies similar data structures. One example is the frayed ribbon filter [Ham20].

1.5 Proposed protocols

In this section, we provide a comprehensive overview of collusion-resistant MPSI protocols in the semi-honest model. We established this body of literature by exploring the references and citations of an initial set of works on MPSI protocols. In this process, we disregard works that rely on differential privacy as they do not satisfy the privacy requirements established in Section 1.2. We also do not consider server-aided protocols, which rely on a non-colluding assumption. Note that we only consider set intersection protocols, so we omit variants such as threshold-intersections [GS19] from the overview.

We summarize all protocols that do not have any known security flaws in Table 1.2. This table sorts the works by their high-level construction as discussed in the previous section and the set representation. Within these categories, the protocols are sorted chronologically. For each work, we state their communication and computation complexity, as well as the number of rounds of interaction. We provide derivations of these complexities in Appendix 1.A. We also state the network topologies, the maximum collusion resistance, and whether there also exists a maliciously-secure version of the protocol (yes ● or no ○). We do not consider whether a protocol is size-hiding, see Remark 3 in Section 1.2.2. Note that complexities using $\tilde{O}$-notation omit logarithmic terms.

Remark 4. Note that in the semi-honest model, any protocol with a mesh topology can be converted to a star topology by encrypting all messages for the intended receiver and routing them through the leader at the cost of more interactions.

Table 1.2: Overview of semi-honest MPSI protocols with no known attacks, resisting multiple colluders

Work		Technique		Communication				Computation		Security	
First author	Year	Encoding	Primitive	Topology	Leader	Assistant	Rounds	Leader	Assistant	Collusion	Malicious
Private homomorphic set representations											
Kissner [KS05]	2005	Polynomial roots	PHE	Mesh	$O(nk)$	$O(tk)$	3	$O(tk^2)$	$O(tk^2)$	$n-1$	●
Sang [San+06]	2006		PHE	Mesh	$O(nk)$	$O(nk)$	3	$O(n^2k)$	$O(n^2k)$	$n-1$	[SS07]
Li [LW07]	2007		-	Mesh	$O(ntk^2)$	$O(ntk^2)$	3	$O(ntk^2)$	$O(ntk^2)$	$\lfloor \frac{n-1}{2} \rfloor$	●
Sang [SS09]	2009		PHE*	Mesh	$O(nk)$	$O(nk)$	2	$O(nk^2)$	$O(nk^2)$	$n-1$	●
Cheon [CJS12]	2012		PHE	Mesh	$O(nk)$	$O(nk)$	3	$O(nk)$	$O(nk)$	$n-1$	●
Ghosh [GN19]	2019		OLE	Star	$O(nk)$	$O(k)$	6	$\tilde{O}(nk)$	$\tilde{O}(k)$	$n-1$	●
Gordon [GHL22]	2022		OLE	Star	$\tilde{O}(nk+nt)$	—	5	—	—	$n-1$	●
Blanton [BA16]	2016	Sorted multiset	-	Mesh	$\tilde{O}(tk)$	$\tilde{O}(tk)$	$O(\log k)$	$\tilde{O}(tk)$	$\tilde{O}(tk)$	$\lfloor \frac{n-1}{2} \rfloor$	●
Poddar [Pod+21]	2021		GC	Mesh	—	—	—	—	—	$n-1$	●
Bay [Bay+21]	2021	Bitset	PHE	Star	$O(tk)$	$O(\lvert\mathcal{U}\rvert)$	3	$O(nk)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	○
Vos [VCE22] (Chp 3)	2022										
Leaky homomorphic set representations											
Inbar [IOP18]	2018	Garbled Bloom filter	OT	Star	$O(nkh)$	$O(nkh)$	3	$O(nkh)$	$O(nkh)$	$n-1$	[Ben+22]
Kavousi [KMS21]	2021		OT	Wheel	$O(nk)$	$O(kh)$	4	$O(nk)$	$O(kh)$	$n-1$	○
Bay [Bay+22]	2022	Bloom filter	PHE	Star	$O(tk)$	$O(k)$	3	$O(nkh)$	$O(k)$	$n-1$	○
Vos [VCE22] (Chp 3)	2022										
Aggregatable membership queries											
Freedman [FNP04]	2004	Polynomial payloads	-	Mesh	$O(n^2k^2)$	$O(n^2k^2)$	4	$O(n^2k^2)$	$O(n^2k^2)$	$n-1$	○
Hazay [HV17]	2017		PHE	Star	$O(nk)$	$O(k)$	4	$\tilde{O}(nk)$	$O(k)$	$n-1$	●
Kolesnikov [Kol+17]	2017	OPPRF	OT	Mesh	$O(nk)$	$O(tk)$	4	$O(n)$	$O(tk)$	$n-1$	[Gar+21]
Chandran [Cha+21]	2021		OT	Star	$\tilde{O}(nk)$	$\tilde{O}(k)$	8	$O(nk)$	$O(k)$	$\lfloor \frac{n-1}{2} \rfloor$	○
Garimella [Gar+21]	2021	OKVS	OT	Star	$O(nk)$	$O(k)$	4	$O(k)$	$O(nk)$	$n-1$	●
Nevo [NTY21]	2021		OT	Mesh	$O(tk)$*	$O(k)$	4	$O(nk-tk)$	$O(tk)$	$n-1$	●

1.5.1 Private homomorphic set representations

Using polynomial roots

Kissner & Song [KS05] propose an MPSI protocol based on the polynomial roots representation. After encoding their set in the roots of a polynomial, each party uses an additively homomorphic cryptosystem (the Paillier cryptosystem) to encrypt the coefficients and sends them to t other parties. Each party locally randomizes the received polynomials, after which the parties sum up the result in a circular fashion, and finally decrypt. Li & Wu [LW07] propose a similar approach, except they use secret sharing. This lowers the maximum collusion threshold from $n-1$ to $\lfloor \frac{n-1}{2} \rfloor$, but the computation is significantly cheaper. Patra et al. [PCR09a; PCR09b] show that the malicious version of Li & Wu's protocol requires more operations than claimed in the original paper, and they provide alternatives with a higher maximum collusion threshold and better efficiency.

After this, Sang et al. [San+06] discuss several steps in optimizing the randomized aggregation operation necessary to compute the intersection in the semi-honest model. In [SS07], they propose a malicious extension of this protocol. The protocols use degree 1 polynomials when randomizing to save computations, and provide proof that this is indeed still resistant against collusion attacks given that the coefficients are multiplied by a non-singular matrix. In later works, Sang & Shen [SS08; SS09] describe a second protocol based on bilinear pairings. However, the pairings are only necessary in the malicious model. We note that it suffices to use elliptic curve-based ElGamal in the semi-honest model, for example. The idea behind this protocol is that it suffices to randomize each polynomial once using a single scalar instead of $t+1$ times by different parties if we only reveal the evaluation of the resulting polynomial. Importantly, this is only secure for cryptosystems where parties cannot determine the discrete log of the final decryption. For this reason, we mark PHE with an asterisk in Table 1.2.

Instead of eliminating polynomial multiplications, some other works focus on lowering their computational cost. Cheon et al. [CJS12], for example, propose a protocol based on that of Kissner & Song [KS05] where polynomials are in point-value representation: instead of encrypting a polynomial by its coefficients, the parties first evaluate the polynomial locally on a set of public points, and encrypt the resulting values. Multiplication of polynomials in this representation scales linearly with the degree rather than quadratically. However, one must ensure there are at least as many encrypted values as the degree of the resulting polynomial. Focusing on the two-party setting, Kim et al. [KK17] propose several other techniques to perform computationally cheaper polynomial multiplications and evaluations.

Ghosh & Nilges [GN19] use the same point-value representation as above, but they show how to compute the randomized sum without the need for homomorphic encryption. Instead, they propose to have every assistant perform a series of OLEs with the leader. In the setting with two parties $\mathcal{P}_1$ and $\mathcal{P}_2$ where only $\mathcal{P}_1$ receives the result, the parties compute $r\hat{X}_1 + \hat{X}_2$ using an OLE, where r is chosen by $\mathcal{P}_2$. In the multi-party setting, the authors propose to use an OLE between each assistant and the leader in both directions, to achieve a secret sharing of the resulting polynomial. By clever use of masking and a non-interactive secret sharing scheme,

the authors make sure that the leader must aggregate all randomized polynomials before being able to reveal the intersection. Note that recently, Abadi et al. [AMZ21] identified multiple attacks against the version of the protocol that was claimed to withstand malicious adversaries. The authors compare the bandwidth cost of their protocol with that by Kolesnikov et al. [Kol+17]. For $k = 2^{20}$ and $t = n - 1$, the bandwidth per party is ≈ 1.25 GB for the protocol by Ghosh & Nilges, and $(n-1) \cdot 467$ MB for the previous work.

The latest work in this category comes from Gordon et al. [GHL22], who propose an MPSI protocol secure in the malicious model. They also propose a version where all parties receive the result. The protocols use OLEs to compute the randomized polynomial sum:

$$\hat{X}_\cap = \left(\sum_{i=1}^{n} r_i\right) \hat{X}_1 + \sum_{i=2}^{n} (s_1^i + s_i)\hat{X}_i \ . \tag{1.10}$$

Here, r_i and s_i are random polynomials selected by each party, and s_1^i are $n-1$ polynomials sampled by the leader. For each set element, each assistant performs four OLEs with the leader. For $k = 2^{20}$ and $t = n - 1$, the malicious protocol by Gordon et al. consistently outperforms Kolesnikov et al. in the semi-honest model. Care must be taken to interpret these numbers as the experiments were run on different hardware (3.60GHz CPUs versus 2.30GHz).

Using sorted multisets

Huang et al. [HEK12] use sorted multisets to perform a private set intersection between two parties, and Poddar et al. [Pod+21] provide an extension to any number of parties. Both works use garbled circuits (GC) to privately compute the intersection between sorted multisets. While garbled circuits by default offer a way to privately perform a two-party computation, Poddar et al. use the method by Wang et al. [WRK17], which first uses an expensive offline phase to build a garbled circuit that can be used for secure multi-party computation. Since the MPSI operation is part of a large compiler pipeline, it is hard to state any asymptotic complexities for the resulting protocol.

Before the work by Poddar et al., Blanton et al. [BA16] used the same representation to compute intersections and other set and multiset operations using secret sharing. The benefit of this method is that computation is extremely efficient, but the protocol does not run in a constant number of rounds due to the merging operations: it scales logarithmically with the size of the sets k. For $n = 3$, $t = 1$, $k = 2048$ and with a 1Gb Ethernet connection, the protocol takes 25 seconds.

Using bitsets

Ruan et al. [Rua+19] proposed the first bitset-based private set intersection protocol. Their protocol is restricted to two parties, so Bay et al. [Bay+21] provide an extension to the multi-party setting, achieving up to $n-1$ collusion resistance. The protocol uses additively homomorphic encryption to compute an element-wise `AND` operation between all bitsets using the method described in Section 1.3.3. Vos

et al. [VCE22] (see Chapter 3) show how to perform this operation in a way that is significantly cheaper. By using ElGamal over an elliptic curve group, operations are less computationally demanding, and the ciphertext size is restricted to 64 bytes. Another improvement comes from reordering the randomization, so that the leader has to perform fewer plaintext multiplications. Without considering communication, these optimizations reduce run time from 100 seconds to 3 seconds for $k = 2^{12}$, $n = 5$ and $t = 4$.

1.5.2 Leaky homomorphic set representations

Using garbled Bloom filters

The first protocol based on garbled Bloom filters was a two-party private set intersection by Dong et al. [DCW13]. Inbar et al. [IOP18] provided three protocols that extend this protocol to the multi-party setting. The first protocol is in the server-aided model, relying on a non-colluding third party, while the next two protocols are collusion-resistant and secure in the semi-honest and augmented semi-honest model, respectively. The latter two protocols rely on oblivious transfers so each party only learns the contents of the garbled Bloom filter bins that they should learn. The augmented semi-honest model allows the parties to perform fewer OT extensions while realizing the same functionality. While OTs are cheap to compute, the protocols require all pairs of parties to perform them, causing the communication to scale quadratically with the number of parties. Recently, Ben-Efraim et al. [Ben+22] provided an extension of this work to the malicious model. They also show how to reduce the number of OTs by 25% by choosing other parameters for the garbled Bloom filter. Their results for $t = n - 1$ show that regardless of k and n, Kolesnikov et al. and Ben-Efraim et al. consistently outperform the work by Inbar et al. Moverover, for $n \geq 10$ and $k \geq 2^{16}$ the malicious protocol by Ben-Efraim et al. consistently outperforms the semi-honest protocol by Kolesnikov et al.

A protocol by Kavousi et al. [KMS21] uses garbled Bloom filters differently, extending the two-party work of Chase & Miao [CM20], who propose a highly efficient OPRF that can be queried in multiple points at once using OTs. Bay & Kayan [BK22] also propose a multi-party extension, but without a garbled Bloom filter and therefore require the asisstants to send the PRF of the hash of each element to the leader. We argue that this renders the protocol insecure, as the leader can enumerate the universe $\mathcal{U}$ to identify preimages.

Using Bloom filters

Bay et al. [Bay+22] propose a collusion-resistant version of the protocol by Miyaji & Nishida [MN15]. Their protocol uses the Paillier cryptosystem to compute the `AND` operation between each party's encrypted Bloom filter following the technique described in Section 1.3.3. Vos et al. [VCE22] provide a similar improvement as described in Section 1.5.1 to significantly reduce the computation and communication cost of the protocol, although it does not impact the asymptotic complexities. These protocols satisfy superset correctness: they only incur false positives with

non-negligible probability. Vos et al. show that for $\varepsilon \geq 0.01\%$, $t = n - 1$ and $n \geq 3$, the protocol consistently outperforms Kolesnikov et al. [Kol+17].

1.5.3 Aggregatable membership queries

Embedding payloads in polynomial roots

Freedman [FNP04] proposed the first MPSI protocol. This secret sharing-based protocol uses the polynomial set encoding to embed payloads in shares containing the evaluation of the polynomial in the elements of the leader's set. This payload is a secret share, so when all polynomials contain a root at this element, the secret can be reconstructed. Hazay & Venkitasubramaniam propose another protocol using PHE, which allows the leader to evaluate the polynomials, instead of the assistants. The leader only inserts payloads of 0. After randomizing and decrypting (reminiscent to Section 1.3.3), the leader receives a result of 0 when the element is in the intersection with overwhelming probability.

Using oblivious programmable PRFs

Kolesnikov et al. [Kol+17] introduce OPPRFs and show how they can be used to instantiate an MPSI protocol. The first part of the protocol is expensive, establishing an `XOR`-sharing of zero for each element that could be in the intersection. After that, the parties hardcode those values in their OPPRF for each element in their set. The leader then queries all assistants' OPPRFs and reconstructs the secret. Chandran et al. [Cha+21] create three modifications of the protocol by Kolesnikov et al., removing the expensive construction of secret shares that `XOR` to zero, replacing it with Shamir's secret sharing scheme. As a consequence, collusion resistance is halved. In their results, this leads to a 1.2 to 6.2 times speedup over Kolesnikov et al. in different settings. For $n = 15$, $t = 7$, and $k = 2^{20}$, protocol C took 244.8 seconds to complete over WAN, while Kolesnikov et al. took 1524.5 seconds.

Using oblivious key-value stores

Garimella et al. [Gar+21] propose OKVSs as a way of reducing the communication cost required in OPPRF-based protocols. The authors show how to use VOLEs to optimize previous protocols, and they instantiate them with oblivious transfers. They show that in slow networks, this approach leads to faster protocol executions. Nevo et al. [NTY21] lower both the computational and communication cost compared to previous protocols. They do so by cleverly using a set of 'pivot' parties that hold the OKVSs, reducing the overall cost. At the same time these pivot parties directly decide the protocols resistance to collusions. For this reason, when $t = n - 1$, the protocol converges with that of Garimella et al. [Gar+21]. In Table 1.2 we mark the leader's communication complexity with an asterisk: When $t \leq \frac{n}{2}$, the leader's communication is instead given by $O(nk - tk)$. Qiu et al. [Qiu+22] consider the protocol by Garimella et al. in a tree-shaped network topology and provide a maliciously-secure protocol. This lowers the computational cost while changing the number of rounds to $2\lfloor \log_2(n+1) \rfloor$. Nevo et al. consistently outperform the works by Kolesnikov et al., Ben-Efraim et al., and Chandran et al. in terms of

required bandwidth and run time. An exception is when $t = n - 1$, the run time is then the same as that of Kolesnikov et al. The authors report that when $n = 15$, $t = 7$, and $k = 2^{20}$, the protocol takes 3 minutes to execute and 1416.9 MB of bandwidth.

1.6 Common pitfalls

In this section, we put forward several common pitfalls in designing MPSI protocols and analyze the ways that these vulnerabilities occur.

1.6.1 Leakage from set representations

As we established in Section 1.4, there are two classes of homomorphic set representations: those that remain private when revealed and those that leak information about the inputs. While it is not always obvious how leaky set representations expose private data, the leakage is not negligible. A common mistake is that a Bloom filter representing the intersection is revealed to some of the parties. This happens for example in the work by Many et al. [MBD12]. Another work by Lai et al. [Lai+06] even uses Bloom filters in plain text, but this allows an attacker to brute force all possible elements encoded within it. In other cases, the protocol involuntarily leaks information about the Bloom filter or its queries. For example, the protocols by Debnath et al. [Deb+21b; Deb+21a] leak the sum of the queried bins, which reveals information about how many parties have a given element in their sets.

1.6.2 Unsafe randomness during aggregation

As mentioned in Section 1.3.3, many MPSI protocols rely on privately computing an `AND` operation. For bits $x_1, \ldots, x_n$, one way is to compute $\mathbf{r}(x_1 + \cdots + x_n)$, where $\mathbf{r}$ is some randomness unknown to any set of colluding parties.

One might think that the requirement above is met when we let the parties compute $r_1(1 - x_1) + \cdots + r_n(1 - x_n)$, where r_i is some randomness only known to party $\mathcal{P}_i$. However, whether parties know $\mathbf{r}$ now depends on the inputs. Notice, for example, that when $x_1 = 0$ and $x_2, \ldots, x_n = 1$, $\mathbf{r} = r_1$, which is known to party $\mathcal{P}_1$ and anyone they collude with. This allows a set of colluding parties to tell if they are the only parties with an input bit of 1.

A related error was made in the protocol by Wei et al. [Wei+22], which privately computes:

$$y = (s_1 x_1 + r'_1(1 - x_1)) + \cdots + (s_n x_n + r'_n(1 - x_n)) \,, \tag{1.11}$$

where s_i is a secret share of 0 encoded in the exponent of a generator, so it holds that $s_1 + \cdots + s_n = 0$. A share is privately constructed as follows: $s_i = \sum_{j=1}^{i-1} r_j - \sum_{j=i+1}^{n} r_j$, where $r_1, \ldots, r_n$ is randomness generated by parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, respectively. Importantly, a party does not know its share until (1.11) is computed. At first sight, this seems to solve the earlier issue. However, since party $\mathcal{P}_j$ knows the randomness r_j that they contributed, they can remove their contribution from

other parties shares $s_{j'}$ for $j' \neq j$. This allows a party to tell that it is the only party with an input bit of 1.

Another mistake is to only let one party contribute randomness. Doing so reduces the maximum collusion resistance to 1. This mistake was made in the protocol by Miyaji & Nishida [MN15], and was patched by Bay et al. [Bay+22]. In this protocol, only the leader $\mathcal{P}_1$ randomized the sum. In other words, the parties compute $r_1(x_1 + \cdots + x_n)$. As a result, the leader can invert this randomization at the end of the protocol to extract the number of input bits set to 1.

1.6.3 Adapting to the malicious model

While this paper focuses on the semi-honest model, one should note that translating protocols from this model to the malicious model comes with its own set of challenges. For example, the polynomial set representation as described in Section 1.4.1 is susceptible to manipulation by malicious adversaries [AMZ21]. While the first two attacks discussed by Abadi et al. are specific to the protocol of Gordon et al. [GN19], the third attack highlights a problem inherent to the point-value representation. Specifically, it allows an adversary to omit elements from the intersection by multiplying each element of the point-value representation by a scalar.

Qiu et al. [Qiu+22] propose a different type of attack against the protocol by Garimella et al. [Gar+21] for returning the output of the MPSI to all parties. The attack allows an adversary to learn information about the elements held by honest parties. The attack does not affect the security of the OKVS.

1.7 Analytical evaluation of computational costs

The performance of MPSI protocols is affected by many parameters, including: its hardness assumptions, the security parameter, the network topology, collusion threshold t, the number of interactions, the number of parties n, the number of elements k, the error probability ε, latency, throughput, the distribution of computer power over different parties, and the number of available threads. As a result, any concrete comparison that makes assumptions about any of these parameters unfairly puts certain schemes at an advantage (in fact, almost any scheme can win in a specific setting), and without established public uses of MPSI protocols, it remains unclear what realistic sets of parameters look like.

Instead, we study the theoretical computational cost of MPSI protocols. We consider PHE-based protocols separately from protocols based on aggregatable membership queries, which rely on two-party computations (2PCs). For brevity, we denote $E = \ln(\varepsilon^{-1})$. At the end of this section we nevertheless try to describe what reasonable parameter sets might look like, and discuss which protocols suit them.

1.7.1 Efficiency of PHE-based protocols

We instantiate the latest version of each PHE-based protocol discussed in Section 1.5 using elliptic curve-based ElGamal, as used by Vos et al. [VCE22]. Next, we analyze

Table 1.3: Number of primitive operations in the worst case

	Leader	Assistant	Total
Elliptic curve multiplications per element			
Sang	$1.5 + 2n + 2k$	1.5	$4n + 2k + t + 1$
Cheon	$4n + 3$	$4n + 3$	$5n + 4nt + 2t + 2$
Hazay	$10n - 7$	3.5	$12n - 6 + 3t$
Vos	3	$1.04E + 3$	$1.04(n-1)E + 3t + 3$
Oblivious transfers			
Inbar	$2.08(n-1)Ek$	$2.08Ek$	$2.08(n-1)Ek$
Kolesnikov	$7(n-1)k$	$3.5(n-1)k$	$3.5(n+1)(n-1)k$
Chandran	$8.96(n-1)k$	$8.96k$	$8.96(n-1)k$
Kavousi	$588(n-1)$	588	$588(n-1)\lambda$
Garimella	—	—	—
Nevo	$3.5tk$	$3.5k$	$3.5tk$
Oblivious linear evaluations			
Ghosh	$2(n-1)k$	$2k$	$2(n-1)k$
Gordon	$(n-1)k$	k	$(n-1)k$

the computational cost of these protocols by counting the number of elliptic curve multiplications required to evaluate them. Since it is possible for one curve point to precompute a basepoint table, we distinguish between regular and precomputed multiplications: precomputed multiplications are approximately four times cheaper. We report the results in Table 1.3. Note that these equations differ from the asymptotic complexities, because they do not consider homomorphic addition, which is negligible compared to multiplications.

Based on this, it depends on the use case which protocol is more computationally efficient. For example, if the assistants have low computational power, the protocol by Hazay et al. [HV17] is faster. Sang et al. [SS09] is also cheap for an assistant, but scales quadratically with k for the leader. If there is no difference in computational power and t is small, Cheon et al. [CJS12] is faster. If t is close to n and ε can be high, then Vos et al. [VCE22] may be cheaper.

1.7.2 Efficiency of 2PC-based protocols

In Table 1.3, we state the number of calls to the sub-protocols of the 2PC-based protocols from Section 1.5. Where possible we analyzed the protocols in the augmented semi-honest model. We ignore setup operations, such as the initial OT phase. It is important to note that we expressed these in terms of the sub-protocols used in the respective papers. However, one might also instantiate the protocol by Kolesnikov et al. [Kol+17], for example, with a VOLE-based OPPRF. Also note that Kavousi et al. [KMS21] requires the parties to take part in a constant number of OTs, regardless of k. Other parts of the protocol do scale with k.

1.7.3 Analysis towards potential use cases

Use case	Setting	n	t	k	ε
(1) Mutual customers	Mesh	≤ 5	$n-1$	$\leq 2^{20}$	5%
(2) Common IP addresses	Star	≤ 50	$\frac{n}{2}$	$\sim 2^{10}$	1%
(3) Consensus voting	Star	≥ 100	$\frac{n}{10}$	$\leq 2^{8}$	2^{-40}

We briefly analyze which protocols would be suitable for three highlighted applications. We present potential parameters in the table above. Application (1) involves some companies that want to identify mutual customers for an ad campaign, (2) involves cyber security organizations that want to identify suspicious IP addresses for investigation, and (3) involves a consensus vote between many parties. The companies in (1) run few multi-threaded servers that are always online. A good fit seems to be the work by Nevo et al. [NTY21], which outperforms other 2PC approaches, supports $t = n-1$, and is fast for large sets with few parties. With $n \leq 50$, (2) requires a lowly-interactive protocol in the star topology like Ghosh, Gordon, Chandran, Vos, Inbar, Hazay, or Garimella. For $\varepsilon = 1\%$ and $n = 50$, Vos only takes 313 multiplications compared to Hazay's 669. The same applies to (3), but it does not permit approximations. Depending on the network, one might choose Ghosh, Gordon, Chandran, or Hazay, as assistants' computation does not scale with n.

1.8 Discussion

Based on Sections 1.5 and 1.7 we observe that older works are still relevant today. Specifically, when instantiating PHE-based protocols with modern cryptosystems such as elliptic curve-based ElGamal, their performance becomes competitive with the latest proposals. Moreover, while it may be convenient to designate one protocol as the state of the art, this is not possible for MPSI protocols: it is possible for each protocol to find application settings where they require fewer calls to sub-protocols or fewer EC operations than other works. This also begs the question what real-life application settings look like: what are common parameters for n, t, and k, and what are the network settings like? Until then, direct comparisons cannot be considered conclusive.

When looking at how MPSI protocols come to be, the process sometimes starts with a two-party protocol, which is extended to a semi-honest protocol, and finally to a version that is maliciously secure. We see a trend where more recent works only propose a multi-party maliciously-secure protocol, skipping the semi-honest model where it might be explained more plainly and nuances are better understood.

When it comes to technical advances, we identify improvements in OKVSs and homomorphic cryptosystems as the most impactful. OKVSs can be optimized in isolation of the protocols that use them so they do not require significant knowledge of cryptography, and more efficient versions would immediately increase performance of the latest 2PC-based protocols. So far, we are not aware

of any collusion-resistant MPSI protocols that use lattice-based homomorphic encryption, or which use leveled/fully-homomorphic encryption to outperform PHE-based protocols.

Finally, when comparing PHE-based MPSI protocols to 2PC-based protocols, we note that the first are easier to analyze for different application settings. This complexity in analyzing 2PC-based protocols comes from having to choose cryptographic parameters, which do not correlate in a straightforward manner with k, for example. Moreover, multiple sets of cryptographic parameters are suitable for each instance. At the same time, it is this flexibility that also allows the user of a protocol to tune between computation and communication, a feature that is not typical of PHE-based protocols. Future research could elaborate on how to choose these parameters, easing the analysis and deployment of 2PC-based protocols.

1.9 Conclusion

This work provides a systematization of collusion-resistant MPSI protocols, focusing on the semi-honest model. We describe the formal requirements that MPSI protocols must satisfy, and present high-level constructions that describe all published MPSI protocols. Next to that, we provide a comprehensive overview of collusion-resistant MPSI protocols and broken protocols, as well as an analytical evaluation of their computational cost. This evaluation shows that there is no such thing as a single state of the art, but rather that each protocol outperforms the others depending on the application setting. We highlight several future research directions, intending to bring the performance of MPSI protocols closer to that of two-party PSIs.

References

[AMZ21] Aydin Abadi, Steven J. Murdoch, and Thomas Zacharias. "Polynomial Representation Is Tricky: Maliciously Secure Private Set Intersection Revisited". In: *Computer Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II*. Ed. by Elisa Bertino, Haya Shulman, and Michael Waidner. Vol. 12973. Lecture Notes in Computer Science. Springer, 2021, pp. 721–742. DOI: 10.1007/978-3-030-88428-4_35. URL: https://doi.org/10.1007/978-3-030-88428-4%5C_35.

[Aza+94] Yossi Azar et al. "Balanced allocations (extended abstract)". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*. Ed. by Frank Thomson Leighton and Michael T. Goodrich. ACM, 1994, pp. 593–602. DOI: 10.1145/195058.195412. URL: https://doi.org/10.1145/195058.195412.

[BA16] Marina Blanton and Everaldo Aguiar. “Private and oblivious set and multiset operations”. In: *Int. J. Inf. Sec.* 15.5 (2016), pp. 493–518. DOI: 10.1007/s10207-015-0301-1. URL: https://doi.org/10.1007/s10207-015-0301-1.

[Bat68] Kenneth E. Batcher. “Sorting Networks and Their Applications”. In: *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*. Vol. 32. AFIPS Conference Proceedings. Thomson Book Company, Washington D.C., 1968, pp. 307–314. DOI: 10.1145/1468075.1468121. URL: https://doi.org/10.1145/1468075.1468121.

[Bay+21] Aslí Bay et al. “Multi-Party Private Set Intersection Protocols for Practical Applications”. In: *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*. Ed. by Sabrina De Capitani di Vimercati and Pierangela Samarati. SCITEPRESS, 2021, pp. 515–522. DOI: 10.5220/0010547605150522. URL: https://doi.org/10.5220/0010547605150522.

[Bay+22] Aslí Bay et al. “Practical Multi-Party Private Set Intersection Protocols”. In: *IEEE Trans. Inf. Forensics Secur.* 17 (2022), pp. 1–15. DOI: 10.1109/TIFS.2021.3118879. URL: https://doi.org/10.1109/TIFS.2021.3118879.

[Ben+22] Aner Ben-Efraim et al. “PSImple: Practical Multiparty Maliciously-Secure Private Set Intersection”. In: *ASIA CCS ’22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*. Ed. by Yuji Suga et al. ACM, 2022, pp. 1098–1112. DOI: 10.1145/3488932.3523254. URL: https://doi.org/10.1145/3488932.3523254.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)”. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by Janos Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213. URL: https://doi.org/10.1145/62212.62213.

[BK22] Aslı Bay and Anıl Kayan. “A new multi-party private set intersection protocol based on OPRFs”. In: *Mugla Journal of Science and Technology* 8.1 (2022), pp. 69–75.

[Cha+21] Nishanth Chandran et al. “Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI”. In: *CCS ’21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim et al. ACM, 2021, pp. 1182–1204. DOI: 10.1145/3460120.3484591. URL: https://doi.org/10.1145/3460120.3484591.

[CJS12] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. "Multi-Party Privacy-Preserving Set Intersection with Quasi-Linear Complexity". In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 95-A.8 (2012), pp. 1366–1378. DOI: 10.1587/transfun.E95.A.1366. URL: https://doi.org/10.1587/transfun.E95.A.1366.

[CM20] Melissa Chase and Peihan Miao. "Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF". In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. Lecture Notes in Computer Science. Springer, 2020, pp. 34–63. DOI: 10.1007/978-3-030-56877-1_2. URL: https://doi.org/10.1007/978-3-030-56877-1%5C_2.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 789–800. DOI: 10.1145/2508859.2516701. URL: https://doi.org/10.1145/2508859.2516701.

[Deb+21a] Sumit Kumar Debnath et al. "Post-quantum secure multi-party private set-intersection in star network topology". In: *J. Inf. Secur. Appl.* 58 (2021), p. 102731. DOI: 10.1016/j.jisa.2020.102731. URL: https://doi.org/10.1016/j.jisa.2020.102731.

[Deb+21b] Sumit Kumar Debnath et al. "Secure and efficient multiparty private set intersection cardinality". In: *Adv. Math. Commun.* 15.2 (2021), pp. 365–386. DOI: 10.3934/amc.2020071. URL: https://doi.org/10.3934/amc.2020071.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. "Efficient Private Matching and Set Intersection". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 1–19. DOI: 10.1007/978-3-540-24676-3_1. URL: https://doi.org/10.1007/978-3-540-24676-3%5C_1.

[Gam84] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2. URL: https://doi.org/10.1007/3-540-39568-7%5C_2.

[Gar+21] Gayathri Garimella et al. “Oblivious Key-Value Stores and Amplification for Private Set Intersection”. In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. Lecture Notes in Computer Science. Springer, 2021, pp. 395–425. DOI: 10.1007/978-3-030-84245-1_14. URL: https://doi.org/10.1007/978-3-030-84245-1%5C_14.

[GG10] Ashish Goel and Pankaj Gupta. “Small subset queries and bloom filters using ternary associative memories, with applications”. In: *SIGMETRICS 2010, Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, New York, USA, 14-18 June 2010*. Ed. by Vishal Misra, Paul Barford, and Mark S. Squillante. ACM, 2010, pp. 143–154. DOI: 10.1145/1811039.1811056.

[GHL22] S. Dov Gordon, Carmit Hazay, and Phi Hung Le. “Fully Secure PSI via MPC-in-the-Head”. In: *Proc. Priv. Enhancing Technol.* 2022.3 (2022), pp. 291–313. DOI: 10.56553/popets-2022-0073. URL: https://doi.org/10.56553/popets-2022-0073.

[GN19] Satrajit Ghosh and Tobias Nilges. “An Algebraic Approach to Maliciously Secure Private Set Intersection”. In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 154–185. DOI: 10.1007/978-3-030-17659-4_6. URL: https://doi.org/10.1007/978-3-030-17659-4%5C_6.

[GS19] Satrajit Ghosh and Mark Simkin. “The Communication Complexity of Threshold Private Set Intersection”. In: *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. Lecture Notes in Computer Science. Springer, 2019, pp. 3–29. DOI: 10.1007/978-3-030-26951-7_1. URL: https://doi.org/10.1007/978-3-030-26951-7%5C_1.

[Ham20] Mike Hamburg. *Compressed maps without the keys, based on frayed ribbon cascades*. 2020. URL: https://github.com/bitwiseshiftleft/compressed%5C_map.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. “Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?” In: *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012. URL: https://www.ndss-symposium.org/ndss2012/private-set-intersection-are-garbled-circuits-better-custom-protocols.

[HL10] Carmit Hazay and Yehuda Lindell. "A Note on the Relation between the Definitions of Security for Semi-Honest and Malicious Adversaries". In: *IACR Cryptol. ePrint Arch.* (2010), p. 551. URL: http://eprint.iacr.org/2010/551.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. "Scalable Multi-party Private Set-Intersection". In: *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*. Ed. by Serge Fehr. Vol. 10174. Lecture Notes in Computer Science. Springer, 2017, pp. 175–203. DOI: 10.1007/978-3-662-54365-8_8. URL: https://doi.org/10.1007/978-3-662-54365-8%5C_8.

[IOP18] Roi Inbar, Eran Omri, and Benny Pinkas. "Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters". In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. Ed. by Dario Catalano and Roberto De Prisco. Vol. 11035. Lecture Notes in Computer Science. Springer, 2018, pp. 235–252. DOI: 10.1007/978-3-319-98113-0_13. URL: https://doi.org/10.1007/978-3-319-98113-0%5C_13.

[KBM22] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. "Faster Secure Comparisons with Offline Phase for Efficient Private Set Intersection". In: *CoRR* abs/2209.13913 (2022). DOI: 10.48550/arXiv.2209.13913. arXiv: 2209.13913. URL: https://doi.org/10.48550/arXiv.2209.13913.

[KK17] Myungsun Kim and Benjamin Z. Kim. "An experimental study of encrypted polynomial arithmetics for private set operations". In: *J. Commun. Networks* 19.5 (2017), pp. 431–441. DOI: 10.1109/JCN.2017.000075. URL: https://doi.org/10.1109/JCN.2017.000075.

[KMS21] Alireza Kavousi, Javad Mohajeri, and Mahmoud Salmasizadeh. "Efficient Scalable Multi-party Private Set Intersection Using Oblivious PRF". In: *Security and Trust Management - 17th International Workshop, STM 2021, Darmstadt, Germany, October 8, 2021, Proceedings*. Ed. by Rodrigo Roman and Jianying Zhou. Vol. 13075. Lecture Notes in Computer Science. Springer, 2021, pp. 81–99. DOI: 10.1007/978-3-030-91859-0_5. URL: https://doi.org/10.1007/978-3-030-91859-0%5C_5.

[Kol+16] Vladimir Kolesnikov et al. "Efficient Batched Oblivious PRF with Applications to Private Set Intersection". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 818–829. DOI: 10.1145/2976749.2978381. URL: https://doi.org/10.1145/2976749.2978381.

[Kol+17] Vladimir Kolesnikov et al. “Practical Multi-party Private Set Intersection from Symmetric-Key Techniques”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 1257–1272. DOI: 10.1145/3133956.3134065. URL: https://doi.org/10.1145/3133956.3134065.

[KS05] Lea Kissner and Dawn Xiaodong Song. “Privacy-Preserving Set Operations”. In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 241–257. DOI: 10.1007/11535218_15. URL: https://doi.org/10.1007/11535218%5C_15.

[Lai+06] Pierre K. Y. Lai et al. “An Efficient Bloom Filter Based Solution for Multiparty Private Matching”. In: *Proceedings of the 2006 International Conference on Security & Management, SAM 2006, Las Vegas, Nevada, USA, June 26-29, 2006*. Ed. by Hamid R. Arabnia and Selim Aissi. CSREA Press, 2006, pp. 286–292.

[Lin17] Yehuda Lindell. “How to Simulate It - A Tutorial on the Simulation Proof Technique”. In: *Tutorials on the Foundations of Cryptography*. Ed. by Yehuda Lindell. Springer International Publishing, 2017, pp. 277–346. DOI: 10.1007/978-3-319-57048-8_6. URL: https://doi.org/10.1007/978-3-319-57048-8%5C_6.

[LW07] Ronghua Li and Chuankun Wu. “An Unconditionally Secure Protocol for Multi-Party Set Intersection”. In: *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. Lecture Notes in Computer Science. Springer, 2007, pp. 226–236. DOI: 10.1007/978-3-540-72738-5_15. URL: https://doi.org/10.1007/978-3-540-72738-5%5C_15.

[MBD12] Dilip Many, Martin Burkhart, and Xenofontas Dimitropoulos. “Fast private set operations with sepia”. In: *ETZ G93* (2012).

[MN15] Atsuko Miyaji and Shohei Nishida. “A Scalable Multiparty Private Set Intersection”. In: *Network and System Security - 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings*. Ed. by Meikang Qiu et al. Vol. 9408. Lecture Notes in Computer Science. Springer, 2015, pp. 376–385. DOI: 10.1007/978-3-319-25645-0_26. URL: https://doi.org/10.1007/978-3-319-25645-0%5C_26.

[NTY21] Ofri Nevo, Ni Trieu, and Avishay Yanai. “Simple, Fast Malicious Multiparty Private Set Intersection”. In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim et al. ACM, 2021, pp. 1151–1165. DOI: 10.1145/3460120.3484772. URL: https://doi.org/10.1145/3460120.3484772.

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16. URL: https://doi.org/10.1007/3-540-48910-X%5C_16.

[PCR09a] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. "Information Theoretically Secure Multi Party Set Intersection Re-visited". In: *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*. Ed. by Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini. Vol. 5867. Lecture Notes in Computer Science. Springer, 2009, pp. 71–91. DOI: 10.1007/978-3-642-05445-7_5. URL: https://doi.org/10.1007/978-3-642-05445-7%5C_5.

[PCR09b] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. "Round Efficient Unconditionally Secure MPC and Multiparty Set Intersection with Optimal Resilience". In: *Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings*. Ed. by Bimal K. Roy and Nicolas Sendrier. Vol. 5922. Lecture Notes in Computer Science. Springer, 2009, pp. 398–417. DOI: 10.1007/978-3-642-10628-6_26. URL: https://doi.org/10.1007/978-3-642-10628-6%5C_26.

[Pin+20] Benny Pinkas et al. "PSI from PaXoS: Fast, Malicious Private Set Intersection". In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 739–767. DOI: 10.1007/978-3-030-45724-2_25. URL: https://doi.org/10.1007/978-3-030-45724-2%5C_25.

[Pod+21] Rishabh Poddar et al. "Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics". In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 2129–2146. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/poddar.

[PR01] Rasmus Pagh and Flemming Friche Rodler. "Cuckoo Hashing". In: *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*. Ed. by Friedhelm Meyer auf der Heide. Vol. 2161. Lecture Notes in Computer Science. Springer, 2001, pp. 121–133. DOI: 10.1007/3-540-44676-1_10. URL: https://doi.org/10.1007/3-540-44676-1%5C_10.

[PSN10] Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl. "Cardinality estimation and dynamic length adaptation for Bloom filters". In: *Distributed Parallel Databases* 28.2-3 (2010), pp. 119–156. DOI: 10.1007/s10619-010-7067-2. URL: https://doi.org/10.1007/s10619-010-7067-2.

[Qiu+22] Zhi Qiu et al. "Maliciously Secure Multi-party PSI with Lower Bandwidth and Faster Computation". In: *Information and Communications Security - 24th International Conference, ICICS 2022, Canterbury, UK, September 5-8, 2022, Proceedings*. Ed. by Cristina Alcaraz et al. Vol. 13407. Lecture Notes in Computer Science. Springer, 2022, pp. 69–88. DOI: 10.1007/978-3-031-15777-6_5. URL: https://doi.org/10.1007/978-3-031-15777-6%5C_5.

[Rua+19] Ou Ruan et al. "New Approach to Set Representation and Practical Private Set-Intersection Protocols". In: *IEEE Access* 7 (2019), pp. 64897–64906. DOI: 10.1109/ACCESS.2019.2917057. URL: https://doi.org/10.1109/ACCESS.2019.2917057.

[San+06] Yingpeng Sang et al. "Efficient Protocols for Privacy Preserving Matching Against Distributed Datasets". In: *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*. Ed. by Peng Ning, Sihan Qing, and Ninghui Li. Vol. 4307. Lecture Notes in Computer Science. Springer, 2006, pp. 210–227. DOI: 10.1007/11935308_15. URL: https://doi.org/10.1007/11935308%5C_15.

[Sch+19] Phillipp Schoppmann et al. "Distributed Vector-OLE: Improved Constructions and Implementation". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro et al. ACM, 2019, pp. 1055–1072. DOI: 10.1145/3319535.3363228. URL: https://doi.org/10.1145/3319535.3363228.

[SS07] Yingpeng Sang and Hong Shen. "Privacy Preserving Set Intersection Protocol Secure against Malicious Behaviors". In: *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), 3-6 December 2007, Adelaide, Australia*. Ed. by David S. Munro et al. IEEE Computer Society, 2007, pp. 461–468. DOI: 10.1109/PDCAT.2007.59. URL: https://doi.org/10.1109/PDCAT.2007.59.

[SS08] Yingpeng Sang and Hong Shen. "Privacy preserving set intersection based on bilinear groups". In: *Computer Science 2008, Thirty-First Australasian Computer Science Conference (ACSC2008), Wollongong, NSW, Australia, January 22-25, 2008*. Ed. by Gillian Dobbie and Bernard Mans. Vol. 74. CRPIT. Australian Computer Society, 2008, pp. 47–54. URL: https://dl.acm.org/citation.cfm?id=1378290.

[SS09] Yingpeng Sang and Hong Shen. "Efficient and secure protocols for privacy-preserving set operations". In: *ACM Trans. Inf. Syst. Secur.* 13.1 (2009), 9:1–9:35. DOI: 10.1145/1609956.1609965. URL: https://doi.org/10.1145/1609956.1609965.

[VCE22] Jelle Vos, Mauro Conti, and Zekeriya Erkin. "Fast Multi-party Private Set Operations in the Star Topology from Secure ANDs and ORs". In: *IACR Cryptol. ePrint Arch.* (2022), p. 721. URL: https://eprint.iacr.org/2022/721.

[WBU21] Zhusheng Wang, Karim Banawan, and Sennur Ulukus. "Multi-Party Private Set Intersection: An Information-Theoretic Approach". In: *IEEE J. Sel. Areas Inf. Theory* 2.1 (2021), pp. 366–379. DOI: 10.1109/JSAIT.2021.3057597. URL: https://doi.org/10.1109/JSAIT.2021.3057597.

[Wei+22] Lifei Wei et al. "Efficient and Collusion Resistant Multi-party Private Set Intersection Protocols for Large Participants and Small Sets Setting". In: *Cyberspace Safety and Security - 14th International Symposium, CSS 2022, Xi'an, China, October 16-18, 2022, Proceedings*. Ed. by Xiaofeng Chen, Jian Shen, and Willy Susilo. Vol. 13547. Lecture Notes in Computer Science. Springer, 2022, pp. 118–132. DOI: 10.1007/978-3-031-18067-5_9. URL: https://doi.org/10.1007/978-3-031-18067-5%5C_9.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. "Global-Scale Secure Multiparty Computation". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 39–56. DOI: 10.1145/3133956.3133979. URL: https://doi.org/10.1145/3133956.3133979.

1.A Derived complexities

Using polynomial roots

The protocol by Kissner & Song first requires each party to send encrypted polynomials to t other parties. The leader must also send the final encrypted polynomial to all $n-1$ assistants. The polynomials grow with a constant factor so the communication complexity for an assistant is $O(tk)$ bits, and $O(nk)$ for the leader. Computation-wise, the most expensive part of the protocol is when each party computes the dot product of $t+1$ encrypted polynomials with random polynomials. This takes $O(tk^2)$ cryptographic operations. The protocol takes 3 rounds: encryption, randomization and addition, and decryption.

The most expensive part of the protocol by Li & Wu is the computation phase. Here, each party sends $O(n(k+1)(k+2))$ secret values to t other parties, which takes $O(ntk^2)$ bits. The computation required is at least equal to this complexity. Note that in this protocol, all parties receive the output, so there is no actual notion of a leader. We only consider the semi-honest case here, which takes 3 rounds.

We derive complexities from the analysis by Sang et al. [San+06]. Each party performs $O(n^2k)$ computations. The total communication is $O(n^2k)$, which we divide by n to get the complexity per party. There are 3 rounds of interaction. The second protocol by Sang & Shen [SS09] takes $O(nk^2)$ computations in step 2.2 for all parties, and $O(nk)$ communicated bits in step 2.3. This takes 2 rounds of interaction.

The protocol by Cheon et al. is similar to Kissner & Song's, but with the encrypted polynomial multiplications taking $O(k)$ rather than $O(k^2)$. It is designed so that all parties receive the result, but we change steps 2-4 so only the leader receives the result. In step 2, assistants send their randomized polynomials only to the leader, step 3 is then computed by the leader, and the threshold decryption continues with only t parties. Then, each party performs $O(nk)$ computations in step 2 and each party sends its encrypted polynomial to all others, which takes $O(nk)$ bits. This takes 3 rounds of online communication: encryption, randomization in steps 2 & 3, then threshold decryption.

For the protocol by Ghosh et al. we use the complexities from their paper: the computation complexity for the leader is $O(nk \log k) = \tilde{O}(nk)$ and for the assistant it is identical to the two-party case $\tilde{O}(k)$. The communication complexity is $O(nk)$ for the leader, $O(k)$ for the assistant. The setup takes 1 interaction, share computation takes 4 given that an OPA takes 2 interactions, the output takes 1 interaction.

The protocol by Gordon et al. requires one interaction for the input sharing phase, one interaction for the coin toss, one interaction for the output aggregation, and one interaction for the OLE if it is instantiated using an efficient OT. We copy the leader's communication complexity from the paper: $\tilde{O}(nk + nt)$. The other complexities are non-trivial as they depend on the choice of primitives and parameters.

Sorted multisets

There is no concept of a leader in the protocol by Blanton & Aguiar. We assume the parties pre-sort their sets, which allows the multiset to be sorted using a merge operation requiring $O(k \log k)$ operations (Section 7.1) rather than a full sort. Next, the parties perform $O(k)$ multiplications and secure equality operations. Since each multiplication requires at least t parties to communicate, this requires $O(tk)$ bits for each party. The total communication and computation complexity for one party is at least $O(k \log k + tk)$. For brevity, we denote this by $\tilde{O}(tk)$. As mentioned in Section 8 of their paper, the protocol runs in $O(\log k)$ rounds due to the merge operation at the start.

Using bitsets

The protocol by Bay et al. requires each assistant to send their encrypted bitset to the leader, which takes $O(|\mathcal{U}|)$ bits. After that, communication is restricted to the leader's k bits, which the leader sends to t assistants for randomization and decryption, taking $O(tk)$ bits. For each assistant, computation is dominated by encryption, which takes $O(|\mathcal{U}|)$ operations. The leader's most expensive step is in aggregating the bitsets, but it only has to consider the k elements in its own set,

taking $O(nk)$ operations. In total, the protocol requires 3 rounds of interactions: encryption and aggregation, randomization, and decryption. The protocol by Vos et al. is asymptotically equivalent when using the composed logic sub-protocol. Otherwise, the leader incurs a factor $|\mathcal{U}|$ in computation and communication.

Using Garbled Bloom filters

For the protocol by Inbar et al., we use the complexities reported by the original paper in the semi-honest model. Assuming a two-round OT protocol, the protocol requires 3 rounds of interaction. For the protocol by Kavousi et al. we extract the complexities from Table 2 in the original paper. Assuming a two-round OT protocol, the protocol requires 4 rounds of interaction.

Using Bloom filters

The protocol by Bay et al. requires each assistant to send an encrypted Bloom filter to the leader, which takes $O(k)$. The leader only has to consider its own k elements, which it sends for randomization and decryption to t assistants, taking $O(tk)$ bits. Each assistant encrypts their entire Bloom filter, taking $O(k)$ operations. The leader must aggregate $O(nkh)$ bins. In total, this protocol requires 3 rounds of interactions, as in Section 1.A. The protocol by Vos et al. is asymptotically equivalent to this work.

Using polynomial payloads

The work of Freedman et al. [FNP04] uses the trick from Section 1.5 to transform the protocol to the star topology. For a fair comparison, we consider their scheme without this transformation, requiring private channels between all parties. The complexities are then $O(n^2k^2)$ in each aspect, and the protocol requires 4 rounds.

Using OPPRFs and OKVSs

Nevo et al. [NTY21] provide detailed complexities for the works of Chandran et al. [Cha+21], Garimella et al. [Gar+21], Kolesnikvo et al. [Kol+17], and their own.

1.B Derived operation counts

1.B.I Elliptic curve multiplications

We refer to multiplications with precomputations as PMULs, and to regular ones as MULs. We assume MUL $\approx$ 4PMUL and that the leader always takes part in decryption. In elliptic curve ElGamal, a homomorphic multiplication with a plaintext requires two elliptic curve multiplications.

Sang & Shen

We alter the protocol by Sang & Shen [SS09] to only let the leader receive the result.

1. n parties encrypt their polynomial: $2k$ PMULs.

2. The leader multiplies nk coefficients by a scalar, which takes $2nk$ MULs.

3. The leader evaluates the polynomial k times, each time multiplying k coefficients by a scalar, which takes $2k^2$ MULs in total. Next, $t+1$ parties each decrypt the k resulting encryptions, which takes k MULs in total.

The leader performs $\frac{2}{4}k+2nk+2k^2+k = 1.5k+2nk+2k^2$ MULs. In the worst case, an assistant performs $\frac{2}{4}k+k = 1.5k$ MULs. The total cost is $2nk+2nk+2k^2+(t+1)k = 4nk+2k^2+tk+k$ MULs.

Cheon et al

We alter the protocol to only let the leader receive the result and to make the collusion resistance variable. The polynomials use the point-value representation, so an encrypted polynomial contains $2k$ ciphertexts.

This protocol has two phases. The input data conversion:

1. n parties each encrypt their polynomial: $4k$ PMULs.

2. No MULs.

The online phase is as follows:

1. No computations (polynomials are sent to $t+1$ parties).

2. $t+1$ parties randomize all polynomials: $4nk$ MULs.

3. The leader sums all randomized polynomials: which takes no MULs.

4. $t+1$ parties each decrypt the resulting polynomial, which takes $2k$ MULs.

Worst-case, the leader performs the same effort as an assistant: $\frac{4}{4}k+4nk+2k = 4nk+3k$ MULs. The total cost is $\frac{4}{4}nk+4n(t+1)k+2(t+1)k = 5nk+4ntk+2tk+2k$.

Hazay & Venkitasubramaniam

The protocol has two phases. Note that it uses the coefficient representation, so polynomial multiplication scales quadratically with the degree. The 2PC phase has two steps:

1. $n-1$ assistants encrypt k coefficients: $2k$ PMULs.

2. The leader evaluates $n-1$ polynomials k times, and randomizes (scalars can be multiplied in advance), which takes $(n-1)k(2k) = 2(n-1)k^2$ MULs.

Concluding the intersection goes as follows:

1. $t+1$ parties each randomize the summed ciphertext which takes $2k$ MULs.

2. The leader adds them up, which takes no MULs.

3. $t+1$ parties each decrypt, which takes k MULs.

4. The leader performs additions and zero-checks, which takes no MULs.

An assistant performs $0.5k + 3k = 3.5k$ MULs in the worst case. The leader performs $2(n-1)k^2 + 2k + k = 2(n-1)k^2 + 3k$ MULs. Now consider the balanced allocation optimization. Here, the maximum number of elements in one bin is at most 5 with overwhelming probability. Now, the leader receives $B = \frac{k}{\log\log k}$ bins in the 2PC part, where each set element is assigned to only one bin. So it evaluates $n-1$ polynomials of degree 5 k times. Each homomorphic multiplication also costs two EC multiplications. So, the leader's total is $10(n-1)k + 2k + k = 10nk - 7k$ MULs. This totals $2(n-1)k + 10nk - 7k + 2(t+1)k + (t+1)k = 12nk - 6k + 3tk$.

Vos et al

We go through the protocol step-by-step:

1. $n-1$ assistants each perform $2m$ PMULs.
2. The leader performs $2k$ MULs.
3. t assistants each perform $2k$ MULs.
4. The leader performs no MULs.
5. $t+1$ parties each perform k MULs.
6. The leader performs no MULs.

The leader performs $2k + k = 3k$ MULs. An assistant performs $2m$ PMULs and $2k + k = 3k$ MULs in the worst case. From (1.8) we have that $m \approx 2.08k \ln(\varepsilon^{-1})$, so an assistant performs approximately $1.04k \ln(\varepsilon^{-1}) + 3k$ MULs. Together, the parties perform $\frac{2}{4}m(n-1) + 2k + 2tk + (t+1)k = 1.04(n-1)k \ln(\varepsilon^{-1}) + 3tk + 3k$ MULs.

1.B.II Efficient two-party subprotocols

We use the fact that one OPPRF costs one OPRF [Kol+17], and one OPRF costs approximately 3.5 OTs [Kol+16]. Since OLEs can be instantiated using OT or based on the learning with errors problem [GHL22], we count OLEs separately.

Inbar et al

In Table 1 of their work, Inbar et al. [IOP18] describe that the leader performs mn OT extensions in the augmented semi-honest model, which we reduce to $m(n-1)$ as the leader does not interact with itself. From (1.8) we have that $m \approx 2.08k \ln(\varepsilon^{-1})$, so the leader performs $2.08(n-1)Ek$ OTs. An assistant performs $2.08Ek$ OTs. In total, there are $2.08(n-1)Ek$ OTs (the leader is involved in each of those).

Kolesnikov et al

The protocol first requires each party to perform k OPPRFs with $n-1$ other parties. After that, the leader performs k additional OPPRFs with the $n-1$ assistants. So, the leader performs $2(n-1)k$ OPPRFs and each assistant performs $(n-1)k$ OPPRFs. In total, there are $n(n-1)k$ OPPRFs in the first part, and $(n-1)k$ in the second. One OPPRF is 3.5 OTs.

Chandran et al

We derive the number of OTs for the 'relaxed batch OPPRF' described in Appendix B of Chandran et al. [Cha+21]. Here, a wPSM between two parties requires two rounds of β OPPRFs, where the authors select $\beta = 1.28k$. The wPSM protocol is executed between every assistant and the leader. So, the leader performs $2.56(n-1)k$ OPPRFs, and an assistant $2.56k$. In total: $2.56(n-1)k$ OPPRFs.

Kavousi et al

In Section 3.3 of their work, Kavousi et al. [KMS21] explain how to choose parameter w. The authors propose to set $m = k$ (where k is the number of set elements). Now, we show that p scales regardless of k:

$$p = \left(1 - \frac{1}{k}\right)^k \approx \frac{1}{e}\,, \tag{1.12}$$

which holds as k grows to infinity, but the approximation is already accurate for small k. As a result, w can be a constant. The lowest value causing the probability to fall below 2^{-40} is $w = 558$. We note that if m is variable, one might choose a lower w, trading off computation and communication.

Garimella et al

We consider the multi-party MPSI protocol in Section 7.2 of the work by Garimella et al. [Gar+21] in the star topology. The authors defer an analysis to a full version of the paper, but at the time of writing this paper is unavailable.

Nevo et al

The OPPRFs are executed in the final step of the protocol as part of the zeroXOR functionality between $t+1$ parties. We consider the leader to be part of this group, acting as the receiver. Here, t assistants perform k OPPRFs with the leader. This comes down to tk OPPRFs for the leader, k for an assistant, and tk in total.

Ghosh & Nilges

The OLEs are performed in the OPA subprotocol, which are performed between each assistant and the leader on k elements. Each OPA requires two calls to an OLE. So, the leader performs $2(n-1)k$ OLEs, an assistant performs $2k$ OLEs. In total: $2(n-1)k$ OLEs.

Gordon et al.

The main cost of this protocol comes from OLEs. For one-sided output, each assistant only performs one OLE with the leader per input item. As a result, the leader performs $(n-1)k$ OLEs, an assistant performs k OLEs, and in total $(n-1)k$ OLEs are performed.

Chapter 2

On the Insecurity of Bloom Filter-Based Private Set Intersections

In the previous chapter, we put forward a systematization of multi-party private set intersections. Two methods that we discussed were hash sets and Bloom filters, where hash sets are essentially Bloom filters with one hash function. These methods are particularly promising, because they support low-round protocols in the star topology, thereby addressing impracticalities 1: High interactivity and 2: Full-mesh topology. Following previous literature, we assumed that hash sets realize private homomorphic set representations and Bloom filters realize leaky homomorphic set representations.

In this chapter, we show that there are several problems with both representations. We show that while both representations can be private, this only happens when the parameters are prohibitively large. We also show that Bloom filters cannot realize leaky homomorphic set representations with smaller parameters without allowing for attacks that undermine the privacy properties of MPSI protocols.

This chapter is an adaptation of the work with the same title that has been submitted to IEEE Symposium on Security and Privacy 2025, authored by Jelle Vos, Jorrit van Assen, Tjitske Koster, Evangelia Anna Markatou, and Zekeriya Erkin.

2.1 Introduction

Private set intersection protocols (PSI) and their multi-party equivalent are protocols for computing the intersection between n parties' private sets, without revealing any other information about those private sets. These protocols enable information sharing in situations where revealing data would be undesirable, like in financial transactions, or where information sharing must be limited, like in threat intelligence or no-fly lists. More formally, a private set intersection protocol is a protocol between n parties $\mathcal{P}_i$ for $i = 1, \ldots, n$. Each party has a private set $X_i \subseteq \mathcal{U}$ of at most k elements. One party that we refer to as the leader (denoted $\mathcal{P}_1$) obtains the intersection $X_1 \cap \cdots \cap X_n$ as the protocol's output. All other attributes of the private sets must remain hidden.

Approximate PSI schemes allow a trade-off between the computational and communicational cost of a protocol and the accuracy of the resulting intersection. A common method for constructing efficient approximate PSI protocols is to use Bloom filters. Bloom filter-based PSI protocols let parties first encode their sets as

Bloom filters $\hat{X}_i \leftarrow \mathsf{Encode}(X_i)$ using h hash functions H_j for $j = 1, \ldots, h$. These filters start out as an indexed set of m Boolean bins that are all set to 0. Each party $\mathcal{P}_i$ uses the filter's hash functions to map each of their elements in set X_i to h of the bins, setting them to 1. One can compute a Bloom filter representing the intersection by combining the Bloom filters using an element-wise logical `AND` operation. The `AND` operation must be performed on the Bloom filters, which must remain private, using a secure computation technique such as homomorphic encryption.

The approximation inherent to Bloom filters is caused by the possibility of hash collisions: a hash of any two distinct elements may map to the same bin (i.e. $H_i(x) = H_j(x')$ where $x \neq x'$). As such, such a Bloom filter-based protocol will never wrongfully exclude elements from the intersection (i.e. there are no false negatives), but the result may include false positives with some probability. Specifically, each negative element in the leader's set may wrongfully appear in the intersection with probability at most p. This makes Bloom filter-based MPSI protocols suitable for use cases, in which false positives may be permissible with a small but non-negligible probability.

Previous work [DCW13; VCE23] has shown that the Bloom filter representing the intersection might leak information if it is revealed. This is because bins in the intersection may be set to 1, even if the same bins are set to 0 when the Bloom filter is obtained by directly encoding the intersection, $\mathsf{Encode}(X_1 \cap \cdots \cap X_n)$. Instead of revealing the combined Bloom filter $\hat{X}_\cap = \hat{X}_1 \wedge \cdots \wedge \hat{X}_n$, private set intersection protocols use secure computation techniques to query the filter on every element in the leader's set and only reveal the result:

$$\hat{X}_\cap[H_1(x)] \wedge \cdots \wedge \hat{X}_\cap[H_h(x)] \text{ for } x \in X_1 \ . \tag{2.1}$$

One might think that this constitutes a secure Bloom filter-based private set intersection protocol, as the leader would not be able to distinguish between false positives and actual elements in the intersection, preventing it from exploiting $\hat{X}_\cap$ to learn anything about the private sets. However, this assumes that the leader has no auxiliary knowledge about the private sets. Rindal & Rosulek [RR17] already showed that Bloom filters lead to problems in security proofs in the malicious setting, and other recent work by Liu et al. [LLT24] identifies problems with Bloom filter-based private set unions. Egert et al. [Ege+15] showed that revealing a Bloom filter with entries pertaining to a random unknown subset of hash functions allows an attacker to perform membership attacks in the context of union-cardinality.

In this work, we show that the above approach is not sufficient to achieve secure MPSI: the approximate nature of Bloom filters does, in fact, allow the leader to learn information about the private sets that it could not from the exact intersection. What is more, previous works do not take this approximation into account in their security proofs. For example, several works prove security with respect to the ideal functionality of an exact PSI to model the security of Bloom filter-based protocols, as opposed to the ideal functionality of an approximate PSI. This gap caused by approximation can only be closed if p is negligible, but this, in turn, causes parameters to grow significantly. In this work, we show that these parameters are so large that they undo the performance benefit of choosing an approximate protocol in the first place.

One might think that the gap can be easily closed by proving security with respect to an ideal functionality for approximate set intersections. Unfortunately, we show that Bloom filter-based PSI protocols are fundamentally flawed in this regard. Wo do so by exploiting the fact that the actual false positive rate is not constant; it depends on the elements in the private sets. The result is that the existence of a false positive in the final intersection may reveal information about any of the private sets. Consider the following minimal example (albeit slightly contrived), which demonstrates that the false positive rate of a Bloom filter does not only affect the correctness but also the security of the protocol. In other words, even a perfectly secure Bloom filter-based PSI leaks information about the input sets with non-negligible probability.

Example 7. Let us analyze the situation where $h = 1$, $n = 2$, and $k = 1$. Assume that we have two distinct party-specific universes, $\mathcal{U}_1 = \{\mathrm{a}\}$ and $\mathcal{U}_2 = \{\mathrm{b}\}$, and we use a Bloom filter-based PSI protocol in which the two parties input sets X_1 and X_2. Then, in the ideal world corresponding to $\varepsilon_{\mathrm{fp}}$-approximate PSI, the leader would get a non-empty intersection with probability $\varepsilon_{\mathrm{fp}}$, regardless of X_2. However, in the real world, the output is non-empty *with probability* $\frac{1}{m}$ if, and only if, $X_1 = \{\mathrm{a}\}$ and $X_2 = \{\mathrm{b}\}$, but is always empty otherwise. I.e. the leader learns the other party's set with probability $\frac{1}{m}$.

The attacks described in this work are all in the augmented semi-honest model; where parties can freely choose their inputs, after which they do not deviate from the protocol. This is an augmentation of the semi-honest model, in which parties that are not corrupted do not deviate from their predetermined inputs to the protocol. It has been shown that this property of the semi-honest model leads to counter-intuitive situations in which a protocol that is secure in the malicious model cannot be proven to be secure in the semi-honest model [HL10].

As a result of the attacks we propose, Bloom filter-based private set intersection protocols that use a non-negligible false positive probability will become slower. The easiest mitigation is to lower the false positive probability, slowing down the protocol due to the use of larger Bloom filters. Alternative solutions would include using oblivious pseudo-random functions [CHL22] such that the hash functions can remain secret, or switching to hash functions that are very expensive to compute, such as password-based key derivation functions like PBKDF2 [Kal00].

The paper is organized as follows. We proceed with a description of Bloom filters in Section 2.2. After that, in Section 2.3, we define the security model for multi-party private set intersections that we use, including definitions for approximate MPSI. Next, we present an abstraction of Bloom filter-based MPSI protocols in Section 2.4. We present our main results in Sections 2.5 and 2.6, in which we put forward our theoretical analysis and our practical attack. We finish by discussing mitigations in Section 2.7, and we conclude in Section 2.8.

2.2 Bloom Filters

Recall from the previous chapter that Bloom filters are probabilistic data structures for efficient set membership queries. Bloom filters exploit the uniformity of h hash functions to each map an element $x \in \mathcal{U}$ to one of m bins. Each bin contains one

bit that all start at 0. To encode an element x, we hash x with h hash functions H_i `for` $i \in \{1, 2, \ldots, h\}$, which select bins of the Bloom filters to be set to 1. We can encode a private set X by encoding each element $x \in X$ individually. We define the notation $\hat{X} \leftarrow \mathsf{Enc}X$ to denote the Bloom filter encoding of set $X \subseteq \mathcal{U}$.

One can test whether an element y is contained in the Bloom filter $\hat{X}$ by computing all bins corresponding to y and checking if indeed all bits are 1.False negatives cannot occur, but when all of the bins of element y are set to 1, it is not sure that the element was indeed encoded in the Bloom filter, or that the bins were set to 1 by encoding other elements. I.e. false positives *can* occur. We use p to denote an upper bound on the probability of a Bloom filter encoding k distinct elements returning a false positive. The probability p depends on the number k of elements inserted, the size of the Bloom filter m and the number h of hash functions used. An upper bound was derived by Goel and Gupta [GG10].

$$p \leq \left(1 - e^{-\frac{h(N+0.5)}{m-1}}\right)^h . \tag{2.2}$$

Given a desired false positive probability ε_{fp} and maximum set size k, we can calculate the corresponding required number of hash functions h and the minimal number of bins m_{opt} as follows:

$$h = -\log_2(\varepsilon_{\text{fp}}) , \tag{2.3}$$

$$m_{opt} \geq \frac{-h\,(k + 0.5)}{\ln\left(1 - \sqrt[h]{p}\right)} + 1 . \tag{2.4}$$

The protocols discussed in this work use Bloom filters to compute intersections. One way in which Bloom filters are convenient for this purpose, is that different Bloom filters can be combined to generate a filter representing the intersection. Specifically, two Bloom filters $\hat{X}_1$ and $\hat{X}_2$ can be combined using a bin-wise `AND` operation to generate a Bloom filter $\hat{X}_\cap$ representing the intersection. Bloom filters do not allow for efficient extraction of the original elements, however, they do allow for efficient testing of the inclusion of a specified value $x \in \mathcal{U}$, so one can extract the intersection by querying the elements from one of the original sets.

2.3 Definition of MPSI security

The results in our work contradict the security proofs in previous work [Deb+21; Bay+22]: we show that Bloom filter-based private set intersections with non-negligible false positive probabilities cannot securely realize private set intersections, whereas previous work contains security proofs for the opposite. In this section, we first discuss how the security definitions and analyses of previous work and show how these are flawed. One of these flaws is that the security proofs attempt to show that Bloom filter-based PSI realizes *exact* PSI, but this is clearly not true when false positives occur. In the second part of this section, we propose new definitions for *approximate* PSI. In Section 2.5, we show that Bloom filter-based PSI also does not realize these weaker functionalities.

2.3.1 Definitions & flaws in existing security proofs

We now inspect the security proofs of Bloom filter-based PSI protocols more closely. We specifically focus on works that use the approximation of a Bloom filter to speed up the protocol. In other words; works that set the false positive probability p to be non-negligible.

In their proof (Theorem 4.2), Debnath et al. [Deb+21] assume that the output is the exact intersection, claiming that the protocol only fails with the false positive probability p. However, in reality, the security proof does not hold the moment that any false positive occurs. This can happen with probability $1 - (1 - p)^k$. Bay et al. [Bay+22] also assume that the output is the exact intersection, so the proof does not hold with probability this probability, but their proof is slightly different. They simulate the inputs of uncorrupted parties by choosing random input sets that conform to the intersection. Given that the combined Bloom filter highly depends on the input sets, this potentially skews the advantage even more.

Other works, like that by Vos et al. [VCE22] only prove that the aggregation is secure, so the security proof does not extend to the final computation of the intersection. The same goes for the work by Miyaji et al. [MNN17], which only considers security for the protocol before decryption.

To summarize, all these proofs either use the MPSI functionality, thereby failing to consider false positives, or the proofs are incomplete (because they do not consider Bloom filters). There is an option for remedying these proofs, namely including the approximate behaviour of the underlying protocol and showing that false positives occur with a negligible probability, e.g. $p \leq 2^{-40}$. However, for the addressed schemes, this results in a significant decrease in performance. There are already other works that take this approach, such as the work by Ben Efraim et al. [Ben+22]. If false positives practically never occur, then the protocol behaves as an exact intersection.

2.3.2 An exact ideal functionality

To treat two-party private set intersections and multi-party private set intersections in general, we define an exact ideal functionality $\mathcal{F}_{\text{MPSI}}$ for MPSI that roughly follows the universal composability model. In this functionality, $\mathcal{S}$ is essentially an external adversary that controls the communication channels. In this work, it is sufficient to think of the $\mathcal{S}$ as an external influence that decides when the protocol finishes. The ideal functionality is on the next page.

2.3.3 An approximate ideal functionality

As mentioned before, the exact ideal functionality is unsuitable for proving security of Bloom filter-based MPSI when the false positive probability is not negligible. After all, any false positive would allow a distinguisher to tell it apart from the exact MPSI ideal functionality. Instead, we define an approximate MPSI ideal functionality $\mathcal{F}_{\text{aMPSI}}$ that returns an intersection based on the leader's set with a constant probability of false positives ε_{fp} and false negatives ε_{fn}. In the rest of our paper, $\varepsilon_{\text{fn}} = 0$. We refer to an approximate MPSI protocol with false positive probability ε_{fp} as ε_{fp}-approximate. The ideal functionality is on the next page.

$\mathcal{F}_{\text{MPSI}}$

Let $X_\cap \leftarrow \mathcal{U}$ and $P = \emptyset$.

On (inp, X_i) **from** $\mathcal{P}_i$:

- *Assert that* this is the first input of $\mathcal{P}_i$
- *Assert that* $X_i \subseteq \mathcal{U}$
- *Assert that* $|X_i| \leq k$
- Store $X_\cap \leftarrow X_\cap \cap X_i$
- Store $P \leftarrow P \cup \{\mathcal{P}_i\}$
- Send $(\text{inp}, \mathcal{P}_i)$ to $\mathcal{S}$

On (finish) **from** $\mathcal{S}$:

- *Assert that* this is the first finish request
- *Assert that* $\mathcal{P}_i \in P$ for all $i \in [1, n]$
- Send $(X_\cap)$ to $\mathcal{P}_1$

$\mathcal{F}_{\text{aMPSI}}$

Let $X_\cap \leftarrow \mathcal{U}$ and $P = \emptyset$.

On (inp, X_i) **from** $\mathcal{P}_i$:

- *Assert that* this is the first input of $\mathcal{P}_i$
- *Assert that* $X_i \subseteq \mathcal{U}$
- *Assert that* $|X_i| \leq k$
- Store $X_\cap \leftarrow X_\cap \cap X_i$
- Store $P \leftarrow P \cup \{\mathcal{P}_i\}$
- Send $(\text{inp}, \mathcal{P}_i)$ to $\mathcal{S}$

On (finish) **from** $\mathcal{S}$:

- *Assert that* this is the first finish request
- *Assert that* $\mathcal{P}_i \in P$ for all $i \in [1, n]$
- Initialize $R \leftarrow \emptyset$
- For $x \in X_\cap$: add x to R with prob. $1 - \varepsilon_{\text{fn}}$
- For $x \in X_1 / X_\cap$: add x to R with prob. ε_{fp}
- Send (R) to $\mathcal{P}_1$

$\mathcal{F}_{\text{waMPSI}}$

Let $X_\cap \leftarrow \mathcal{U}$ and $P = \emptyset$.

On (inp, X_i) **from** $\mathcal{P}_i$:

- *Assert that* this is the first input of $\mathcal{P}_i$
- *Assert that* $X_i \subseteq \mathcal{U}$
- *Assert that* $|X_i| \leq k$
- Store $X_\cap \leftarrow X_\cap \cap X_i$
- Store $P \leftarrow P \cup \{\mathcal{P}_i\}$
- Send (inp, $\mathcal{P}_i$) to $\mathcal{S}$

On (finish) **from** $\mathcal{S}$:

- *Assert that* this is the first finish request
- *Assert that* $\mathcal{P}_i \in P$ for all $i \in [1, n]$
- Initialize $R \leftarrow \emptyset$
- $\varepsilon_{\text{fn}} \leftarrow f_{\text{fn}}(|X_1|, \ldots, |X_n|, |X_1 \cap \cdots \cap X_n|)$
- $\varepsilon_{\text{fp}} \leftarrow f_{\text{fp}}(|X_1|, \ldots, |X_n|, |X_1 \cap \cdots \cap X_n|)$
- For $x \in X_\cap$: add x to R with prob. $1 - \varepsilon_{\text{fn}}$
- For $x \in X_1 / X_\cap$: add x to R with prob. ε_{fp}
- Send (R) to $\mathcal{P}_1$

2.3.4 A weaker ideal functionality

In Section 2.5, we show that Bloom filters with a non-negligible false positive probability also cannot securely realize $\mathcal{F}_{\text{aMPSI}}$. One might argue that the only reason why Bloom filters are not approximate MPSIs is that their false positive probability varies, but that this variance is only induced by some values that can be permitted to be leaked. E.g. one might argue that the size of the input sets and the size of the exact intersection is not secret. As such, we define a weaker functionality called $\mathcal{F}_{\text{waMPSI}}$ in which ε_{fp} and ε_{fn} are functions of the sizes of the sets: $|X_i|$ for $i = 1, \ldots, n$ and $|X_\cap|$. We denote these functions by f_{fp} and f_{fn}.

2.4 An abstraction of Bloom filter-based PSI

In this work, we set out to show that Bloom filters are fundamentally flawed. Instead of going through each Bloom filter-based protocol individually and showing that they suffer from security problems, we present an idealized abstraction of Bloom filter-based PSI. After that, we discuss previously proposed protocols and how each inherits the security problems from our idealized abstraction.

$\Pi_{\mathbf{BF}}$

Let $\hat{X}_\cap \leftarrow 1^m$ and $P = \emptyset$.

On (inp, X_i) **from** $\mathcal{P}_i$:

- *Assert that* this is the first input of $\mathcal{P}_i$
- *Assert that* $X_i \subseteq \mathcal{U}$
- *Assert that* $|X_i| \leq k$
- Store $\hat{X}_\cap \leftarrow \hat{X}_\cap \wedge \mathsf{Enc} X_i$
- Store $P \leftarrow P \cup \{\mathcal{P}_i\}$
- Send $(\mathsf{inp}, \mathcal{P}_i)$ to $\mathcal{S}$

On (finish) **from** $\mathcal{S}$:

- *Assert that* this is the first finish request
- *Assert that* $\mathcal{P}_i \in P$ for all $i \in [1, n]$
- Initialize $R \leftarrow \emptyset$
- For $x \in X_1$: Add x to R if $\mathsf{contains}(\hat{X}_\cap, x)$
- Send (R) to $\mathcal{P}_1$

2.4.1 Our idealized abstraction

The idea of our idealized abstraction is to model the behavior of Bloom filters in isolation; without communication between individual parties or use of cryptographic primitives. We present this abstraction Π_{BF} in such a way that it has the same interface as the ideal functionalities defined in the previous section. Π_{BF} is conceptually simple: instead of combining the private sets using an actual intersection, it encodes sets as Bloom filters and combines those instead. It returns the intersection to the leader by returning the leader's elements that are contained in the resulting Bloom filter.

2.4.2 Two-party private set intersections

We first consider two-party protocols, explaining the general workings of these protocols and how it might be possible to create a simulator for them around Π_{BF}. The idea is that any problems inherent to Π_{BF} are inherited by the protocols below.

Debnath and Dutta

Debnath & Dutta [DD15] propose a PSI protocol using Goldwasser-Micali encryption and inverted Bloom filters. The client $\mathcal{P}_1$ and server $\mathcal{P}_2$ agree on k hash functions to make the Bloom filter, and the client generates an inverted and encrypted Bloom filter and sends it to the server. For each of its elements, the

server selects the bins that the element maps to and homomorphically XORs a hash of the element onto these bins. So, if an element is contained in the inverted Bloom filter of $\mathcal{P}_1$, the result is an encryption of a hash of the element. If the element is not contained in it, the result is a distorted hash. The client can extract the intersection by checking which elements match the hashes it receives.

We can simulate this protocol using Π_{BF} with high probability. Notice that while Π_{BF} outputs $\{x \in X_1 | \text{contains}(\hat{X}_1 \wedge \hat{X}_2, x)\}$, the protocol by Debnath & Dutta outputs $\{x \in X_2 \mid \text{contains}(\hat{X}_1, x)\}$ with high probability. However, these are the same because Bloom filters do not cause false negatives, so $x \in X_2 \implies \text{contains}(\hat{X}_2, x)$. Besides this, the simulator must still simulate the encryptions that are sent from the client to the server and back.

Davidson and Cid

Davidson and Cid [DC17] also propose a private set intersection protocol based on encrypted and inverted Bloom filters, which was reformulated by Bay et al. [Bay+22]. We discuss this reformulation. [1] The client $\mathcal{P}_1$ encodes their elements X_1 in a Bloom filter as usual and inverts it (i.e. flipping all bits of the filter) before encrypting it. It then sends this filter to the server $\mathcal{P}_2$. For each element $y \in X_2$ of the servers set, the server calculates the corresponding bins in the encrypted inverted Bloom filter and sums the values in these bins with outcome S. It then adds S to the encrypted value of y. It returns the pair $(S, S + \text{enc}(y))$ to the client who computes the intersection. For an element $y \in X_2$, all its corresponding bins in the inverted Bloom filter have value 0, and thus S is the encryption of 0. If this encrypted value of 0 is added to the encrypted value of y the decryption gives $0 + y$. For any element not in the intersection, the decryption reveals nothing about y. [2] For each pair, the client checks whether the decryption of S equals 0; if so, it decrypts the second value of the pair and assumes it to be in the intersection.

One would roughly simulate this protocol using Π_{BF} as follows. The encrypted inverted Bloom filter would be made up of m random encryptions, and the server returns k pairs of specific encryptions to the client. The elements in the pair are encryptions of 0 if the element is in the intersection returned by Π_{BF}, and random otherwise.

2.4.3 Multi-party private set intersections

Next, we discuss several Bloom filter-based multi-party private set intersection protocols and how they relate to Π_{BF}.

Bay, Erkin, Hoepman, Samardjiska, and Vos

The protocol proposed by Bay et al. [Bay+22] is an extension of [DC17] in the multi-party variant. The difference is that the server learns the intersection instead

[1] There seems to be a mistake in the original work because when an element is in the Bloom filter, the sum of the selected bins is 0, so the client would not learn the values that are in the intersection.

[2] That said, both versions of this protocol seemingly reveal the number of bins that were set in the Bloom filter.

of the clients. The adjustments of the protocol are minor. All clients have a private secret key, but the public key corresponding to all private keys is shared. All clients calculate the encrypted inverted Bloom filter with the public key. The server combines the Bloom filters to calculate the sum S for all elements x in its set. Here S is the same as defined in Section 2.4.2. The clients jointly decrypt the encrypted value so that the server can learn if x is in the intersection. This protocol can be simulated using Π_{BF} in a similar way as in Section 2.4.2. A similar protocol was presented by Debnath et al. [Deb+21].

Vos, Conti, and Erkin

Vos et al. [VCE22] (see Chapter 3) propose a similar MPSI protocol for large universes using ElGamal encryption. Each of the parties starts by computing a Bloom filter for their input. Then they invert this Bloom filter so that an element x_i is in set X_i if all its corresponding bins have value 0. Then, the protocol securely performs an `OR` operation on all inverted Bloom filters. The resulting Bloom filter is inverted again, and then an element is in the intersection if all its corresponding bins have value 1. This is equivalent to performing an `AND` operation on regular Bloom filters. Vos et al. already show how to simulate the OR protocol, so the simulation around Π_{BF} is straightforward.

Ruan, Yan, Zhou, and Ai

Ruan et al. [Rua+23] present an MPSI protocol for unbalanced scenarios where the server $\mathcal{P}_1$ has a significantly larger set. Each of the clients $\mathcal{P}_2, \ldots, \mathcal{P}_n$ computes a Bloom filter for their input set X_i. The bins of the Bloom filter that contain 0 are randomized to any number but zero and one. This is needed to apply an ElGamal encryption scheme that cannot encrypt 0. A trusted third-party generates an ElGamal public-private key pair (pk, sk) and divides the secret key over the clients sk_i. Each client $\mathcal{P}_i$ for $i \geq 2$ generates a Bloom filter on their input set X_i, randomizes the bins of value 0, and encrypts the filter with the public key. The resulting filter is sent to the server $\mathcal{P}_1$. The server then selects the bins of the Bloom filter $\hat{X}_i$ pertaining to its elements for each $i \geq 2$, homomorphically aggregates them, and sends the results back to the clients. The clients each perform a computation on the received Bloom filter such that the server is able to combine all filters to decrypt the intersection. Simulation using Π_{BF} would be a multi-party extension of the simulation described for the protocol by Debnath et al. [DD15].

Ruan and Ai

Ruan and Ai [RA23] made an MPSI protocol for the balanced scenario that is much like the unbalanced scenario. All clients compute the encrypted Bloom filter in the same manner as in [Rua+23]. The server does the exact same computation as the clients. All these encrypted Bloom filters are sent to the server, which combines them and sends the combined filter back to the clients. All clients decrypt this Bloom filter using their own private key and send the result to the server. The server then can combine all Bloom filters to find the decrypted Bloom filter of the

intersection $\hat{X}_\cap$. The server then performs the normal contains$(\hat{X}_\cap, x)$ function for all its elements $x \in X_1$.

2.4.4 Outsourced private set intersections

Since the ideal functionalities nor the idealized abstraction Π_{BF} describe *who* computes something, they also apply to the outsourced computation case, in which most of the computations are performed by a server that does not take part in the protocol. We cover two Bloom filter-based outsourced private set intersection protocols.

Qiu, Zhang, Liu, Yan, and Cheng

Qiu et al. [Qiu+22] propose a PSI protocol where both clients $\mathcal{P}_1$ and $\mathcal{P}_2$ learn the intersection, and the computation is done by a computational powerful server S. Both clients compute a Bloom filter and encrypt this filter with a shared secret key. They permute the filter with a secret shared permutation π and forward it to the server. The server then computes which indices of the received filters are equal and forwards this set of indices to the clients. Both clients perform an inverted permutation on this set of indices to obtain the indices of the original Bloom filter. For each element x in the set X_1, $\mathcal{P}_1$ checks whether all bins in the Bloom filter are set to 1 and indicated by the server. If so, the element x is considered to be in the intersection. $\mathcal{P}_2$ does the same computation. Due to the fact that the parties undo the permutation, the resulting behavior is exactly the same as that in Π_{BF}. One would still have to simulate the communication between the parties and the server.

Miyaji and Nishida

Miyaji and Nishida [MN15] propose a multiparty private set intersection based on Bloom filters and a distributed ex-El Gamal encryption. For this protocol, they use a dealer D who does not participate in the intersection and only helps reduce the computational power for the clients. Each client $\mathcal{P}_i$ generates a secret key x_i and a public key g^{x_i}. The jointly public key of the clients is constructed as $pk = \prod_{i=1}^{n} g^{x_i}$. Each of the clients computes their Bloom filter, encrypts it with the public key pk, and sends it to the dealer. The dealer aggregates the Bloom filters by homomorphically adding them. To compute the Bloom filter of the intersection, the dealer subtracts n (the number of clients) from each entry of the Bloom filter. It then sends the resulting Bloom filter to all the clients for joint decryption. If, for an element, all its corresponding bins have the value 0, the element is considered to be in the intersection. This is functionally the same as the protocol by Bay et al. [Bay+22], but without inverting the Bloom filters.

2.5 Analysis of Bloom filter-based PSI

In this section we show that there are fundamental limitations to the security of Bloom filter-based private set intersections. We do so by showing that a certain indistinguishability notion cannot be met when the false positive probability

is not negligible. These problems are caused by the fact that a Bloom filter becomes deterministic when its hash functions are fixed. We first discuss the indistinguishability game that we use to define security, after which we present a distinguisher that reliably tells apart $\mathcal{F}_{\text{waMPSI}}$ from Π_{BF} when false positives occur with non-negligible probability. Based on these results we conclude that the upper bound on the false positive probability p of a Bloom filter must be less than $\frac{0.5}{|\mathcal{U}|-k}$ in practice. After that, we discuss how one can choose concrete Bloom filter parameters to securely realize $\mathcal{F}_{\text{waMPSI}}$ or $\mathcal{F}_{\text{aMPSI}}$ for small values of ε_{fp}.

2.5.1 The indistinguishability game

In the universal composability (UC) framework [Can01], the security of a protocol is defined by the advantage with which a distinguisher can tell it apart from the ideal functionality that it is designed to realize. In this section, we present lower bounds for this advantage, which allows us to show that there are many choices of parameters for which Bloom filter-based private set intersection protocols cannot be UC-secure (or the security guarantees would be broken with high probability). To do so, we consider the advantage of a distinguisher $\mathcal{D}$ in distinguishing the abstract Bloom filter-based MPSI protocol Π_{BF} from some ideal functionality $\mathcal{F}$:

$$\text{Adv}_{\text{ind}}^{\Pi_{\text{BF}}}(\mathcal{D}) = 2\left|\Pr[\mathcal{D}(\Pi) = \Pi \mid \Pi \in_R \{\mathcal{F}, \Pi_{\text{BF}}\}] - \frac{1}{2}\right| \tag{2.5}$$

Specifically, we are interested in analyzing the minimal advantage when distinguishing Π_{BF} from the weakest ideal functionality $\mathcal{F}_{\text{waMPSI}}$, which would allow us to draw the strongest conclusions. In the remainder of this section, we propose such a strong distinguisher, and we show that it only fails with low probability for any value of ε_{fp}.

2.5.2 A reliable distinguisher

In this section, we show that the false positive probability of a Bloom filter denoted by p must be negligible for such a Bloom filter-based private set intersection protocol to realize a (weakly-)approximate private set intersection. We do so by showing that the idealized Bloom filter-based PSI Π_{BF} can be distinguished with relative ease from $\mathcal{F}_{\text{waMPSI}}$ by a distinguisher that learns the result (so $\mathcal{P}_1$ is corrupted). Let $\varepsilon_{\text{fp}} = f_{\text{fp}}(k, k, 0)$. We propose the following distinguisher:

$$\mathcal{D}(\Pi) \leftarrow \begin{cases} \mathcal{D}_{\text{FPs}}(\Pi) & \text{If } p(|\mathcal{U}| - k) \geq \varepsilon_{\text{fp}} k \\ \mathcal{D}_{\text{TNs}}(\Pi) & \text{Otherwise} \end{cases} \tag{2.6}$$

Note that this distinguisher would also apply to $\mathcal{F}_{\text{aMPSI}}$.

Depending on the parameters k, p, $|\mathcal{U}|$, and ε_{fp}, this distinguisher calls $\mathcal{D}_{\text{FPs}}$ or $\mathcal{D}_{\text{TNs}}$. We define $\mathcal{D}_{\text{FPs}}$ as follows:

1. $\mathcal{D}_{\text{FPs}}$ chooses random $X_2 \subset \mathcal{U}$ such that $|X_2| = k$, and computes $\hat{X_2} \leftarrow \text{Enc}X_2$.
2. $\mathcal{D}_{\text{FPs}}$ chooses $X_1 \subseteq \mathcal{U}/X_2$ such that $|X_1| = k$, maximizing the number of elements $x \in X_1$ for which it holds that $\text{contains}(x, \hat{X_2})$; false positives.

3. $\mathcal{D}_{\text{FPs}}$ lets parties $\mathcal{P}_1$ and $\mathcal{P}_2$ input X_1 and X_2, respectively. It waits for $\mathcal{S}$ to send (finish).

4. $\mathcal{P}_1$ is corrupted, so $\mathcal{D}$ receives (R). If R matches the expected output of Π_{BF}, $\mathcal{D}_{\text{FPs}}$ guesses that $\Pi = \Pi_{\text{BF}}$. Otherwise, it guesses that $\Pi = \mathcal{F}_{\text{waMPSI}}$.

The other distinguisher, $\mathcal{D}_{\text{TNs}}$, follows $\mathcal{D}_{\text{FPs}}$ but it maximizes the number of true negatives, so step 2 is different:

2. $\mathcal{D}_{\text{TNs}}$ chooses $X_1 \subseteq \mathcal{U}/X_2$ such that $|X_1| = k$, maximizing the number of elements $x \in X_1$ for which $\text{contains}(x, \hat{X}_2)$ is false; true negatives.

It is easy to extend the distinguisher to more than two parties. The following two lemmas express the probability with which these distinguishers fail. We use these results to derive $\text{Adv}^{\Pi_{\text{BF}}}_{\text{ind}}(\mathcal{D})$, see (2.5).

Lemma 1. *$\mathcal{D}_{FPs}$ always correctly identifies Π_{BF}, but it sometimes misclassifies $\mathcal{F}_{waMPSI}$. It does so with the following probability:*

$$\Pr[\mathcal{D}_{FPs}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] = \sum_{i=0}^{k-1} \Pr[FPs = i]\varepsilon_{fp}^{i}(1-\varepsilon_{fp})^{k-1-i} + \Pr[FPs \geq k]\varepsilon_{fp}^{k}$$

Proof. $\mathcal{D}_{\text{FPs}}$ maximizes the number of false positives in X_1, but it is not guaranteed to find such elements in $\mathcal{U}/X_2$. We use $\Pr[\text{FPs} = i]$ to denote the probability with which $\mathcal{D}_{\text{FPs}}$ finds i false positives. The final probability is:

$$\begin{aligned}\Pr[\mathcal{D}_{\text{FPs}}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] = \\ \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \cdot \Pr[\mathcal{D}_{\text{FPs}}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}} \mid \text{FPs} = i] \\ + \Pr[\text{FPs} \geq k] \cdot \Pr[\mathcal{D}_{\text{FPs}}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}} \mid \text{FPs} \geq k]\end{aligned}$$

The probability that $\mathcal{D}_{\text{FPs}}$ misclassifies $\mathcal{F}_{\text{waMPSI}}$ is the probability that Π_{BF} would return R on inputs X_1 and X_2. So, each false positive in X_1 is included in R, which happens with probability $\varepsilon_{\text{fp}}^{i}$. Moreover, each true negative should *not* be in R, which happens with probability $(1-\varepsilon_{\text{fp}})^{k-1-i}$. In other words:

$$\Pr[\mathcal{D}_{\text{FPs}}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}} \mid \text{FPs} \geq k] = \varepsilon_{\text{fp}}^{i}(1-\varepsilon_{\text{fp}})^{k-1-i}$$

Notice that when $\text{FPs} \geq k$, the distinguisher simply chooses k false positives, so $\Pr[\mathcal{D}_{\text{FPs}}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}} \mid \text{FPs} = j] = \varepsilon_{\text{fp}}^{k}$ for all $j \geq k$. This proves our lemma. □

Lemma 2. *$\mathcal{D}_{TNs}$ always correctly identifies Π_{BF}, but it sometimes misclassifies $\mathcal{F}_{waMPSI}$. It does so with the following probability:*

$$\Pr[\mathcal{D}_{TNs}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] = \sum_{i=0}^{k-1} \Pr[TNs = i](1-\varepsilon_{fp})^{i}\varepsilon_{fp}^{k-1-i} + \Pr[TNs \geq k](1-\varepsilon_{fp})^{k}.$$

Proof. This proof follows similarly to that of Lemma 1. □

The condition $p(|\mathcal{U}| - k) \geq \varepsilon_{\text{fp}} k$ marks the point beyond which one expects to find more false positives in $\mathcal{U}/X_2$ than the number of false positives that one expects Π_{BF} to output.

In the remainder of this section, we define three different scenarios based on two conditions, depending on the median of the binomial distribution of Pr[FPs]. These conditions are as follows (see Appendix 2.A for more details):

- When $p < \frac{1}{|\mathcal{U}|-k}$, the median is at or below Pr[FPs = 0], so Pr[FPs = 0] $\geq \frac{1}{2}$. See Lemma 7.
- When $p > \frac{k-1}{|\mathcal{U}|-k}$, the median is at or above Pr[FPs = k], so Pr[FPs = k] $\geq \frac{1}{2}$. See Lemma 8.

2.5.3 Upper bounds on the failure probability

We want to obtain upper bounds on the failure probability independent of ε_{fp}. This allows us to make statements about the security gap that arises when trying to realize $\mathcal{F}_{\text{waMPSI}}$ using a Bloom filter-based protocol regardless of the choice of ε_{fp} in the ideal functionality. Our main result is an upper bound on the failure probability of our distinguisher for all values of p, which we present in Theorem 6. We provide a summary in Figure 2.1, showing among others, that the attack succeeds with high probability when $\frac{1}{|\mathcal{U}|-k} \leq p \leq \frac{k-1}{|\mathcal{U}|-k}$, or when $p > \frac{k-1}{|\mathcal{U}|-k}$ and k is large.

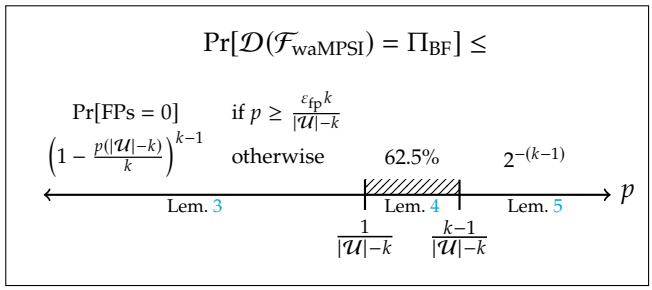


Figure 2.1: Upper bounds on the distinguisher's failure probability for different values of the false positive probability p. The bounds depend on the size of the input sets k, the size of the universe $|\mathcal{U}|$, and the false positive probability $\varepsilon_{\text{fp}} \leftarrow f_{\text{fp}}(k, k, 0)$ of $\mathcal{F}_{\text{waMPSI}}$. The attack success probability cannot be made negligible in the shaded area.

Our first lemma considers the case where p is so small that Pr[FPs = 0] is the most likely (and the same holds for Pr[TNs $\geq k$]). We obtain different bounds depending on whether $\mathcal{D} = \mathcal{D}_{\text{FPs}}$ or $\mathcal{D} = \mathcal{D}_{\text{TNs}}$. For the proofs we often use the fact that $\varepsilon_{\text{fp}} \leq (1 - \varepsilon_{\text{fp}})$ for $\varepsilon_{\text{fp}} \leq \frac{1}{2}$.

Lemma 3. *If $p < \frac{1}{|\mathcal{U}|-k}$, $k \geq 2$, $|\mathcal{U}| \geq 2k$, and $\varepsilon_{fp} \leq \frac{1}{2}$, we can bound the failure probability of $\mathcal{D}$ from above:*

$$\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq \begin{cases} \Pr[FPs = 0] & \text{If } p \geq \frac{\varepsilon_{fp} k}{|\mathcal{U}|-k} \\ \left(1 - \frac{p(|\mathcal{U}|-k)}{k}\right)^{k-1} & \text{otherwise.} \end{cases}$$

Proof. If $p \geq \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, then $\mathcal{D} = \mathcal{D}_{\text{FPs}}$ (see (2.6)). By Lemma 7, we have that $\Pr[\text{FPs} = 0] \geq \Pr[\text{FPs} = i]$ for $i = 1, 2, \ldots,$ so (see Lemma 1):

$$\begin{aligned} \Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] &\leq \sum_{i=0}^{k-1} \Pr[\text{FPs} = 0] \cdot \varepsilon_{\text{fp}}^{i} (1 - \varepsilon_{\text{fp}})^{k-1-i} \\ &\quad + \Pr[\text{FPs} = 0] \cdot \varepsilon^{k} \\ &= \Pr[\text{FPs} = 0] \cdot \sum_{i=0}^{k} \varepsilon_{\text{fp}}^{i} (1 - \varepsilon_{\text{fp}})^{k-i}. \end{aligned}$$

Claim: the term $\sum_{i=0}^{k} \varepsilon_{\text{fp}}^{i} (1 - \varepsilon_{\text{fp}})^{k-i}$ is at most 1, which it achieves when $\varepsilon_{\text{fp}} = 0$. This proves the first part of the lemma. To prove this claim, notice that this term effectively models a sum of sequences of k Bernoulli trials. Specifically, the sum of probabilities that the first i trials succeed with probability ε_{fp} each, and the next $k - i$ trials fail with probability $1 - \varepsilon_{\text{fp}}$ (ignoring the cases where trials succeed and fail in a different pattern). This is a well-defined probability distribution, so the term does not exceed 1.

In the other case, when $p < \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, $\mathcal{D} = \mathcal{D}_{\text{TNs}}$. By $\varepsilon_{\text{fp}} \leq (1 - \varepsilon_{\text{fp}})$, we have that $(1 - \varepsilon_{\text{fp}})^{k-1-i} \varepsilon_{\text{fp}}^{i} \leq (1 - \varepsilon_{\text{fp}})^{k-1}$ for $i = 0, 1, \ldots, k-1$, so (see Lemma 2):

$$\begin{aligned} \Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] &\leq \sum_{i=0}^{k-1} \Pr[\text{TNs} = i] \cdot (1 - \varepsilon_{\text{fp}})^{k-1} \\ &\quad + \Pr[\text{TNs} \geq k] \cdot (1 - \varepsilon_{\text{fp}})^{k-1} \\ &= (1 - \varepsilon_{\text{fp}})^{k-1}. \end{aligned}$$

Recall that $p < \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, so ε_{fp} is bounded from below:

$$\varepsilon_{\text{fp}} > \frac{p(|\mathcal{U}| - k)}{k}.$$

The largest value that $\Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}]$ can take occurs when ε_{fp} is as small as possible. This proves the second part of the lemma. □

In the first case, when $p \geq \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, the attack's failure rate is bounded by the probability that the distinguisher cannot find any false positives in $\mathcal{U}/X_2$. This is the same probability with which a distinguisher could tell apart Π_{BF} from an exact

MPSI $\mathcal{F}_{\text{MPSI}}$. In other words, in this scenario, Bloom filters are not suitable when they approximate the intersection. In the other case, notice that:

$$\left(1 - \frac{p(|\mathcal{U}| - k)}{k}\right)^{k-1} \approx e^{-p(|\mathcal{U}|-k)} \leq 2^{-p(|\mathcal{U}|-k)} \ . \tag{2.7}$$

So, for the attack to succeed with negligible probability (i.e., the attack to fail with high probability), the exponent $-p(|\mathcal{U}| - k)$ must remain a small negative number. For example, for the attack to fail with overwhelming probability, we must have that $p \ll (|\mathcal{U}| - k)^{-1}$.

Our second lemma considers the case where p is neither very large nor small. In this case, the attack succeeds with high probability.

Lemma 4. *If $\frac{1}{|\mathcal{U}|-k} \leq p \leq \frac{k-1}{|\mathcal{U}|-k}$, $k \geq 2$, $|\mathcal{U}| \geq 2k$, and $\varepsilon_{fp} \leq \frac{1}{2}$, we can bound the failure probability of $\mathcal{D}$ from above:*

$$\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq 62.5\% \ .$$

Proof. If $p \geq \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, then $\mathcal{D} = \mathcal{D}_{\text{FPs}}$ (see (2.6)). Since $0 \leq \varepsilon_{\text{fp}} \leq 0.5$, we have that $\varepsilon_{\text{fp}}(1 - \varepsilon_{\text{fp}})^{k-1} \geq \varepsilon_{\text{fp}}^{i}(1 - \varepsilon_{\text{fp}})^{k-i}$ for $i = 1, 2, \ldots, k$, so (see Lemma 1):

$$\Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] \leq \Pr[\text{FPs} = 0] + (1 - \Pr[\text{FPs} = 0]) \cdot \varepsilon_{\text{fp}}(1 - \varepsilon_{\text{fp}})^{k-1} \ .$$

Next, we show that the supremum of $\varepsilon_{\text{fp}}(1 - \varepsilon_{\text{fp}})^{k-1}$ occurs when $\varepsilon_{\text{fp}} = k^{-1}$, by checking when its derivative equals 0:

$$\begin{aligned} 0 &= (1 - \varepsilon_{\text{fp}})^{k-1} - (k-1)\varepsilon_{\text{fp}}(1 - \varepsilon_{\text{fp}})^{k-2} \ , \\ &= (1 - \varepsilon_{\text{fp}})^{k-2}((1 - \varepsilon_{\text{fp}}) - (k-1)\varepsilon_{\text{fp}}) \ , \\ &= (1 - \varepsilon_{\text{fp}})^{k-2}(1 - k\varepsilon_{\text{fp}}) \ . \end{aligned}$$

The only valid root occurs when the rightmost term is 0; i.e., $\varepsilon_{\text{fp}} = k^{-1}$. We ignore the bound $p \geq \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, making our final upper bound looser. We get that:

$$\begin{aligned} \Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] &\leq \Pr[\text{FPs} = 0] \\ &\quad + (1 - \Pr[\text{FPs} = 0])\frac{1}{k}\left(1 - \frac{1}{k}\right)^{k-1} \\ &\leq 0.5 + 0.5 \cdot \frac{1}{k}\left(1 - \frac{1}{k}\right)^{k-1} \\ &\leq 0.5 + 0.5 \cdot 0.5(0.5)^{1} = 62.5\% \ . \end{aligned}$$

This works because $\Pr[\text{FPs} = 0] < 0.5$ due to Lemma 7, and that $k^{-1}(1 - k^{-1})^{k-1}$ is monotonically decreasing for $k = 2, 3, \ldots$ (we do not prove this), so we fill in $k = 2$. This concludes the proof for $\mathcal{D} = \mathcal{D}_{\text{FPs}}$.

In the other case, when $p < \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, $\mathcal{D} = \mathcal{D}_{\text{TNs}}$. We get a similar situation:

$$\Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] \leq (1 - \Pr[\text{TNs} \geq k])(1 - \varepsilon_{\text{fp}})^{k-1} \varepsilon_{\text{fp}} + \Pr[\text{TNs} \geq k],$$

where $\Pr[\text{TNs} \geq k] < 0.5$ due to Lemma 8, so we obtain the same bound. □

Our final lemma relating to these upper bounds is for the case where p is large, such that $\Pr[\text{FPs} \geq k]$ is the most likely (and the same holds for $\Pr[\text{TNs} = 0]$). In this case, we do not obtain different bounds depending on $\mathcal{D}$; the only possible case is $\mathcal{D} = \mathcal{D}_{\text{FPs}}$.

Lemma 5. *If $p > \frac{k-1}{|\mathcal{U}|-k}$, $k \geq 2$, $|\mathcal{U}| \geq 2k$, and $\varepsilon_{fp} \leq \frac{1}{2}$, we can bound the failure probability of $\mathcal{D}$ from above:*

$$\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq 2^{-(k-1)} .$$

Proof. If $p(|\mathcal{U}| - k) < \varepsilon_{\text{fp}} k$, then $\mathcal{D} = \mathcal{D}_{\text{TNs}}$ (see (2.6)). However, we have that:

$$p < \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}| - k} \leq \frac{k-1}{|\mathcal{U}| - k} < p ,$$

which is a contradiction, so $\mathcal{D} = \mathcal{D}_{\text{FPs}}$.

When $\mathcal{D} = \mathcal{D}_{\text{FPs}}$ we claim that an upper bound is $2^{-(k-1)}$ thus we will show that

$$\sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^i (1 - \varepsilon_{\text{fp}})^{k-1-i} + \Pr[\text{FPs} \geq k] \cdot \varepsilon_{\text{fp}}^k \leq 2^{-(k-1)}.$$

Note that $\varepsilon_{\text{fp}} \leq \frac{1}{2}$, thus the LHS is smaller than $\sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^i (1 - \varepsilon_{\text{fp}})^{k-1-i} + \Pr[\text{FPs} \geq k] 2^{-k}$, which allows us to instead prove the following inequality:

$$\sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^i (1 - \varepsilon_{\text{fp}})^{k-1-i} \leq 2^{-(k-1)} (1 - \Pr[\text{FPs} \geq k]) \tag{2.8}$$

$$\leq 2^{-(k-1)} \Pr[\text{FPs} < k]. \tag{2.9}$$

By Lemma 9, the LHS is a monotonic increasing function, so it reaches its maximum at the edge of the domain ($\varepsilon_{\text{fp}} = \frac{1}{2}$). The maximum is given by:

$$\begin{aligned}\sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^i (1 - \varepsilon_{\text{fp}})^{k-1-i} &= \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \left(\frac{1}{2}\right)^{k-1} \\ &= 2^{-(k-1)} \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \\ &= 2^{-(k-1)} \Pr[\text{FPs} < k].\end{aligned}$$

This is equal to the bound in (2.9), so $2^{-(k-1)}$ is indeed an upper bound. □

Our main result is the following theorem, which combines the above lemmas.

Theorem 6. *Given $k \geq 2$, $|\mathcal{U}| \geq 2k$, and $0 \leq \varepsilon_{fp} \leq \frac{1}{2}$, we have the following upper bounds for $\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}]$, as summarized in Figure 2.1:*

- $\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq 2^{-(k-1)}$ *if* $p > \frac{k-1}{|\mathcal{U}|-k}$.
- $\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq 62.5\%$ *if* $\frac{1}{|\mathcal{U}|-k} \leq p \leq \frac{k-1}{|\mathcal{U}|-k}$.
- *Otherwise,* $\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq \Pr[FPs = 0]$ *if* $p \geq \frac{\varepsilon_{fp} k}{|\mathcal{U}|-k}$.
- *Otherwise,* $\Pr[\mathcal{D}(\mathcal{F}_{waMPSI}) = \Pi_{BF}] \leq \left(1 - \frac{p(|\mathcal{U}|-k)}{k}\right)^{k-1}$.

Proof. We refer the reader to the following lemmas:

- If $p > \frac{k-1}{|\mathcal{U}|-k}$, then this holds by Lemma 5.
- If $\frac{1}{|\mathcal{U}|-k} \leq p \leq \frac{k-1}{|\mathcal{U}|-k}$, then this holds by Lemma 4.
- If $p < \frac{1}{|\mathcal{U}|-k}$ and $p \geq \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, then this holds by Lemma 3.
- If $p < \frac{1}{|\mathcal{U}|-k}$ and $p < \frac{\varepsilon_{\text{fp}} k}{|\mathcal{U}|-k}$, this holds by Lemma 3. □

2.5.4 Obtaining secure parameters

While the bounds we derive above provide lower bounds for the success probability of the attack, they may underestimate it. Moreover, they quantify over all values of ε_{fp}, while in practice, one may wish to only choose small values (or there would be many false positives). We now consider how to choose the smallest Bloom filter for which our attack does not work. We formulate this as the following constrained optimization problem, in which we want to realize $\mathcal{F}_{\text{waMPSI}}$ with at most false positive probability $\varepsilon^*_{\text{fp}}$:

$$\begin{aligned} \max \quad p \quad \text{s.t.} \quad & \mathrm{Adv}^{\Pi_{\text{BF}}}_{\text{ind}}(\mathcal{D}) \leq 2^{-\lambda} \\ & 0 \leq \varepsilon_{\text{fp}} \leq \varepsilon^*_{\text{fp}} \\ & p < \tfrac{1}{|\mathcal{U}|-k} \end{aligned} \tag{2.10}$$

After all, the larger the false positive probability p, the smaller the Bloom filter can be. Note that the last constraint is implied by the first constraint when $k > \lambda$. We assume this to be the case because λ, the statistical security parameter that decides the chance of the attack succeeding, is typically a value such as 40 or 128, whereas the number of elements in a set k can be orders of magnitude higher. We note that the constraints imply that $\varepsilon^*_{\text{fp}} \leq \frac{1}{k}$.

Since we only look for $p < \frac{1}{|\mathcal{U}|-k}$, and since $\mathcal{D}$ only misclassifies $\mathcal{F}_{\text{waMPSI}}$, we get that:

$$\begin{aligned} \mathrm{Adv}^{\Pi_{\text{BF}}}_{\text{ind}}(\mathcal{D}) &= \left|1 - \frac{1}{2}\right| + \left|1 - \Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] - \frac{1}{2}\right| \\ &= 1 - \Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] \,. \end{aligned}$$

Table 2.1: Parameters for $\mathcal{F}_{\text{waMPSI}}$ with $\varepsilon_{\text{fp}} \leq k^{-1}$ and $|\mathcal{U}| = 2^{32}$. Secure parameters are larger by an order of magnitude.

Setting		Old parameters			$\mathcal{F}_{\text{waMPSI}}$ parameters			
λ	k	p	h	m	p	h	m	Factor
40	256	2^{-6}	6	2,962	2^{-72}	72	26,645	9×
	4096	2^{-8}	8	70,922	2^{-72}	72	425,522	6×
	65536	2^{-12}	12	1,512,788	2^{-72}	72	6,807,543	4.5×
80	256	2^{-6}	6	2,962	2^{-112}	112	41,447	13×
	4096	2^{-8}	8	70,922	2^{-112}	112	661,922	9.33×
	65536	2^{-12}	12	1,512,788	2^{-112}	112	10,589,510	7×
128	256	2^{-6}	6	2,962	2^{-160}	160	59,210	20×
	4096	2^{-8}	8	70,922	2^{-160}	160	945,602	13.33×
	65536	2^{-12}	12	1,512,788	2^{-160}	160	15,127,871	10×

This uses that $\Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}] \leq \frac{1}{2}$.

Instead of using the bounds described in Figure 2.1, we will use the equations from Lemmas 1 & 2 as not to underestimate the distinguisher. That said, we use a heuristic to decide for which ε_{fp} we have that $\Pr[\mathcal{D}(\mathcal{F}_{\text{waMPSI}}) = \Pi_{\text{BF}}]$ is maximal. Specifically, when $\mathcal{D} = \mathcal{D}_{\text{FPs}}$ and p tends to 0, most of the probability mass occurs at $\Pr[\text{FPs} = 0]$ (see Lemma 7). It is easy to see that $\varepsilon_{\text{fp}} = 0$ maximizes the failure probability. For the case when $\mathcal{D} = \mathcal{D}_{\text{TNs}}$, most of the probability mass occurs at $\Pr[\text{TNs} \geq k]$, so we also want to minimize ε_{fp}. However, when $\varepsilon_{\text{fp}} = 0$, we always get that $\mathcal{D} = \mathcal{D}_{\text{FPs}}$. We get that:

$$\text{Adv}_{\text{ind}}^{\Pi_{\text{BF}}}(\mathcal{D}) \approx 1 - \Pr[\text{FPs} = 0] = 1 - (1-p)^{|\mathcal{U}|-k} \,. \tag{2.11}$$

This corresponds to the same scenario as exact MPSI ($\mathcal{F}_{\text{MPSI}}$), in which any false positive occurring is enough for the distinguisher to succeed. Notice that it is not enough for the probability of a false positive to occur in a set of k elements to be negligible; this would only defend against semi-honest parties. In the augmented semi-honest model and beyond, in which corrupt parties can choose their own inputs, the false positive probability must decrease when the gap between $|\mathcal{U}|$ and k increases.

Finally, given the largest p, we can generate Bloom filter parameters m and h as described in Section 2.2. We provide examples of these parameters in Table 2.1 corresponding to $|\mathcal{U}| = 2^{32}$, and compare them against old parameters, as used by previous work. We generate these parameters by iterating over $p = 2^{-1}, 2^{-2}, \ldots$ until the constraints from (2.10) hold, using the approximation from (2.11). For the old parameters we use $p = \frac{1}{k}$, because this would be expected to realize $\mathcal{F}_{\text{waMPSI}}$ with $\varepsilon_{\text{fp}} = \frac{1}{k}$. Note that the parameters we propose only protect against this distinguisher, but stronger attacks may exist.

2.5.5 A note on PSI-cardinality

While the distinguisher $\mathcal{D}$ examines the set output by the protocol Π to determine if it is interacting with $\mathcal{F}_{\text{waMPSI}}$ or Π_{BF}, we note that it is also possible to base this choice solely on the cardinality of the output. As such, this slightly weaker distinguisher $\mathcal{D}'$ would also apply to PSI-cardinality that protocols, which only output the size of the resulting set $|R|$. The condition at which $\mathcal{D}'$ classifies Π as Π_{BF} is when $|R| \neq \text{FPs}$. If $\varepsilon_{\text{fp}} \approx 0$, then $\text{Adv}^{\Pi_{\text{BF}}}_{\text{ind}}(\mathcal{D}') \approx \text{Adv}^{\Pi_{\text{BF}}}_{\text{ind}}(\mathcal{D})$. Since the weakness originates in the Bloom filter, this security problem also affects quantum PSI-cardinality protocols based on Bloom filters [Liu+21].

2.6 Practical attack on Bloom filter-based PSI

In this section, we study membership inference attacks performed by a corrupted leader in the augmented semi-honest model. We present a practical attack on the parameters used by many works [Bay+22; VCE22; Deb+21] discussed in Section 2.4.

2.6.1 Security game

An intuitive definition of a private set intersection's security property is that a leader should only learn about another party's elements when they appear in the intersection. In other words, a leader cannot infer information about the elements that do not appear in the intersection. We formalize this using the concept of membership inference attacks, in which the corrupted leader (the adversary) must guess an element in the other party's set. For simplicity, we only consider two parties, but the concept extends beyond the two-party setting.

To incorporate the fact that the leader learns the result of the protocol, we model membership inference against PSI as an adaptive security game in which the adversary consists of two algorithms: $\mathcal{A}_{\text{pre}}$ and $\mathcal{A}_{\text{post}}$. The first algorithm inputs $\mathcal{U}_2$, which is a superset of the victim's set X_2, for which it holds that $|X_2| = k$. $\mathcal{A}_{\text{pre}}$ outputs the leaders input to the protocol Π and an element $t \in \mathcal{U}_2$. The second algorithm $\mathcal{A}_{\text{post}}$ inputs the element t and the result of the protocol, and outputs the guess of the adversary. We assume the adversary already has access to all public parameters (for us, this includes the Bloom filter's hash functions). We define an adversary's probability of beating the membership inference security game against a PSI protocol Π as follows:

$$\Pr[\mathcal{A} \text{ succeeds}|u] = \Pr\left[\begin{array}{l} \xrightarrow{(\text{inp},X_1)} \boxed{\Pi} \xleftarrow{(\text{inp},X_2)} \\ \xleftarrow{(R)} \\ \mathcal{A}_{\text{post}}(t,R) = b \end{array} \middle| \begin{array}{l} \mathcal{U}_2 \subseteq_R \mathcal{U} \text{ s.t. } |\mathcal{U}_2| = u \\ (X_1, t) \leftarrow \mathcal{A}_{\text{pre}}(\mathcal{U}_2) \\ \quad \text{s.t. } (t \in \mathcal{U}_2) \wedge (t \notin X_1) \\ b \in_R \{0,1\} \\ X_2 \subseteq_R \mathcal{U}_2 \text{ s.t. } (|X_2| = k) \wedge \\ \quad (t \in X_2 \iff b = 1) \end{array} \right] \tag{2.12}$$

The advantage of an adversary $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{post}})$ over random guessing is:

$$\text{Adv}^{\Pi}_{\text{memb}}(\mathcal{A}, |\mathcal{U}_2|) = 2\left|\Pr[\mathcal{A} \text{ succeeds} \mid u = |\mathcal{U}_2|] - \frac{1}{2}\right| \tag{2.13}$$

If one can show that the advantage is negligible for all adversaries, then the protocol is secure against membership inference attacks. In the next subsections, we present attacks in which the advantage is non-negligible when the false positive probability is non-negligible.

2.6.2 Proposed attack

The foundation of our proposed attack lies in two key observations: First, when a Bloom filter's hash functions are known, we may determine which elements in $\mathcal{U}$ have overlapping bins when encoded in a Bloom filter. Second, for Bloom filter encodings of most subsets of $\mathcal{U}$, there are bins that are set only by a single element of this subset. We first expand on both observations.

We use the first observation to find probabilities of Bloom filter false positives. Given a set X, and its Bloom filter encoding, $\hat{X}$, we observe that knowledge of the hash functions collapses the false positive probability. Let us denote $\mathsf{contains}(\hat{X}, x)$ by $x \in \hat{X}$. For any $y \notin X$, we verify whether y is a false positive in $\hat{X}$ by calculating $H_i(y)$, for $i \in 1, \ldots, h$, and verifying that each bin is set. If we know X, the conditional probabilities $\Pr[y \in \hat{X} \mid X]$ become deterministic. If we do not know X, yet instead, we know the distribution of X sampled from $\mathcal{U}$, we find $\Pr[y \in \hat{X}] = \sum_{X, y \in \hat{X}} \Pr[X]$.

The second observation helps detect the target's presence in the Bloom filter encoding. Some bins may *uniquely identify* an element x amongst a subset $\mathcal{U}_2$ of the total universe. In other words, a bin b may exist such that only one element $x \in \mathcal{U}_2$ maps to this bin with any hash function. From Section 2.2, recall that one chooses the parameters of a Bloom filter such that, for k elements, the probability of any element having no unique bin is at most p. However, an element y not present in $\mathcal{U}_2$ may hash to this bin. When y is a false positive in a Bloom filter encoding of a subset of $\mathcal{U}_2$, we know that this subset contained x. The probability of finding any element with a uniquely identifying bin depends on both $\mathcal{U}_2$ and k.

We define the adversary's universe as $\mathcal{U}_1 = \mathcal{U} \setminus \mathcal{U}_2$. We realize A_{pre} by means of the following steps:

1. We find an optimal target $t \in \mathcal{U}_2$ which can be uniquely identified amongst the other elements of $\mathcal{U}_2$,

2. we reduce $\mathcal{U}_1$ to C, leaving only elements that hash to the bins that are unique to t and which have a non-zero probability of occurring as a false positive,

3. we rank all elements $y \in C$ based on the probability of appearing in the Bloom filter encoding of X_2,

4. we return t and the top k elements of C.

To determine the bit b, by testing the result of the protocol execution R. If R is non-empty, we successfully identified t and $\mathcal{A}_{\text{post}}$ returns 1. Otherwise, $\mathcal{A}_{\text{post}}$ guesses that t is not part of X_2 and returns 0.

Choosing the target

As the target t, we select an element from $\mathcal{U}_2$ with the least overlap with other bins of other elements in $\mathcal{U}_2$. Bins that are unique to the target t can be used to conclusively decide whether the target is included in the input set X_2 of the victim $\mathcal{P}_2$. We find an optimal target $t \in \mathcal{U}_2$ using an exhaustive search:

$$t = \min_{x \in \mathcal{U}_2} |\{x' \in \mathcal{U}_2 \setminus \{x\} \mid \exists (i,j) : H_i(x) = H_j(x')\}|. \tag{2.14}$$

Filtering the attack universe

Given a target t the adversary $\mathcal{P}_1$ searches for an attack input set X_1 that maximizes the detection rate of the target element t. We can filter any element from $\mathcal{U}_1$, which does not contribute to detecting t. We use the following two criteria to narrow down $\mathcal{U}_1$ to the candidate subset C: First, for any candidate $y \in C$, at least one hash function must exist such that y is mapped to a bin that *uniquely identifies* t from the other elements in $\mathcal{U}_2$. Second, for each hash function H, H maps y to a bin that at least one element in $\mathcal{U}_2$ hashes to.

Selecting the attack set

If for the number of candidates C it holds that $|C| \leq k$, we set $X_1 = C$. Otherwise, we aim to find the input set $X_1 \subset C$ with $|X_1| = k$ that is most likely to successfully identify t. Calculating this 'effectiveness' of an input set X_1 requires that we assume knowledge of the distribution of $X_2 \subseteq \mathcal{U}_2$. Ideally, we would maximize the effectiveness by maximizing the probability that at least one of the elements in X_1 is contained in the Bloom filter $\hat{X}_2$. Instead of carefully computing an inclusion-exclusion, we approximate this probability by simply summing over the probability with which each element in X_1 is contained in $\hat{X}_2$:

$$\Pr[\exists y \in X_1 \mid y \in \hat{X}_2] \approx \sum_{y \in X_1} \Pr[y \in \hat{X}_2]. \tag{2.15}$$

In other words, our objective is to maximize the probability with which the intersection obtained by the leader is non-empty. For the remaining part of this section, we assume that $\mathcal{P}_2$ uniformly samples its private input set X_2 from $\mathcal{U}_2$. This aligns with our definition of $\mathrm{Adv}^{\Pi_{\mathrm{BF}}}_{\mathrm{memb}}$. The probability $\Pr[y \in \hat{X}_2]$ equals the number of private input sets X_2 that activate all bins of y, divided by the number of different input sets. We rank all elements in C based on the number of X_2 for which $y \in \hat{X}_2$. Afterward, we select the k elements with the highest probability of appearing in the Bloom filter as the attack set.

Finding the number of X_2 for which $y \in \hat{X}_2$

To find the number of X_2 for which $y \in \hat{X}_2$ holds, we focus on finding a formula for the number of possible input sets X_2 that activate the bins of y. We can construct

lists $b_1, .., b_h$, where each list b_i contains the elements of $\mathcal{U}_2$ that hashes to bin $H_i(y)$, in other words:

$$b_i = \{x \in \mathcal{U}_2 \mid \exists j \in 1..h H_j(x) = H_i(y)\}.$$

We distinguish two cases: In the first case, all elements from all these lists b_i are distinct. In the second case, one or more elements appear in at least two bins. We focus on the first case and later show how the second case can be reduced to the first.

All elements are distinct. For the first case, we emulate picking elements from $\mathcal{U}_2$ to create X_2. Let $\mathcal{U}'$ denote the universe we are picking from, and $|\mathcal{U}'|$ the number of elements in this universe. The goal is to fill the k' spots of the private input set with elements from $\mathcal{U}'$ with the following restrictions:

- None of the elements can appear more than once.
- The order of the elements does not matter, i.e. "1 2" and "2 1" are the same.
- There are r lists $b_1, .., b_r$ each consisting of a number of elements. These lists are all pairwise distinct, i.e., for each $i, j \leq r$ holds that $b_i \cap b_j = \emptyset$. From each of these lists, at least one element should be taken.

Example 8. Take $\mathcal{U}' = \{1..6\}$, $b_1 = \{1, 2\}$, $b_3 = \{3\}$ and $k' = 3$. The order does not matter. Therefore, we choose to first write the element of bin 1, then the element of bin 2, and then fill the last spot.

1 3 2	~~2 3 1~~
1 3 4	2 3 4
1 3 5	2 3 5
1 3 6	2 3 6

See that 1 3 2 is equivalent to 2 3 1 and should not be counted. There are 7 options.

Example 8 shows that the elements we can choose for the last bin must not occur in the sequence yet, and neither must create a sequence that has already been counted. We solve this by creating $r + 1$ different lists. We use b_0 to denote $\mathcal{U}' \setminus (b_1 \cup b_2 \cup \cdots \cup b_r)$; The list that contains all elements that occur in no other. This simplifies the problem to dividing k' choices over $r + 1$ lists, where for b_i $i = 1 \ldots r$ at least one element must be chosen. We calculate the answer to this problem by expressing it as a product of the generating function, and then finding the coefficient of the term with degree k'.

For each list b_i we find a corresponding generating function that expresses in how many ways we can choose elements from that list. The exponent of the variable z denotes the number of elements chosen from the list b_i. The coefficient of z^j tells us in how many ways j elements can be chosen from b_i. Since we pick without replacement, choosing j elements from b_i can be done in $\binom{|b_i|}{j}$ ways. Given that we may pick no elements from b_0, we find the following functions for b_0 and b_i with $i \neq 0$ respectively:

$$f(|b_0|, z) = \sum_{j=0}^{|b_0|} \binom{|b_0|}{j} z^j, \quad g(|b_i|, z) = \sum_{j=1}^{|b_i|} \binom{|b_i|}{j} z^j.$$

The combined formula allows us to determine the number of ways to pick k' elements for X_2 such that $y \in \hat{X}_2$, by calculating the coefficient of the term $z^{k'}$. The formula for the number of combinations is as follows:

$$[z^{k'}]\colon f\left(|b_0|, z\right) \cdot \prod_{i=1}^{r} g\left(|b_i|, z\right). \tag{2.16}$$

Not all elements are distinct. If an element $x \in \mathcal{U}_2$ activates two bins of the Bloom filter that the element y hashes to, we lose the independence requirement and, as such, Eq. (2.16) does not hold. We observe that given $\mathcal{U}', k', \{b_1, \ldots, b_r\}$, the number of valid combinations is equal to the number of valid combinations with $x \in \mathcal{U}'$ plus the number of valid combinations without x. We can define the number of valid combinations with x as the number of combinations of universe $\mathcal{U}' \setminus \{x\}$, spots $k' - 1$, and $\{b_i \mid x \notin b_i\}$. Likewise the number of valid combinations where x is not included is the number of combinations of universe $\mathcal{U}' \setminus \{x\}$, spots k', and $\{b_1 \setminus \{x\}, \ldots, b_r \setminus \{x\}\}$. By reducing $\mathcal{U}_2$, k, and the lists to a summation of Eq. (2.16) with different arguments, we find an exact expression of the number of X_2 for which $y \in \hat{X}_2$ holds.

2.6.3 Results

To evaluate our attack, we implement it in Rust.[3] The experiments are run on an AMD Ryzen Threadripper 7970X 32-core CPU at around 4.6 GHz. We use constants hash seeds for each experiment and search an attack input set X_1 and target t once. Afterward, we determine the advantage by uniformly sampling $X_2 \subset \mathcal{U}_2$ 1 million times with $t \in X_2$ and 1 million times with $t \notin X_2$. We limit the time for searching an attack input set to two hours. The results are shown in Table 2.2. We indicate executions of our attack that took over two hours to compute with '-'.

2.7 Mitigations

Our theoretical analysis and practical attack require protocol designers to choose Bloom filter parameters that can easily be an order of magnitude larger, as shown in Table 2.1. This may be acceptable in protocols that use oblivious transfers [Ben+22], but this may significantly decrease the efficiency of protocols based on homomorphic encryption [Bay+22; VCE22; Deb+21]. Fortunately, one may still design Bloom filter-based protocols that prevent these attacks in other ways, without having to increase the size of the Bloom filter significantly. In this section, we provide suggestions as to such mitigations.

2.7.1 Replacing hash functions with OPRFs

In all protocols discussed in Section 2.4, the parties have access to the hash functions of the Bloom filter. However, this also allows corrupted parties to identify elements

[3] The source code is available at: `Redacted for reviews.`

Table 2.2: Overview of the results of our practical attack applied to different false positive rates p and set sizes k. We evaluate the attack for $|\mathcal{U}_2| = 2k, 3k, 4k$. The attack succeeds with non-negligible probability, even for $p = 2^{-20}$.

Setting		Parameters		Results		
p	k	h	m_{opt}	$\text{Adv}^{\Pi}_{\text{memb}}(\mathcal{A}, 2k)$	$\text{Adv}^{\Pi}_{\text{memb}}(\mathcal{A}, 3k)$	$\text{Adv}^{\Pi}_{\text{memb}}(\mathcal{A}, 4k)$
2^{-5}	256	5	1,852	1.00	1.00	1.00
2^{-10}	256	10	3,702	1.00	0.82	0.61
	4,096	10	59,102	1.00	-	-
	65,536	10	945,493	1.00	-	-
2^{-20}	256	20	7,403	0.15	0.01	$2^{-8.9}$
	4,096	20	118,202	0.98	-	-
	65,536	20	1,890,985	0.68	-	-
2^{-30}	256	30	11,103	0.03	2^{-15}	-
	4,096	30	177,302	0.02	-	-
	65,536	30	2,836,477	2^{-10}	-	-

they can exploit in an attack, as shown in Section 2.6. Instead, one could replace the hash functions with oblivious pseudo-random functions (OPRFs) [CHL22] with a secret seed. This would limit the parties in the number of queries they can make to the hash functions. Specifically, we can limit each party to k calls to each OPRF, which may be significantly smaller than the number of calls $|\mathcal{U}| - k$ that we use in our attack.

Designing a protocol around OPRFs requires answering the question of which parties can know the secret seed. We suggest to assign t parties who choose their own secret seed. Each party then engages in an OPRF protocol with each of these t parties to generate t pseudo-random values, from which one can derive a single pseudo-random value for which one can only find the preimage if the t parties collude. This approach would add two rounds to the total protocol's execution because each party must first run kt parallel OPRF evaluations before they can construct their Bloom filter.

2.7.2 Replacing hash functions with PBKDFs

Another approach is to replace the hash functions with extremely slow hash functions, such as password-based key derivation functions [Kal00]. We note that this approach does not prevent polynomial-time attackers because the evaluation of the hash functions only increases with a polynomial factor. However, in practice, it would be infeasible (or at least extremely expensive) for an attacker to make $|\mathcal{U}| - k$ queries to the PBKDF.

While this approach offers a simple patch to existing Bloom filter-based PSI protocols, it is in stark contrast with the common choice of choosing extremely fast statistical hash functions without cryptographic guarantees. As a result, the protocol is also significantly slower to execute for honest parties. Depending on the

number of elements each party encodes in their Bloom filter, choosing larger Bloom filter parameters may be cheaper. Next to that, it may be hard to parameterize the PBKDF because one would have to estimate the concrete cost of the PBKDF and the computational abilities of an attacker. One solution may be to balance the cost of the hash function evaluation with larger Bloom filter parameters to find a set of parameters that punish attackers without significantly slowing down the protocol for honest parties.

2.7.3 Authorized private set intersections

If the parties running a Bloom filter-based PSI protocol trust a semi-honest third party, this party may authorize their sets, preventing the elements in these sets from being selected maliciously. An example of such a protocol is by Kerschbaum [Ker12], who designed an authorized PSI protocol based on Bloom filters, in which a judge prevents the client from picking elements that are likely to cause a false positive in the Bloom filter of the server. To prevent a client from doing so, the elements of the server X_2 are encrypted before they are added to the Bloom filter. This encryption ensures that the client does not have prior knowledge of the server's Bloom filter. In order to perform the intersection, however, the client's elements should be encrypted with the same key.

In practice, the judge takes the elements of the client $\mathcal{P}_1$ and raises them to some secret power e. These elements are then stored in a Bloom filter by the judge. The Bloom filter is encrypted using a public key and signed, after which the encryption and signature are returned to the client. The client forwards the Bloom filter and the signed Bloom filter to the server, which then can verify that the judge signed this Bloom filter. Only the non-signed filter is used for the remainder of the protocol.

Applying authorization to the sets may not be possible in practice because the parties may not have a semi-honest third party they trust. Additionally, a judge that inspects (a subset of) the private input sets also learns private elements from the honest parties' sets. In any case, it adds at least two rounds to the execution of the protocol, as the clients have to submit their sets, and the judge has to approve them.

2.8 Conclusion

In this work, we propose both theoretical and practical attacks against Bloom filter-based private set intersection protocols. We show that secure parameters must be an order of magnitude larger than parameters where the false positive probability is not negligible. As a result, Bloom filter-based PSI cannot use the approximation provided by Bloom filters to speed up the protocol. Alternatively, one might consider consider replacing the hash functions with OPRFs or PBKDFs, but both approaches cause the protocol to become slower.

With these results, we are not aware of any approximate (M)PSI protocols that outperform exact (M)PSI protocols. As such, an open question is whether there are efficient alternatives for Bloom filter-based approximate (M)PSI protocols. Other

future work may look at the following questions:

- Is it possible to extend our attack to protocols like that by Zhu et al. [Zhu+18] and those based on garbled Bloom filters that go beyond regular Bloom filters?
- How are other deterministic approximate data structures such as approximate membership query filters affected by these results?

On a positive note, we notice that Bloom filters still offer useful characteristics for designing (M)PSI protocols, even when their parameters must be large. Specifically, it is still convenient to compute a Bloom filter representing the intersection. For example, the work by Ben Efraim et al. [Ben+22] uses parameters that are significantly larger than the parameters suggested in our work (our work considers the augmented semi-honest model while theirs considers the malicious model), but it is still concretely efficient.

References

[Bay+22] Aslí Bay et al. "Practical Multi-Party Private Set Intersection Protocols". In: *IEEE Trans. Inf. Forensics Secur.* 17 (2022), pp. 1–15. DOI: 10.1109/TIFS.2021.3118879. URL: https://doi.org/10.1109/TIFS.2021.3118879.

[Ben+22] Aner Ben-Efraim et al. "PSImple: Practical Multiparty Maliciously-Secure Private Set Intersection". In: *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*. Ed. by Yuji Suga et al. ACM, 2022, pp. 1098–1112. DOI: 10.1145/3488932.3523254. URL: https://doi.org/10.1145/3488932.3523254.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888. URL: https://doi.org/10.1109/SFCS.2001.959888.

[CHL22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. "SoK: Oblivious Pseudorandom Functions". In: *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 2022, pp. 625–646. DOI: 10.1109/EUROSP53844.2022.00045. URL: https://doi.org/10.1109/EuroSP53844.2022.00045.

[DC17] Alex Davidson and Carlos Cid. "An efficient toolkit for computing private set operations". In: *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part II 22*. Springer. 2017, pp. 261–278.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 789–800. DOI: 10.1145/2508859.2516701. URL: https://doi.org/10.1145/2508859.2516701.

[DD15] Sumit Kumar Debnath and Ratna Dutta. "Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter". In: *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*. Ed. by Javier López and Chris J. Mitchell. Vol. 9290. Lecture Notes in Computer Science. Springer, 2015, pp. 209–226. DOI: 10.1007/978-3-319-23318-5_12. URL: https://doi.org/10.1007/978-3-319-23318-5%5C_12.

[Deb+21] Sumit Kumar Debnath et al. "Secure and efficient multiparty private set intersection cardinality". In: *Adv. Math. Commun.* 15.2 (2021), pp. 365–386. DOI: 10.3934/amc.2020071. URL: https://doi.org/10.3934/amc.2020071.

[Ege+15] Rolf Egert et al. "Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters". In: *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*. Ed. by Ernest Foo and Douglas Stebila. Vol. 9144. Lecture Notes in Computer Science. Springer, 2015, pp. 413–430. DOI: 10.1007/978-3-319-19962-7_24. URL: https://doi.org/10.1007/978-3-319-19962-7%5C_24.

[GG10] Ashish Goel and Pankaj Gupta. "Small subset queries and bloom filters using ternary associative memories, with applications". In: *SIGMETRICS 2010, Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, New York, USA, 14-18 June 2010*. Ed. by Vishal Misra, Paul Barford, and Mark S. Squillante. ACM, 2010, pp. 143–154. DOI: 10.1145/1811039.1811056. URL: https://doi.org/10.1145/1811039.1811056.

[HL10] Carmit Hazay and Yehuda Lindell. "A Note on the Relation between the Definitions of Security for Semi-Honest and Malicious Adversaries". In: *IACR Cryptol. ePrint Arch.* (2010), p. 551. URL: http://eprint.iacr.org/2010/551.

[Kal00] Burt Kaliski. *RFC2898: Pkcs# 5: Password-based cryptography specification version 2.0*. 2000.

[Ker12] Florian Kerschbaum. "Outsourced private set intersection using homomorphic encryption". In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '12. Seoul, Korea: Association for Computing Machinery, 2012, pp. 85–86. ISBN: 9781450316484. DOI: 10.1145/2414456.2414506. URL: https://doi.org/10.1145/2414456.2414506.

[Liu+21] Bai Liu et al. “Quantum private set intersection cardinality based on bloom filter”. In: *Scientific Reports* 11.1 (Aug. 30, 2021), p. 17332. DOI: 10.1038/s41598-021-96770-1. URL: https://doi.org/10.1038/s41598-021-96770-1.

[LLT24] Keyang Liu, Xingxin Li, and Tsuyoshi Takagi. “Review the Cuckoo Hash-Based Unbalanced Private Set Union: Leakage, Fix, and Optimization”. In: *Computer Security - ESORICS 2024 - 29th European Symposium on Research in Computer Security, Bydgoszcz, Poland, September 16-20, 2024, Proceedings, Part II*. Ed. by Joaquín García-Alfaro et al. Vol. 14983. Lecture Notes in Computer Science. Springer, 2024, pp. 331–352. DOI: 10.1007/978-3-031-70890-9_17. URL: https://doi.org/10.1007/978-3-031-70890-9%5C_17.

[MN15] Atsuko Miyaji and Shohei Nishida. “A Scalable Multiparty Private Set Intersection”. In: *Network and System Security*. Ed. by Meikang Qiu et al. Cham: Springer International Publishing, 2015, pp. 376–385. ISBN: 978-3-319-25645-0.

[MNN17] Atsuko Miyaji, Kazuhisa Nakasho, and Shohei Nishida. “Privacy-Preserving Integration of Medical Data - A Practical Multiparty Private Set Intersection”. In: *J. Medical Syst.* 41.3 (2017), 37:1–37:10. DOI: 10.1007/S10916-016-0657-4. URL: https://doi.org/10.1007/s10916-016-0657-4.

[Qiu+22] Shuo Qiu et al. “SE-PSI: Fog/Cloud server-aided enhanced secure and effective private set intersection on scalable datasets with Bloom Filter”. In: *Mathematical Biosciences and Engineering* 19.2 (2022), pp. 1861–1876. ISSN: 1551-0018. DOI: 10.3934/mbe.2022087. URL: https://www.aimspress.com/article/doi/10.3934/mbe.2022087.

[RA23] Ou Ruan and Chaohao Ai. “An efficient multi-party private set intersection protocols based on bloom filter”. In: *Second International Conference on Algorithms, Microchips, and Network Applications (AMNA 2023)*. Ed. by Kannimuthu Subramaniam and Arunkumar Palanisamy Muthuramalingam. Vol. 12635. International Society for Optics and Photonics. SPIE, 2023, p. 1263518. DOI: 10.1117/12.2678880. URL: https://doi.org/10.1117/12.2678880.

[RR17] Peter Rindal and Mike Rosulek. “Improved Private Set Intersection Against Malicious Adversaries”. In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes in Computer Science. 2017, pp. 235–259. DOI: 10.1007/978-3-319-56620-7_9. URL: https://doi.org/10.1007/978-3-319-56620-7%5C_9.

[Rua+23] Ou Ruan et al. “A Practical Multiparty Private Set Intersection Protocol Based on Bloom Filters for Unbalanced Scenarios”. In: *Applied Sciences* 13.24 (2023). ISSN: 2076-3417. DOI: 10.3390/app132413215. URL: https://www.mdpi.com/2076-3417/13/24/13215.

[VCE22] Jelle Vos, Mauro Conti, and Zekeriya Erkin. "Fast Multi-party Private Set Operations in the Star Topology from Secure ANDs and ORs". In: *IACR Cryptol. ePrint Arch.* (2022), p. 721. URL: https://eprint.iacr.org/2022/721.

[VCE23] Jelle Vos, Mauro Conti, and Zekeriya Erkin. "SoK: Collusion-resistant Multi-party Private Set Intersections in the Semi-honest Model". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1777. URL: https://eprint.iacr.org/2023/1777.

[Zhu+18] Hongliang Zhu et al. "Outsourcing set intersection computation based on bloom filter for privacy preservation in multimedia processing". In: *Security and Communication Networks* 2018.1 (2018), p. 5841967.

2.A Conditions on the false positive probability

Lemma 7. $\Pr[FPs = 0] \geq \frac{1}{2}$ *if and only if* $p < \frac{1}{|\mathcal{U}|-k}$.

Proof. First, consider that the distribution of the number of false positives is binomial $\Pr[\text{FPs}] \sim \mathcal{B}(|\mathcal{U}| - k, p)$, because $|\mathcal{U}/X_2| = |\mathcal{U}| - k$ and p is the false positive probability. The median is given by $\lfloor (|\mathcal{U}| - k)p \rfloor$ or $\lceil (|\mathcal{U}| - k)p \rceil$.

Our lemma holds when the median M is at or below FPs = 0. Since $\lfloor (|\mathcal{U}| - k)p \rfloor \leq \lceil (|\mathcal{U}| - k)p \rceil$, we get that:

$$M \leq 0 \tag{2.17}$$
$$\lfloor (|\mathcal{U}| - k)p \rfloor \leq 0 \tag{2.18}$$
$$(|\mathcal{U}| - k)p < 1 \tag{2.19}$$
$$p < \frac{1}{|\mathcal{U}| - k}$$

□

Lemma 8. $\Pr[FPs = k] \geq \frac{1}{2}$ *if and only if* $p > \frac{k-1}{|\mathcal{U}|-k}$.

Proof. Like in Lemma 7, the distribution of the number of false positives is binomial $\Pr[\text{FPs}] \sim \mathcal{B}(|\mathcal{U}| - k, p)$, and the median is given by $\lfloor (|\mathcal{U}| - k)p \rfloor$ or $\lceil (|\mathcal{U}| - k)p \rceil$.

Our lemma holds when the median M is at or above FPs = k. Since $\lfloor (|\mathcal{U}| - k)p \rfloor \leq \lceil (|\mathcal{U}| - k)p \rceil$, we get that:

$$M \geq k \tag{2.20}$$
$$\lceil (|\mathcal{U}| - k)p \rceil \geq k \tag{2.21}$$
$$(|\mathcal{U}| - k)p > k - 1 \tag{2.22}$$
$$p > \frac{k-1}{|\mathcal{U}| - k}$$

□

2.B Additional lemmas for proving upper bounds

Lemma 9. *Given $p > \frac{k-1}{|\mathcal{U}|-k}$ and $0 \leq \varepsilon_{fp} \leq \frac{1}{2}$, the following is monotonically increasing with ε_{fp}:*

$$\sum_{i=0}^{k-1} \Pr[FPs = i] \varepsilon_{fp}^{i} (1-\varepsilon_{fp})^{k-1-i}$$

Proof. We show that the derivative is non-negative for the entire interval $\varepsilon_{\text{fp}} \in [0, \frac{1}{2})$:

$$0 \leq \frac{\delta}{\delta \varepsilon_{\text{fp}}} \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i} (1-\varepsilon_{\text{fp}})^{k-1-i} \ , \tag{2.23}$$

$$= \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \left(i \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-1-i} + \varepsilon_{\text{fp}}^{i} (-i-k+1)(1-\varepsilon_{\text{fp}})^{k-2-i} \right) \ , \tag{2.24}$$

$$= \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left(i(1-\varepsilon_{\text{fp}}) + \varepsilon_{\text{fp}}(i-k+1) \right) \ , \tag{2.25}$$

$$= \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left(i - i \cdot \varepsilon_{\text{fp}} + i \cdot \varepsilon_{\text{fp}} - k \cdot \varepsilon_{\text{fp}} + \varepsilon_{\text{fp}} \right) \ , \tag{2.26}$$

$$= \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left(i + (1-k) \cdot \varepsilon_{\text{fp}} \right) \ , \tag{2.27}$$

$$\geq \sum_{i=0}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left(i + (1-k) \cdot \frac{1}{2} \right) \ . \tag{2.28}$$

At this point, it remains to show that (2.28) is non-negative. Notice that half of the terms in this sum are positive, and half of them are negative. Specifically, it is the rightmost clause that determines the sign: the sign is positive if and only if $i \geq \frac{k-1}{2}$. To ensure that the derivative is always positive, we proceed to show that the magnitude of the positive terms is at least as large as the magnitude of the negative terms. That is:

$$\sum_{i=\lceil \frac{k-1}{2} \rceil}^{k-1} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left(i + (1-k) \cdot \frac{1}{2} \right) \ , \tag{2.29}$$

must be at least as large as:

$$\left| \sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left(i + (1-k) \cdot \frac{1}{2} \right) \right| = \tag{2.30}$$

$$\sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \Pr[\text{FPs} = i] \varepsilon_{\text{fp}}^{i-1} (1-\varepsilon_{\text{fp}})^{k-2-i} \left((k-1) \cdot \frac{1}{2} - i \right) \ . \tag{2.31}$$

It is sufficient to show that each term in (2.31) has a unique dominating term in (2.29). We do so by comparing the ith term in (2.31) with the i'th term in (2.29), where $i' = k - 1 - i$, and showing that the term for i' dominates the term of i:

$$\Pr[\text{FPs} = i']\varepsilon_{\text{fp}}^{i'-1}(1-\varepsilon_{\text{fp}})^{k-2-i'}\left(i' + (1-k)\cdot\frac{1}{2}\right) \geq \tag{2.32}$$

$$\Pr[\text{FPs} = i]\varepsilon_{\text{fp}}^{i-1}(1-\varepsilon_{\text{fp}})^{k-2-i}\left((k-1)\cdot\frac{1}{2} - i\right) . \tag{2.33}$$

We already know that $\Pr[\text{FPs} = i]$ is monotonically increasing with i for $i \in [0, k]$, so it remains to show that:

$$\varepsilon_{\text{fp}}^{i'-1}(1-\varepsilon_{\text{fp}})^{k-2-i'}\left(i' + (1-k)\cdot\frac{1}{2}\right) \geq \varepsilon_{\text{fp}}^{i-1}(1-\varepsilon_{\text{fp}})^{k-2-i}\left((k-1)\cdot\frac{1}{2} - i\right) . \tag{2.34}$$

Next, we have that:

$$\left(i' + (1-k)\cdot\frac{1}{2}\right) = \left((k-1)\cdot\frac{1}{2} - i\right) , \tag{2.35}$$

$$k - 1 - i + \frac{1}{2} - k\cdot\frac{1}{2} = k\cdot\frac{1}{2} - \frac{1}{2} - i , \tag{2.36}$$

$$k - 1 - i = k - 1 - i . \tag{2.37}$$

Finally, we must show that:

$$\varepsilon_{\text{fp}}^{i'-1}(1-\varepsilon_{\text{fp}})^{k-2-i'} \geq \varepsilon_{\text{fp}}^{i-1}(1-\varepsilon_{\text{fp}})^{k-2-i} . \tag{2.38}$$

We give a proof by induction to show that $\varepsilon_{\text{fp}}^{x-1}(1-\varepsilon_{\text{fp}})^{k-2-x}$ decreases monotonically with x. This implies that the LHS of (2.38) is greater than or equal to the RHS because i' is larger than i. Incrementing x by 1 yields:

$$\varepsilon_{\text{fp}}^{x}(1-\varepsilon_{\text{fp}})^{k-3-x} = \varepsilon_{\text{fp}}(1-\varepsilon_{\text{fp}})^{-1}\left[\varepsilon_{\text{fp}}^{x-1}(1-\varepsilon_{\text{fp}})^{k-2-x}\right] . \tag{2.39}$$

This factor $\varepsilon_{\text{fp}}(1-\varepsilon_{\text{fp}})^{-1} \leq 1$, because $\varepsilon_{\text{fp}} \in [0, \frac{1}{2}]$ and $(1-\varepsilon_{\text{fp}})^{-1} \in [1, 2]$.

We note that when k is odd, the sum in (2.28) has $\frac{k-1}{2}$ positive and negative terms (covered by our analysis), while the last term $i = \frac{k-1}{2}$ always equals zero, so we do not need to consider it. This concludes the proof. □

Chapter 3

Fast Multi-party Private Set Operations in the Star Topology from Secure ANDs and ORs

Previous homomorphic encryption-based multi-party private set operation protocols are all instantiated with integer-based homomorphic encryption. In this chapter, we propose low-round homomorphic encryption-based protocols in the star topology that are instantiated over elliptic curves, reducing the computational cost and bandwidth required for these protocols to complete. While the previous chapters focused on set intersections, we also present multi-party private set union protocols in the star topology. In the first part of this chapter, we use the bitset representation to propose efficient intersections and unions when the universe is small.

When the universe is large, bitset-based set operations no longer perform efficiently as the representation grows linearly with the number of elements in the universe. In the second part of this chapter, we put forward alternative ways of performing set intersections and unions in the star topology for large universes. Specifically, we use Bloom filters to perform intersections, and we use a divide-and-conquer approach to exponentially reduce the size of the bitset representation at the cost of introducing more interactions for set unions.

In the previous chapter, we showed that there are security problems with small Bloom filters. This required us to revisit the parameterization of our multi-party private set intersection protocol for large universes when compared to the original version of this work.

This chapter is a revised version of the work with the same title that has been made available on the IACR ePrint Archive in 2022, authored by Jelle Vos, Mauro Conti, and Zekeriya Erkin.

3.1 Introduction

Our increasingly digital society is making a growing amount of data available to computers, networks, and third parties. As a consequence, our sensitive data is in danger of getting exposed. The field of multi-party computation attempts to mitigate this by studying protocols that enable parties to perform their operations digitally without the risk of privacy-violating data leaks. Among those operations are multi-party private set operations. We consider multi-party private

set intersections (MPSI) and unions (MPSU): Consider n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, who each have a set $X_1, \ldots, X_n$, respectively. For MPSIs, the task is to privately compute $X_1 \cap \cdots \cap X_n$. For MPSUs, the parties must compute $X_1 \cup \cdots \cup X_n$. Each set contains at most k elements from a finite universe $\mathcal{U}$, so $|X_i| \leq k$. In our setting, we select a leader who receives the result of the operation, but all parties are allowed to learn it. We refer to the other parties as assistants. We denote $\mathcal{P}_1$ as the leader, without loss of generality.

MPSI and MPSU protocols serve many different applications, as set operations form a fundamental building block in day-to-day functionality. For example, by using a private set intersection on possible dates for a meeting, multiple colleagues can select a meeting date at which they are all available without revealing other information about their calendar. A use case of private set unions is the creation of no-fly lists: Several agencies can prevent passengers from flying, but it would leak information if an agency knew which individuals the other agencies were investigating. The result of a private set union reveals the complete set of banned passengers, but without reference to which or how many agencies are investigating them. Other use cases of MPSIs include confidential data sharing on security incident information and botnet detection [Bay+21]. At the same time, MPSUs form the basis of other privacy-preserving protocols such as private data mining and graph algorithms [Fri07].

These MPSI and MPSU protocols have been studied for almost two decades now, but the current state of the art still suffers from significant costs when the number of elements k in the set or the number of parties n grows, making these protocols prohibitively expensive in practice. For example, protocols using integer-based homomorphic encryption require $O(k)$ 3072-bit ciphertexts [Bar20] and long run times due to the expensive public-key operations. Oblivious transfer-based protocols offer performance gains by offloading public-key operations to an initial phase, but they require all involved parties to send messages to all other parties.

Interestingly, several homomorphic encryption-based protocols for MPSIs [MN15; Bay+21; Deb+21a; Bay+22] and MPSUs [Bay+21] implicitly rely on secure multi-party logic in the form of private `AND` and `OR` operation. So, efficient protocols for these building blocks directly lead to efficient MPSI and MPSU protocols. In a private logic protocol, parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ submit input bits $x_1, \ldots, x_n$ to privately compute $x_1 \vee \cdots \vee x_n$. Through DeMorgan's law one can transform the same protocol to compute $x_1 \wedge \cdots \wedge x_n$. In this work we also consider the notion of 'composed' `OR`s and `AND`s, where the parties submit multiple bits at once, and a leader chooses the bits to compute these logical operations over. While private logic protocols have been studied before [HZ06], current solutions either provide weak privacy guarantees or require a high degree of interaction between all parties.

In this paper, we propose efficient protocols for performing these private `AND` and `OR` operations. Instead of offloading public key operations to an earlier phase like oblivious-transfer based protocols do, we make the operations significantly cheaper by using elliptic curve cryptography [Ber06]. In this way, we decrease the computational overhead of homomorphic encryption while the parties communicate strictly in a star topology with minimal overhead.

We compare our work against four works that represent the state of the art of MPSI and MPSU protocols [Kol+17; Bay+21; Bay+22; BA16]. The MPSI protocol by

Kolesnikov et al. [Kol+17] scales efficiently with the number of elements k, but scales quadratically with the number of parties n as it requires communication between all pairs of parties. This protocol scales largely independent of the size of the universe $|\mathcal{U}|$, and it is suitable for smaller numbers of parties. Bay et al. [Bay+21], on the other hand, propose efficient MPSI and MPSU protocols that scale linearly with the number of parties n but also with the size of the universe $|\mathcal{U}|$. Therefore, these protocols are suitable for many parties so long as the universe remains small; they would be unsuitable to represent IP elements, where $|\mathcal{U}| = 2^{32} \approx 4 \times 10^9$. Bay et al. [Bay+22] also propose another protocol that would be suitable for many parties and large universes, but the final result can contain false positives, and it only outperforms Kolesnikov et al. when the number of parties is relatively large. For example, when $k = 128$, the number of parties must exceed 65. We consider the current state of the art of MPSU protocols for large universes to be Blanton & Aguiar's [BA16]. Their secret sharing-based protocol is concretely efficient for small set sizes k, but the round complexity is $O(\log^2 k)$.

We propose to instantiate the protocols by Bay et al. [Bay+21; Bay+22] with our secure logic, providing improvements in computation and communication over previous MPSIs and MPSUs. Firstly, we achieve a run time improvement of two orders of magnitude compared to the original integer-based homomorphic MPSI protocol by Bay et al. [Bay+21], as we demonstrate in Section 3.5. We also provide run time results for our MPSU protocol that are similar to the work by Blanton & Aguiar [BA16] but with a *constant* round complexity. We claim that with these improvements and the fact that our protocols runs in the star topology, they are more practical to deploy than previous protocols that are slower or require a full mesh topology. Concretely, our contributions are as follows:

- We present two private `OR` protocols in the star topology, namely a standard and composed variant, both of which can be transformed into private `AND` protocols. We prove our protocols to be secure in the semi-honest model using a simulation-based proof and experimentally demonstrate that their run time is constant.
- We use this private logic to instantiate MPSI and MPSU protocols based on previous work [Bay+21] that compute the exact set intersection and union efficiently for small universes. The MPSI protocol is two orders of magnitude faster.
- We instantiate an approximate MPSI protocol based on previous work [Bay+22] that scales independently from the size of the universe.
- We present a novel efficient MPSU protocol that scales only logarithmically with the size of the universe for a chosen constant number of interactions.[1]

The rest of the paper is organised as follows. We discuss related work in Section 3.2, and elliptic curve cryptography and Bloom filters in Section 3.3. In Section 3.4 we propose our private `OR` and `AND` protocols. Then, we use these protocols in Section 3.5 to construct an MPSI and and MPSU protocol for small universes.

[1] Our implementations are at DOI `10.4121/5db7b31f-61f4-4b81-94e4-ab92ece86a7c` and GitHub

After that, we construct an MPSI protocol for large universes in Section 3.6, and an MPSU protocol for large universes in Section 3.7. Finally, we state opportunities for future work and conclude our paper in Section 3.8.

3.2 Related work

In this section, we first highlight previous works on multi-party private logic, and continue with multi-party private set operations. For MPSI and MPSU protocols, there are solutions based on Oblivious Transfer (OT) [Kol+17; IOP18], homomorphic encryption [KS05], and secret sharing [LW07; SCK12], among others. For both MPSI and MPSU protocols, we provide an overview of recent works with their characteristics in Table 3.1, along with our constant-time protocols and the first works on in this field. In the table, t represents the maximum resistance to collusion attacks. In other words, how large can a group of colluding parties be before the protocol's privacy guarantees fail. For a fair comparison, we take $t = n - 1$.

3.2.1 Multi-party private logic

While oblivious transfer and garbled circuits provide fast solutions for two-party private logic operations, they do not extend straightforwardly to the multi-party case. In this subsection, we discuss previous works about multi-party private `AND` and `OR` operations. Here, parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with input bits $x_1, \ldots, x_n$ want to compute either $x_1 \wedge \cdots \wedge x_n$ or $x_1 \vee \cdots \vee x_n$, respectively.

Private `OR` operations have been studied under the name *veto voting*. At the same time, previous MPSI protocols implicitly use similar constructions as veto voting schemes but inversely to compute `AND` operations.

One of the first veto voting schemes came in the form of anonymous veto networks (AV-nets) [HZ06], which are closely related to the dining cryptographers problem [Cha88]. In an AV-net, any set of parties can veto some decision without the other parties identifying them. However, this requirement is not sufficient to guarantee a private `OR` operation. Specifically, a party can locally perform the second round of the protocol on a different input to examine the result had they changed their mind. Essentially, this means that an AV-net securely computes an `OR` operation between the parties outside of each colluding set, but that makes it unusable for multi-party private set operations. PriVeto [BAH19] fixes these privacy problems using NIZKs, but as a consequence, this requires a full mesh topology.

Another veto voting scheme by Kiayias & Yung [KY03] computes $x'_1 + \cdots + x'_n$, where $x'_i = 0$ if $x = 0$, and otherwise x'_i is some random element. Debnath et al. [Deb+21a] use a similar approach for an MPSI protocol, where x'_i is either 0 or 1. The problem with the former scheme is that a party can tell if it is the only one who submitted a one [Boy+19]. The latter also leaks the number of ones in the output.

The MPSI protocol of Miyaji et al. [MN15] implicitly performs a private `AND` operation by computing $r_1(\overline{x_1} + \cdots + \overline{x_n})$ homomorphically, and checking if the result is the identity element. The randomness r_1 is generated by the leader to

prevent revealing the number of submitted ones. However, since the leader knows this randomization, it can revert it. A secure version of the protocol comes from Bay et al. [Bay+21], which computes $(r_1 + \cdots + r_n)(\overline{x_1} + \cdots + \overline{x_n})$. This scheme is conceptually identical to the veto voting scheme by Brandt [Bra05].

More generally, these MPSI protocols compute an `AND` operation as $\mathbf{r}(x_1 + \cdots + x_n)$ and then check equality with the identity element, where $\mathbf{r}$ is some randomness not known to any set of colluding parties. One can also perform this arithmetic using general-purpose multi-party computation techniques such as secret sharing, but this has two major shortcomings. First, providing collusion resistance for up to $n - 1$ parties requires each party to communicate in a full mesh topology. This would require significant bandwidth for an assistant, especially when n is large. Secondly, it is not trivial to perform *composed* operations, where the leader selects the inputs to perform the logical operation on, keeping this choice private.

An alternative arithmetic circuit for the `AND` operation is $x_1 \times \cdots \times x_n$. Also, by the inclusion-exclusion principle, the `OR` operation can be expressed as:

$$(x_1 + \cdots + x_n) + \cdots \left[\begin{smallmatrix}\text{other}\\ \text{terms}\end{smallmatrix}\right] \cdots - 1^{n+1}(x_1 \times \cdots \times x_n)\ , \tag{3.1}$$

but both operations require an n-degree multiplication. As a result, instead of a constant-round protocol, the parties need at least $O(\log n)$ rounds of communication.

In this work, we propose private `AND` and `OR` protocols that strictly function in a star topology and run in a constant number of rounds. We also provide *composed* versions. Instead of computing the aforementioned circuit for $\mathbf{r} = r_1 + \cdots + r_n$, we compute $\mathbf{r} = r_1 r_2 + r_1 r_3 + \cdots + r_1 r_n$, allowing for further optimizations.

3.2.2 Multi-party private set intersections

In this subsection, we highlight several of the latest works on MPSI, but we omit developments in two-party set intersections and threshold intersections, as these works pertain to a different setting.

Kissner & Song [KS05] proposed one of the first MPSI protocols in 2005, along with protocols that perform more complex set operations. Their approach involves encoding set elements as the roots of a polynomial. Then, using a threshold version of the Paillier cryptosystem, they add and randomize encrypted polynomials by passing them around the group of parties. The resulting polynomial only reveals the elements that were in each input set, along with a negligible probability of false positives. In Table 3.1 we refer to the topology as a 'wheel', because next to a channel between each assistant and the leader, each assistant has a channel to one other assistant, creating the shape of a wheel. After Kissner & Song [KS05], Li & Wu [LW07] proposed a similar protocol based on Shamir's secret sharing. Later works used the same set encoding [SS09; CJS12].

Table 3.1: Comparison of selected works in terms of communication, computation and security using the notation from Table 3.2. For a more comprehensive overview of collusion-resistant MPSI protocols, see Table 1.2 in Chapter 1. We present our protocols with $t = n - 1$, but it is possible to reduce the collusion threshold t, which lowers the communicational complexity.
*We adapted these complexities from the original works, see Appendix 3.A & 3.B.

Work		**Communication**				**Computation**		**Security**	
Ref.	Year	Topology	Leader	Assistant	Rounds	Leader	Assistant	Collusion	Assumption
Multi-party Private Set Intersection (MPSI) protocols									
[KS05]	2005	Wheel	$O(nk)$*	$O(tk)$*	$O(n)$	$O(tk^2)$*	$O(tk^2)$*	$n-1$	DCR
[HV17]	2017	Star	$O(nk)$	$O(k)$	$O(1)$	$O(nk^2)$*	$O(k)$*	$n-1$	DCR
[Kol+17]	2017	Full mesh	$O(nk)$	$O(tk)$	$O(1)$	$O(n)$	$O(t)$	$n-1$	TDP
[IOP18]	2018	Full mesh	$O(nk)$*	$O(nk)$*	$O(1)$	$O(nk)$*	$O(nk)$*	$n-1$	TDP
[Bay+21]	2021	Star	$O(nk)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(nkh)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	DCR
[Deb+21b]	2021	Star	$O(nk)$*	$O(k)$*	$O(1)$	$O(nkh)$*	$O(k)$*	$n-1$	DDH
[Cha+21]	2021	Full mesh	$O(nk \log k)$	$O(k \log k)$	$O(1)$	$O(nk)$	$O(k)$	$n-1$	TDP
[NTY21]	2021	Full mesh	$O(k \max(t, n-t))$	$O(k)$	$O(1)$	$O(k(n-t))$	$O(tk)$	$n-1$	TDP
[Bay+22]	2022	Star	$O(nk)$*	$O(k)$*	$O(1)$	$O(nkh)$*	$O(k)$*	$n-1$	DCR
MPSI small		Star	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	ECDDH
MPSI large		Star	$O(nk)$	$O(k)$	$O(1)$	$O(nkh)$	$O(k)$	$n-1$	ECDDH
Multi-party Private Set Union (MPSU) protocols									
[Fri07]	2007	Wheel	$O(nk)$*	$O(nk)$*	$O(n)$	$O(nk^2)$*	$O(nk^2)$*	$n-1$	DCR
[SCK12]	2012	Full mesh	$O(n^3k^2)$	$O(n^3k^2)$	$O(1)$	$\tilde{O}(n^4k^2)$*	$\tilde{O}(n^4k^2)$*	$\lfloor \frac{n}{2} \rfloor$	-
[BA16]	2016	Full mesh	$O(nk \log k + n^2)$		$O(\log k)$	$O(nk \log k + n^2)$		$\lfloor \frac{n}{2} \rfloor$	-
MPSU small		Star	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	ECDDH
MPSU large		Star	$O(n^2k \log \lvert\mathcal{U}\rvert)$	$O(nk \log \lvert\mathcal{U}\rvert)$	$O(1)$	$O(n^2k \log \lvert\mathcal{U}\rvert)$	$O(nk \log \lvert\mathcal{U}\rvert)$	$n-1$	ECDDH

Later, Miyaji & Nishida [MN15] proposed a Bloom filter-based MPSI that yields the filter representing the intersection, extending the idea of Kerschbaum et al. [Ker12] to multiple parties. They encrypt the Bloom filters using a threshold version of exponential ElGamal.

In 2017, Hazay et al. [HV17] proposed a protocol that uses the polynomial set encoding. They evaluate the polynomials obliviously using an additive homomorphic threshold cryptosystem, and provide an extension of the protocol secure in the malicious model.

Kolesnikov et al. [Kol+17] propose a protocol that uses an OT-based primitive called oblivious programmable pseudo-random functions (OPPRFs), which return a pre-programmed value when queried on elements in the receiver's set. The authors provide a public implementation with which they set speed records, but the protocol requires each pair of parties to interact with each other.

Inbar et al. [IOP18] propose another OT-based protocol that uses garbled Bloom filters. Their protocol is a multi-party version of a similar 2-party protocol [DCW13]. While in a regular Bloom filter one checks if the selected bins are set to 1, in a garbled Bloom filter one performs an `XOR` operation between those bins to check if the result is some specific value. The protocol requires all parties to interact.

Since then, Abadi et al. [ATD20] proposed an MPSI protocol in the delegated setting, where the majority of computation is outsourced to a semi-honest third party that *cannot collude* with any of the other parties participating in the protocol. Thus, this setting is different from ours, as we are interested in defending against any collusion. For this reason, we exclude that work.

Bay et al. [Bay+21; Bay+22] propose multi-party private set operations based on bitsets and Bloom filters using the threshold Paillier cryptosystem, extending the ideas of Ruan et al. [Rua+19] and fixing the security problem of Miyaji & Nishida [MN15]. The bitset-based protocols scale linearly with the size of the universe, while the Bloom filter-based MPSI scales with the number of elements k in exchange for a chance of false positives. Debnath et al. [Deb+21b] proposed a similar Bloom filter-based protocol using a threshold version of ElGamal.

Finally, Chandran et al. [Cha+21] and Nevo et al. [NTY21] published pre-prints that propose protocols inspired by Kolesnikov et al. [Kol+17], using OPPRFs as a core functionality. The work by Nevo et al. is secure in the malicious model and it outperforms both Chandran et al. and Kolesnikov et al. in their experiments. In the case when the collusion resistance $t = n - 1$, their protocol is equivalent to the protocol by Kolesnikov et al. that is secure in the (augmented) semi-honest model. For this reason, we do not compare their concrete performance, but we list their complexities in Table 3.1.

All of the papers above fall into one of two categories. Those in the first category use integer-based homomorphic encryption, do not require pairwise communication, and generally scale linearly with the number of parties. These protocols incur high computational costs for large numbers of elements k. The second category contains secret sharing and oblivious transfer-based protocols that scale quadratically with the number of parties since the complexity for an assistant scales with n or t, and require a full mesh topology. While they are concretely efficient for small numbers of parties, the protocols become prohibitively expensive for large n.

3.2.3 Multi-party private set unions

Frikken [Fri07] presents one of the first MPSU protocols. Each party represents its set as an encrypted polynomial. In turn, each party receives an encrypted polynomial, multiplies it with their polynomial, and evaluates it for their elements. The parties shuffle and decrypt the resulting ciphertexts so that for each corresponding element, there is only one ciphertext that does not decrypt to 0. Since parties pass their ciphertexts around in a circular fashion, the number of rounds in the protocol scales with the number of parties.

While the work by Shishido & Miyaji [SM18] refers to a set union in its title, the actual functionality reflects that of a multiset union as it reveals the multiplicity of each element in the resulting set. For this reason, we omit this work from our comparison. The MPSU protocol by Seo et al. [SCK12] does not have this problem, as the parties compute the least common multiple of the polynomials that represent their sets, removing any multiplicities from the polynomial roots. After this operation, the polynomials must be factored. The authors use reversed Laurent series to speed up this step. The protocol revolves around arithmetic on the rational randomized polynomials, which are shared using Shamir's secret sharing. As a result, the protocol is information-theoretically secure, but the multiplication sub-protocol requires all parties to communicate with each other. Consequently, the protocol scales poorly with the number of parties n, and quadratically with the set size k.

Blanton & Aguiar [BA16] propose multi-party private set and multiset operations using general multi-party computation techniques based on secret sharing. Their protocols involve sorting the elements, after which there exist efficient algorithms for computing the set operations. While their MPSI protocol does not reach the same level of performance as other solutions, their MPSU protocol outperforms other solutions, running in the order of seconds for small problem instances. The protocol is dominated by the oblivious sorting protocol, but if the parties already sort their sets, the round complexity is $O(\log k)$. In Table 3.1 we assume this scenario.

Finally, the only multi-party private set operation relying on elliptic curve cryptography is the union-cardinality protocol by Vos et al. [VED21]. Their protocol approximates the cardinality of an aggregated Bloom filter by shuffling it and counting the number of ones. The operation differs from an MPSU protocol.

Table 3.2: Description of symbols in this work.

Symbol	Description
	Secure logic
n	Number of parties
t	Collusion resistance, for us $t = n - 1$
$\mathcal{P}_i$	Party i
x_i	Party $\mathcal{P}_i$'s input bit
pk	Public key
sk_i	Secret key of party $\mathcal{P}_i$
f_i^x	xth evaluation pattern over party $\mathcal{P}_i$'s bits
	Sets
k	Maximum set size, so $\lvert X_i \rvert \leq k$
$\mathcal{U}$	Universe of elements
X_i	The set of party $\mathcal{P}_i$
$\hat{X}_i[j]$	Bin j of party i's set representation
	Bloom filters
N	Number of elements in a Bloom filter
m	Number of bins in a Bloom filter
h	Number of hashes in a Bloom filter
ε	Error rate of a membership query
	Divide-and-conquer
N	Length of a vector of bits
T	Number of ones in a vector of bits
R	Maximum number of iterations
D	Number of splits per iteration
	Security
$\overset{c}{\equiv}$	Computationally indistinguishable
$\overset{s}{\equiv}$	Statistically indistinguishable
C	Indices of colluding parties
$\mathbb{G}$	Elliptic curve subgroup for which DDH holds
G	Generator of group $\mathbb{G}$
$\mathcal{R}(\mathbb{G})$	Freshly random element from $\mathbb{G}$
view_i	An actual view of party $\mathcal{P}_i$
	Security assumptions
DCR	Decisional Composite Residuosity
TDP	Trapdoor Permutations
DDH	Decisional Diffie-Hellman
ECDDH	Elliptic-Curve Decisional Diffie-Hellman

3.3 Preliminaries

In this section, we give a short introduction about ElGamal over elliptic curves and Bloom filters. The notation that we use here and in the remainder of this paper can be found in Table 3.2.

3.3.1 Elliptic curve ElGamal

The ElGamal cryptosystem allows the use of any group $\mathbb{G}$ in which the DDH assumption holds [ElG84]:

Definition 5 (Decisional Diffie-Hellman)**.** Given aG and bG for some random $a, b \in \mathbb{Z}_{|\mathbb{G}|}$, abG is computationally indistinguishable from some $R \in_R \mathbb{G}$, which we write as $\mathcal{R}(\mathbb{G})$.

We use the additive notation, as is common for elliptic curve cryptography. For some elliptic curve groups, DDH is assumed to hold: in this work, we use Curve25519 [Ber06]. This curve has a co-factor of 8, which means that the prime order subgroup that we actually use in cryptographic applications is one eighth of the size of the total group. To prevent issues related to this co-factor, we use a highly-optimized encoding that realizes a true prime-order group [Ham15; VLA21b], eliminating the co-factor. Additionally, this technique allows for faster equality checks [VLA21a]. Compressed elements are only 32 bytes in size, so a single ElGamal ciphertext takes 64 bytes.

3.3.2 Bloom filters

A Bloom filter is an approximate data structure for representing sets. It consists of m bins initially set to 0. When inserting an element into the Bloom filter, the values of several bins selected by h hash functions are changed to 1. We denote such a hash function by $\mathcal{H}_i$, where i is the seed. The function maps elements uniformly to $\{0, \ldots, m-1\}$. In our implementation, we use the xxh3 hash function [Col21], which is a fast statistical hash function. We map the results to the correct range using a modulo operation. Note that the hash function does not have to be cryptographically secure, as the security of Bloom filter-based private set operations does not rely on the security of the hash function. Algorithm 1 describes how to create the Bloom filter of a set X.

Algorithm 1 Creates a Bloom filter for set X

```
1: procedure CreateBF(X, m, h)
2:     X̂ ← [0, ..., 0]                          ▷ Bit vector of length m
3:     for x ∈ X do
4:         for i = 1, ..., h do
5:             X̂[H_i(x)] ← 1
6:     return X̂
```

To check whether a Bloom filter contains a given element, we check whether the corresponding bins chosen by the hash functions are indeed all set to 1. This

operation is approximate because it might falsely conclude that an element is contained in the Bloom filter when the bins were set to 1 by coincidence through the insertion of other elements. Fortunately, this problem is well-studied, and Goel & Gupta [GG10] provide an upper bound for the probability of such a false positive ε when N elements have been inserted in a Bloom filter:

$$\varepsilon \leq \left(1 - e^{-\frac{h(N+0.5)}{m-1}}\right)^h \ . \tag{3.2}$$

In practice, we only tolerate a maximum probability of false positives ε. So, we want to select the most compact Bloom filter to satisfy this constraint, which leads to a convex minimization problem:

$$\min_{h \geq 1} \left\lceil -\frac{h(N+0.5)}{\ln 1 - \sqrt[h]{\varepsilon}} \right\rceil + 1 \ . \tag{3.3}$$

Finally, one can combine multiple Bloom filters to construct a filter representing the intersection or union using logical operations. For example, computing an `AND` operation between the bins of two respective filters yields a third filter representing their intersection, where ε is equal to that of the original Bloom filters [PSN10].

3.4 Private ORs & ANDs

In this section, we present a new protocol for privately performing `OR` or `AND` operations among multiple parties. That is, each party has an input bit, and the leader outputs the result of the logical operation over all these bits without revealing them or how many bits were true. The intuition behind our protocol is that the `OR` operation can be modeled as a summation by outputting 0 only when the sum of all inputs is 0. When at least one of the inputs is 1, the leader retrieves randomness instead, preventing the sum from revealing how many inputs were 1. By leveraging the fact that parties can submit any randomness when the input is 1, we introduce optimizations. In this section, we propose our `OR` protocol. An `AND` protocol follows by DeMorgan's law:

$$x_1 \wedge \cdots \wedge x_n = \overline{\overline{x_1} \vee \cdots \vee \overline{x_n}} \ . \tag{3.4}$$

3.4.1 Protocol description

Before executing any of our protocols, the parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ execute a short distributed setup operation over public authenticated channels. We assume that the identities of the parties are known, and a leader has been chosen beforehand. The parties aim to generate n secret ElGamal keys sk_i and a corresponding public key pk. Each party chooses their secret key randomly $sk_i \in_R \mathbb{Z}_q$. Then, they send $pk_i \leftarrow sk_i\, G$ to the leader $\mathcal{P}_1$, where G is the public generator element. G is typically chosen by the same authority that chooses group $\mathbb{G}$. Eventually, the leader computes and broadcasts public key $pk \leftarrow \sum_{i=1}^{n} pk_i$. This setup operation can also take place in a distributed fashion, where each party broadcasts pk_i. The result is a threshold version of ElGamal that requires all parties to decrypt, denoted

by (n, n)-ElGamal. One can also use a custom (t, n) setup, although this would lower the protocol's collusion resistance to t. We present our private OR operation in Protocol 1.

Instead of expressing this protocol as ElGamal operations, we use raw curve elements to perform multiple optimizations. First, we alter the encryption operation in step 1 of the protocol; since parties only have to encrypt the identity O when $x_i = 0$, or any randomness when $x_i = 1$, we let parties either create a valid encryption of O or simply choose two random curve points. To ensure that the protocol takes a constant run time, the parties perform two fixed-basepoint multiplications.

A second optimization that is particularly relevant for our *composed* private logic comes by letting the leader randomize the summation in step 2 rather than step 3 like the assistants. In doing so, the aggregated ciphertext does not reveal its constituent ciphertexts without having to perform a rerandomization operation by adding a fresh encryption of O. Finally, instead of performing a full ElGamal decryption, the leader sums all σ_i and checks if it equals $\overline{\beta}$, saving a point subtraction in the process. We also optimize point compression when performing multiple OR operations in parallel. We elaborate on this in Section 3.4.5, where we summarize the cost in elliptic curve operations.

Note that it is technically possible in the semi-honest model to let parties generate random encryptions for which they do not know the plaintext value, and replace steps 1 to 3 of the protocol. Since the parties are semi-honest, they would follow this protocol faithfully. However, it is not possible to distinguish between those randomly-generated encryptions and encryptions for which the plaintext is known. So, such a protocol does not translate to the malicious model using zero-knowledge proofs. We pose that our current protocol does not suffer from such caveats.

Private OR protocol

1. Each party $\mathcal{P}_i$ for $i = 1, \ldots, n$ computes $\langle \alpha_i, \beta_i \rangle$, where $y_i, y_i' \in_R \mathbb{Z}_q$:

$$\langle \alpha_i, \beta_i \rangle \leftarrow \begin{cases} \langle y_i G,\ y_i\, pk \rangle & \text{if } x_i = 0 \\ \langle y_i G,\ y_i'\, pk \rangle & \text{if } x_i = 1 \end{cases},$$

   and each assistant $\mathcal{P}_i$ for $i = 2, \ldots, n$ sends compressed $\langle \alpha_i, \beta_i \rangle$ to the leader $\mathcal{P}_1$.

2. The leader $\mathcal{P}_1$ computes $\langle \alpha, \beta \rangle$, where $r_1 \in_R \mathbb{Z}_q$:

$$\langle \alpha, \beta \rangle \leftarrow \left\langle r_1 \sum_{i=1}^{n} \alpha_i,\ r_1 \sum_{i=1}^{n} \beta_i \right\rangle,$$

   and sends compressed $\langle \alpha, \beta \rangle$ to the assistants.

3. Each assistant $\mathcal{P}_i$ for $i = 2, \ldots, n$ replies with compressed $\langle \overline{\alpha}_i, \overline{\beta}_i \rangle$, where $r_i \in_R \mathbb{Z}_q$:

$$\langle \overline{\alpha}_i, \overline{\beta}_i \rangle \leftarrow \left\langle r_i\, \alpha,\ r_i\, \beta \right\rangle.$$

4. The leader $\mathcal{P}_1$ computes $\langle \overline{\alpha}, \overline{\beta} \rangle$:

$$\langle \overline{\alpha}, \overline{\beta} \rangle \leftarrow \left\langle \sum_{i=2}^{n} \overline{\alpha}_i,\ \sum_{i=2}^{n} \overline{\beta}_i \right\rangle,$$

   and sends compressed $\overline{\alpha}$ to the assistants.

5. Each party $\mathcal{P}_i$ for $i = 1, \ldots, n$ computes $\overline{\sigma}_i$:

$$\sigma_i \leftarrow sk_i\, \overline{\alpha},$$

   and each assistant $\mathcal{P}_i$ for $i = 2, \ldots, n$ sends compressed σ_i to the leader $\mathcal{P}_1$.

6. The leader $\mathcal{P}_1$ returns the result z:

$$z \leftarrow \sum_{i=1}^{n} \sigma_i \stackrel{?}{\neq} \overline{\beta}\,.$$

Protocol 1: Our multi-party private OR protocol.

3.4.2 Composed logic

In the previous protocol, each party $\mathcal{P}_i$ contributes one bit x_i and the leader outputs $x_1 \vee \cdots \vee x_n$. The parties can also perform multiple parallel operations, where each party submits k bits, and the leader outputs the k results of the OR operations. An interesting case arises when the leader wants to compute an OR operation over bits of its choosing, which is a generalization of the former functionality. In this section, we propose Protocol 2 for this purpose. We show that this protocol is also privacy-preserving, and that the assistants do not learn the pattern of bits selected by the leader.

Private composed OR protocol

1. Each party $\mathcal{P}_i$ for $i = 1, \ldots, n$ computes $\langle \alpha_i^j, \beta_i^j \rangle$, where $y_i^j, y_i'^j \in_R \mathbb{Z}_q$ and $j = 1, \ldots, m$:

$$\langle \alpha_i^j, \beta_i^j \rangle \leftarrow \begin{cases} \langle y_i^j G,\ y_i^j\, pk \rangle & \text{if } x_i^j = 0 \\ \langle y_i^j G,\ y_i'^j\, pk \rangle & \text{if } x_i^j = 1 \end{cases},$$

   and each assistant $\mathcal{P}_i$ for $i = 2, \ldots, n$ sends compressed $\langle \alpha_i^j, \beta_i^j \rangle$ to the leader $\mathcal{P}_1$.

2. The leader $\mathcal{P}_1$ computes $\langle \alpha^j, \beta^j \rangle$, where $r_1^t \in_R \mathbb{Z}_q$ and $t = 1, \ldots, k$:

$$\langle \alpha^t, \beta^t \rangle \leftarrow \left\langle r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} \alpha_i^j,\ r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} \beta_i^j \right\rangle,$$

   and sends compressed $\langle \alpha^t, \beta^t \rangle$ to the assistants.

The parties continue the remaining steps in the same way as they did in Protocol 1, albeit as k parallel runs over $\langle \alpha^t, \beta^t \rangle$ for $t = 1, \ldots, k$. The leader outputs results z^t.

Protocol 2: Our composed OR protocol, where the leader chooses the bits to perform the logic over.

More formally, we let the leader select an evaluation pattern f_i^t that is a subset of $\{1, \ldots, k\}$, representing the index of the bits of party $\mathcal{P}_i$ that should be incorporated in the tth logic operation. For example, $\forall_i\ f_i^1 = \{1, \ldots, k\}$ would denote that the first logic operation incorporates all parties' bits, so the leader would learn the OR over *all* submitted bits. If $f_1^t = \emptyset$, this tth evaluation does not incorporate any of the leader $\mathcal{P}_1$'s bits. The equivalent private composed AND protocol achieved through DeMorgan's law forms the basis for the Bloom filter-based MPSI protocol in Section 3.6.

3.4.3 Correctness

For correctness, protocol 2 must output $x_1 \vee \cdots \vee x_n$ with overwhelming probability, which also implies the correctness of Protocol 1. In other words, the output is 0 only when all inputs were 0, otherwise it is 1:

Theorem 10. *With overwhelming probability, $z^t = 0$ if and only if $\forall_{i=1}^{n} \forall_{j \in f_i^t} \; x_j^i = 0$.*

Proof. We first prove the sufficient condition, so $z^t = 0$ when $\forall_{i=1}^{n} \forall_{j \in f_i^t} \; x_j^i = 0$. Following the protocol's last step, it must hold:

$$\overline{\beta}^t = \sum_{i=1}^{n} \sigma^t = \sum_{i=1}^{n} sk_i \, \overline{\alpha}^t = sk \, \overline{\alpha}^t \; , \tag{3.5}$$

where $sk = \sum_{i=1}^{n} sk_i$, the underlying key of the threshold cryptosystem. After substituting steps 3 and 4 from the original protocol and 1 and 2 from the composed protocol, we get the following identities:

$$\overline{\beta}^t = \sum_{i=2}^{n} \overline{\beta}_i^t = \sum_{i=2}^{n} r_i^t \beta^t \; , \tag{3.6}$$

$$= \sum_{i=2}^{n} r_i^t \left(r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} \beta_i^j \right) , \tag{3.7}$$

$$= \sum_{i=2}^{n} r_i^t \left(r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} y_i^j pk \right) . \tag{3.8}$$

$$sk \, \overline{\alpha}^t = sk \sum_{i=2}^{n} \overline{\alpha}_i^t = sk \sum_{i=2}^{n} r_i^t \alpha^t \; , \tag{3.9}$$

$$= sk \sum_{i=2}^{n} r_i^t \left(r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} \alpha_i^j \right) , \tag{3.10}$$

$$= sk \sum_{i=2}^{n} r_i^t \left(r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} y_i^j G \right) , \tag{3.11}$$

$$= \sum_{i=2}^{n} r_i^t \left(r_1^t \sum_{i=1}^{n} \sum_{j \in f_i^t} y_i^j pk \right) . \tag{3.12}$$

This last step works because $pk = sk \, G$. This completes the first part of the proof.

Next, we prove the necessary condition, so $z^t = 1$ when $\exists_{i=1}^{n} \exists j \in f_i^t \; x_i^j = 1$ with overwhelming probability. Since some $x_i^j = 1$, the corresponding y_i^j in Equation (3.11) will be replaced by some $y_i^{'j} \in_R \mathbb{Z}_q$. As a result the equality only holds with a uniformly random probability of $\frac{1}{q}$, which is negligible. □

3.4.4 Privacy

We now provide a simulation-based proof to formally show that our protocols are indeed private in the semi-honest model, following the requirements for a deterministic functionality as described in [Lin17]. Notice that when $f_i^t = \{t\}$ for $t = 1, \ldots, k$ and $i = 1, \ldots, n$, Protocol 2 reduces down to k parallel executions of Protocol 1. In other words, proving security of the first implies security of the latter. For this reason we only provide such a proof for Protocol 2. We pose that our protocol is secure against $n-1$ colluding parties, so two cases arise:

1. The leader is honest and up to $n-1$ assistants are colluding.
2. The leader is colluding with up to $n-2$ assistants.

Multiple parts of our proof rely on the following lemma:

Lemma 11. *Consider group $\mathbb{G}$ for which the decisional Diffie-Hellman (DDH) assumption is assumed to hold. Given element $G' \in \mathbb{G}$, unknown randomness $r \in_R \mathbb{Z}_q$, and $p = sG'$ for the unknown $s \in \mathbb{Z}_q$, it holds that:*

$$\langle rG', rsG' \rangle \overset{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle \ .$$

Proof. The first term is statistically indistinguishable from randomness, since $r \in_R \mathbb{Z}_q$, so $rG' \overset{s}{\equiv} \mathcal{R}(\mathbb{G})$. The second term rsG' is computationally indistinguishable by the DDH assumption in Definition 5 by taking $a \leftarrow r$, $b \leftarrow s$, $G \leftarrow G'$ (so $bG \leftarrow p$). So, $rsG' \overset{c}{\equiv} \mathcal{R}(\mathbb{G})$. □

We first prove that a simulator exists for the first case, which generates a view for up to $n-1$ colluding assistants that is indistinguishable from their own, given these parties' inputs.

Theorem 12. *For a set of colluding parties $C \subseteq \{2, \ldots, n\}$ there exists a simulator $\mathcal{S}_1$ so that:*

$$\mathcal{S}_1() \overset{c}{\equiv} \bigcup_{c \in C} \mathsf{view}_c(x_c) \ . \tag{3.13}$$

Proof. We construct simulator $\mathcal{S}_1$. The simulator takes no inputs because in this case, we can generate an indistinguishable view without explicitly incorporating them. Since the colluding parties are all assistants, the simulator also does not consider any output of the protocol. The view generated by the simulator is a complete set of simulated messages from the honest parties, because the channels are public: $\{\alpha^t, \beta^t, \overline{\alpha}^t\}$ and $\{\alpha_i^j, \beta_i^j, \overline{\alpha}_i^t, \overline{\beta}_i^t, \sigma_i^t\}$ for all honest $\mathcal{P}_i$, in other words $i \in \overline{C}$.

Simulator $\mathcal{S}_1$ generates the view by sampling random elements from the curve group for all the aforementioned messages except for $\overline{\alpha}^t$ and $\overline{\beta}^t$, which it computes by executing step 4.We show that such a view is indeed indistinguishable from the actual views.

For step 1 of the protocol, we must show that it holds that $\langle \alpha_i^j, \beta_i^j \rangle \overset{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$, or more specifically:

$$\langle y_i^j G, \ y_i'^j \, pk \rangle \overset{s}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle \ , \tag{3.14}$$

$$\langle y_i^j G, y_i^j pk \rangle \overset{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle \ . \tag{3.15}$$

Equation 3.14 holds because y_i^j and $y_i'^j$ are sampled randomly, covering the case where $x_i^j = 1$. Next, Equation 3.15 holds by Lemma 11 where $s \leftarrow sk$, $G' \leftarrow G$, and $r \leftarrow y_i^j$, covering the case $x_i^j = 0$.

For step 2 of the protocol we must show $\langle \alpha^t, \beta^t \rangle \stackrel{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$. To simplify notation, we say that:

$$v_i'^j = \begin{cases} y_i^j & \text{if } x_i^j = 0 \\ y_i'^j & \text{if } x_i^j = 1 \end{cases} . \tag{3.16}$$

Then, our former statement holds by Lemma 11, where:

$$r \leftarrow r_1^t, \quad G' \leftarrow \sum_{i=1}^{n} \sum_{j \in f_i^t} \alpha_i^j, \quad s \leftarrow sk \sum_{i=1}^{n} \sum_{j \in f_i^t} \frac{v_i^j}{y_i^j} ,$$

and s is unknown because it is a factor of the unknown key sk.

For step 3 of the protocol we must show $\langle \overline{\alpha}_i^t, \overline{\beta}_i^t \rangle \stackrel{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$. Again, this holds by Lemma 11, where $r \leftarrow r_i$, $G' \leftarrow r_1 G'$, and $s \leftarrow r_1 s$. Here we reuse the values from our previous argument.

For step 4 of the protocol, the simulator executes the step as usual for $\overline{\alpha}^t$. For corrupted parties $c \in C$, the simulator samples $\langle \overline{\alpha}_c^t, \overline{\beta}_c^t \rangle \leftarrow \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$, which is statistically indistinguishable from the actual view.

For step 5 of the protocol we must show that $\sigma_i \stackrel{c}{\equiv} \mathcal{R}(\mathbb{G})$. This holds because sk_i is unknown to the corrupted parties and is sampled uniformly from $\mathbb{G}$. □

We also prove that a simulator exists that generates a view for a colluding leader and up to $n - 2$ colluding assistants in Theorem 13, which is indistinguishable from their own, given these parties' inputs and the protocol's output:

Theorem 13. *For a set of colluding parties $C = \{1\} \cup C'$ $C' \subset \{2, \ldots, n\}$ there exists a simulator $\mathcal{S}_2$ so that:*

$$\mathcal{S}_2(z) \stackrel{c}{\equiv} \bigcup_{c \in C} \text{view}_c(x_c) . \tag{3.17}$$

Proof. We construct simulator $\mathcal{S}_2$, which takes the protocol's output z. The simulator takes no inputs because we can generate an indistinguishable view without explicitly incorporating them. The output of the simulator is a complete set of simulated incoming messages from the remaining honest assistants: $\{\alpha_i^j, \beta_i^j, \overline{\alpha}_i^t, \overline{\beta}_i^t, \sigma_i^t\}$ for all honest $\mathcal{P}_i$, in other words $i \in \overline{C}$.

First, the simulator randomly samples $\langle \alpha_i^j, \beta_i^j \rangle \leftarrow \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$ and $\langle \overline{\alpha}_i^t, \overline{\beta}_i^t \rangle \leftarrow \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$, which are computationally indistinguishable from actual views as argued in Theorem 12.

The simulator then generates σ_i^t, which depends on the expected output z. If $z = 1$, the simulator randomly samples $\sigma_i^t \leftarrow \mathcal{R}(\mathbb{G})$. However, if $z = 0$, the simulated output would be incorrect with overwhelming probability, as shown at the end of Theorem 10. Instead, the simulator will choose one honest assistant

$H \in \overline{C}$ for which it generates another σ_H^t. For the other assistants $c \in \overline{C} \setminus \{H\}$, the simulator samples $\sigma_c^t \leftarrow \mathcal{R}(\mathbb{G})$. The simulator computes:

$$\sigma_H \leftarrow \overline{\beta}^t - \left(\sum_{c \in C} \sigma_c + \sum_{c \in \overline{C} \setminus \{H\}} \sigma_c \right). \tag{3.18}$$

It is clear to see that $\sum_{i=1}^{n} \sigma_i^t = \overline{\beta}^t$, so the output is 0.

Finally, we show that σ_i^t is indeed indistinguishable from the actual views. If it was sampled randomly, then it holds that $\sigma_i^t \overset{s}{\equiv} sk_i\, \overline{\alpha}^t$, because sk_i is an unknown value sampled randomly from $\mathbb{Z}_q$. Moreover, since we only choose assistant $H \in \overline{C}$ when $z = 0$, its σ_H^t is also statistically indistinguishable from an actual view since the other values in the summation are statistically indistinguishable and the output is known to be 0. □

Finally, we show that our implemented protocol runs in constant-time in the next subsection.

3.4.5 Efficiency

When presenting the protocol, we hinted that we can optimize the point compression step when performing multiple OR operations in parallel. The reason is that the compression operation is batchable, if we allow the compressed point to be doubled. Note that for steps 1 to 4 of this protocol, this has no impact on the encrypted value, but for steps 5 and 6 there will be a factor 2 discrepancy between σ and β. Fortunately, one can compensate for this in the secret keys. In short, after generating the public key pk, each party divides their secret key by 4 offsetting the factor induced by batch-compressions, so the actual key becomes $sk \leftarrow \frac{1}{4} sk_i$.

We summarize the computational cost of our protocol in Table 3.3 as the number of elliptic curve operations performed, and compare it against the naive approach of computing $(r_1 + \cdots + r_n)(x_1 + \cdots + x_n)$ using additively homomorphic encryption. Communication-wise, the leader must send four compressed points to each assistant, while each assistants sends five compressed points to the leader. Given our choice of Curve25519, this means that for one private logic operation, the leader sends $128(n-1)$ bytes and an assistant sends 160 bytes. Asymptotically, both the computational and communicational complexities are $O(n)$ for the leader and $O(1)$ for an assistant, although the number of point multiplications stays constant, regardless of n. As described in Section 3.2.1, one can also perform this arithmetic circuit using secret sharing. However, the total communication cost would scale quadratically with n since it requires all parties to communicate. We analytically compare the communication cost of this approach with our protocol in Figure 3.1, where shares are 5 bytes in size and the parties compute the multiplication using pre-distributed Beaver triplets. For $n \geq 10$, the communication overhead of this approach would exceed that of our protocol.

While we described our private OR protocol to function on single-bit inputs, the parties can perform this operation on many bits in parallel. We explicitly use this technique to perform efficient MPSI and MPSU protocols. We provide an

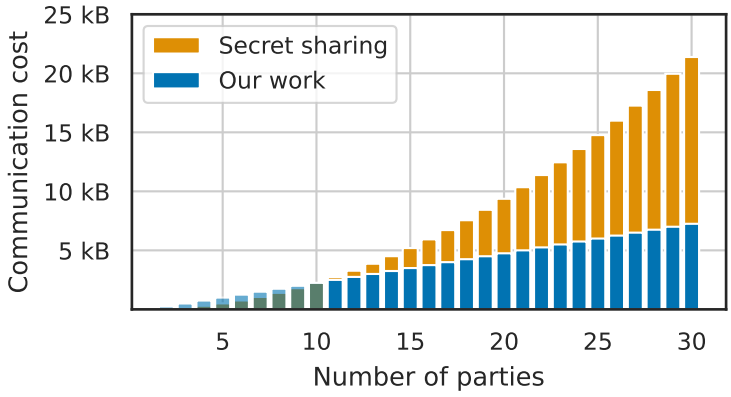


Figure 3.1: Communication in one private OR computation

open-source implementation of such parallel `OR`s and `AND`s. In our implementation, we do not simulate communication delays, but we do route the messages through Unix streams. Figure 3.2 shows the run time of our private `OR` protocol on 1024 bits in parallel for an increasing number of inputs that are 1. The figure underlines that the run time of our protocol indeed does not depend on the input. In the remainder of this work we perform all our experiments on a Unix machine with 30 virtual Intel® Xeon® Cascade Lake CPUs at 3100 MHz. We assign each party one execution thread to run on. The machine also has 120 GB of memory allocated to it, but in our experiments we only use a fraction of this. All our implementations are written in Rust.

3.5 Private set operations for small universes

One approach for computing a set intersection is to check for each element in the universe that it is present in all the sets. For the union, one checks if an element occurs in at least one of the sets. This is the idea behind the protocols of Bay et al. [Bay+21], which use the bitset representation. A bitset represents a set as a vector of bits corresponding to each element in the universe. When an element is in the set, the corresponding bit is set to 1; otherwise, it is 0. Computing the intersection then constitutes an element-wise `AND` operation, and the union constitutes an `OR`

Table 3.3: EC operations for our private `OR` protocol on one bit.

		Addition	Fixed mult.	Variable mult.
Naive	Leader	$5n-1$	4	3
	Assistant	1	2	3
Ours	Leader	$5n-7$	2	3
	Assistant	-	2	3

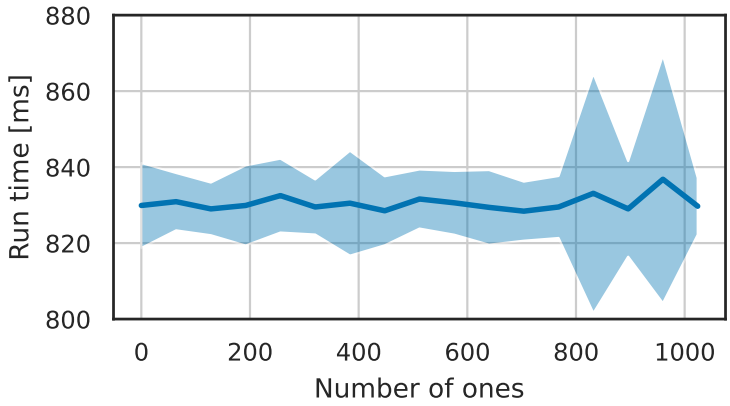


Figure 3.2: The run time of 1024 private ORs is constant w.r.t. the number of 1s. The shaded area is the 99% confidence interval.

operation. In this section, we instantiate such bitset-based protocols using our private AND and OR protocols, as presented in Protocol 3.

MPSI protocol for small universes

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the bitset $\hat{X}_i$ of their set X_i:

$$\hat{X}_i[j] = \begin{cases} 1 & \text{if } j \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases}$$

2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ take part in a private AND protocol on $\hat{X}_i$, so the leader $\mathcal{P}_1$ retrieves $\hat{Z}$.

3. The leader $\mathcal{P}_1$ returns the result Z:

$$Z = \{j \in \mathcal{U} \mid \hat{Z}[j] = 1\}$$

Protocol 3: A bitset-based MPSI protocol. An MPSU protocol would use a private OR protocol instead.

This MPSI protocol inherits its security properties from the private AND protocol since the private data is only accessed during the execution of that sub-protocol. The same holds for the MPSU protocol. The efficiency of these protocols is also decided by the private logic protocols as they dominate the computation required. Since the parties perform one private logic operations for each element in the universe, the computational complexity is $O(n\,|\mathcal{U}|)$ for the leader and $O(|\mathcal{U}|)$ for an assistant.

We experimentally compare the run time of this MPSI protocol with the implementation by Bay et al. [Bay+21] using the same setup as before. Note, however, that the original work uses a 1024 bit modulus to instantiate the Paillier cryptosystem, which corresponds to a legacy security strength of 80 bits. To ensure a fair comparison and cryptographic security, we instead choose a 3072 bit modulus as per NIST's standard [Bar20], corresponding to 128 bits of security. The results of this experiment are in Figure 3.3.

For $n = 2$ and $|\mathcal{U}| = 256$, this protocol outperforms the implementation by Bay et al. by almost two orders of magnitude. While, the implementation by Bay et al. seems to be hardly affected by the number of parties in Figure 3.3, this is an artifact of the logarithmic axis. The absolute increase in run time when the number of parties grows is comparable to ours: for $|\mathcal{U}| = 256$, Bay et al. takes $9.47, 9.62, 9.73$ seconds for $n = 2, 5, 10$, while this protocol takes $0.12, 0.20, 0.34$ seconds.

Finally, notice that Protocol 3 can actually be further optimized by instantiating it with a *composed* `AND` protocol, in exchange for leaking the leader's set size. The initial steps of this protocol would still scale with $|\mathcal{U}|$, but the remaining steps would scale with k.

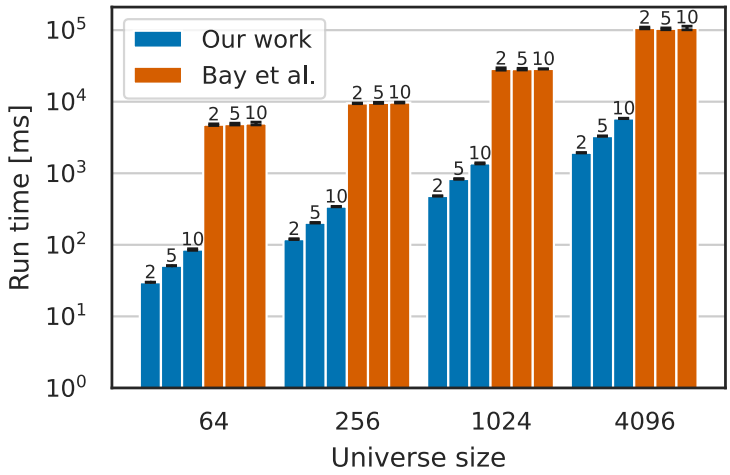


Figure 3.3: Run time comparison between our MPSI protocol and [Bay+21]. The numbers over the bars indicate the number of parties n and the error bars the 99% confidence interval.

3.6 Private set intersections for large universes

In practice, the size of a party's set is often significantly smaller than the size of the universe, meaning $k \ll |\mathcal{U}|$. In this section, we instantiate an MPSI protocol with our private logic that scales only with k rather than $|\mathcal{U}|$ in exchange for a false positive rate ε that must be so small that it is negligible.

The difference between this protocol and the MPSI protocol for small universes is that parties represent their set as a Bloom filter rather than a bitset. As we discuss in Section 3.6.3, this causes the protocol to scale independently of the size of the

universe. We aggregate the Bloom filters similarly to bitsets, but using our private composed AND protocol. We adapt this idea from Bay et al. [Bay+22], although the same idea has been applied more often in MPSI protocols, such as by Miyaji & Nishida [MN15] and Debnath et al. [Deb+21b], but these suffer from security flaws as described in Section 3.2.1.

We present the updated Protocol 4, where $X_1[t]$ represents the tth element of the leader's set. Here, the parties engage in a private *composed* logic protocol to ensure that no information is leaked from the resulting Bloom filter. In other words, the leader computes an AND operation between the bins corresponding to each of its elements, essentially performing at most k private membership checks. The privacy of the leader's elements in turn relies on the assistants not learning the evaluation pattern f_i^x. We provide more details on the security properties of the protocol in Section 3.6.2.

MPSI protocol for large universes

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the Bloom filter $\hat{X}_i$ of their set X_i:
$$\hat{X}_i = \textsc{CreateBF}(X, m, h)$$
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ take part in a private composed AND protocol on $\hat{X}_i$ with $f_i^t = \{\mathcal{H}_j(X_1[t]) \mid t = 1, \ldots, k,\ j = 1, \ldots, h\}$, so that the leader $\mathcal{P}_1$ retrieves $\hat{Z}$.
3. The leader $\mathcal{P}_1$ returns the result Z:
$$Z = \left\{X_1[t] \mid \hat{Z}[t] = 1 \text{ for } t = 1, \ldots, k\right\}$$

Protocol 4: A Bloom filter-based MPSI protocol.

3.6.1 Correctness

There are three properties that must hold with overwhelming probability for the protocol to be correct:

- When all parties have an element $X_1[t] = x$ it must hold that $\hat{Z}[t] = 1$.
- When the leader has an element $X_1[t] = x$ but at least one other party does not have x in their set $\hat{Z}[t] = 0$ must hold.
- When an element $x \notin X_1$ it must hold that $x \notin Z$.

For the first case, notice that $\hat{X}_i[j] = 1$ for all parties $i = 1, \ldots, n$ and $j = \mathcal{H}_1(x), \ldots, \mathcal{H}_h(x)$ after the parties create their Bloom filters. Since $X_1[t] = x$, it

holds that:

$$\hat{Z}[t] = \bigwedge_j \hat{X}_1[j] \wedge \cdots \wedge \hat{X}_n[j] = 1 \,. \tag{3.19}$$

In the second case, there is a probability ε that all bins corresponding to the element x are set to 1. However, as explained in Section 3.3.2, we can choose parameters so that ε is negligible. Then, with overwhelming probability, there is at least one party i' and Bloom filter bin j' for which it holds that $\hat{X}_{i'}[j'] = 0$. Now:

$$\hat{Z}[t] = \cdots \wedge \hat{X}_{i'}[j'] \wedge \cdots = 0 \,. \tag{3.20}$$

In the last case, for each $t = 1, \ldots, k$ it also holds with overwhelming probability that at least for one bin $\hat{X}_1[j'] = 0$ with $j' \in \{\mathcal{H}_1(x), \ldots, \mathcal{H}_h(x)\}$. As a result:

$$\hat{Z}[t] = \cdots \wedge \hat{X}_1[j'] \wedge \cdots = 0 \,. \tag{3.21}$$

3.6.2 Privacy

Since the leader chooses the evaluation pattern to reflect the operation of checking whether an element is contained in the Bloom filter, the protocol's entire security again relies on the private logic primitive. Note that if we had not used the *composed* `AND` protocol and therefore exposed the entire resulting Bloom filter, the protocol would leak information that is inherent to the way Bloom filters combine under intersections. This problem was also hinted at in previous works [Bur+10; DCW13]. This leakage arises because it does **not** necessarily hold that:

$$\begin{aligned} &\textsc{CreateBF}(X_1 \cap \cdots \cap X_n, m, h) = \\ &\textsc{CreateBF}(X_1) \texttt{ AND } \ldots \texttt{ AND } \textsc{CreateBF}(X_n) \,, \end{aligned} \tag{3.22}$$

which occurs when 1s in the input Bloom filters not belonging to the actual intersection align by accident.

Moreover, in Chapter 2, we showed that the approximate nature of Bloom filter also leads to other exploitable weaknesses when using small Bloom filters. Specifically, one can use the non-negligible probability of false positives in such a small Bloom filter to learn information about elements that are not strictly in the intersection. Consequently, for our protocol to be secure, one must choose large enough parameters, for example according to (2.10) in Section 2.5.4.

3.6.3 Efficiency

We rewrite Equation 3.2 to isolate m:

$$\varepsilon \leq \left(1 - e^{-\frac{h(N+0.5)}{m-1}}\right)^h \tag{3.23}$$

$$m \geq -\frac{h(N+0.5)}{\ln\left(1 - \sqrt[h]{\varepsilon}\right)} + 1 \tag{3.24}$$

$$m \geq \left(\frac{-h}{\ln\left(1 - \sqrt[h]{\varepsilon}\right)}\right) N - \left(\frac{0.5h}{\ln\left(1 - \sqrt[h]{\varepsilon}\right)}\right) + 1 \tag{3.25}$$

As such, for a constant h and ε, it holds that the minimal number of bins m scales linearly with N. In short, $m = O(N)$. In practice we choose h depending on ε and N to choose the smallest m in that situation. In the protocol, the private `AND` operations dominate the run time. Each party takes part in h of such operations over m bits, so the computational complexity for the leader is $O(nmh)$, and for an assistant is $O(m)$. Since $m = O(N)$ and $N = k$, we write the final complexities as $O(nkh)$ for the leader and $O(k)$ for an assistant. For concrete parameter choices, we refer the reader to Table 2.1 in Chapter 2.

3.7 Private set unions for large universes

When $k \ll |\mathcal{U}|$, a bitset representation would be filled almost entirely with 0s, but we are only interested in searching for the 1s. To prevent wasting computations on this sparse vector, we propose a divide-and-conquer algorithm that isolates these 1s. The intuition is as follows: Each party splits their bit vector into D partitions and locally computes the logical `OR` of the bits in each partition. After that, the parties take part in our private `OR` protocol on the aggregated bits. In this way, they can discard all partitions for which the result is 0, as none of the original bits in the partition is a 1. As a concession, this approach allows the assistants to learn information about the final set union, but they do not learn more than the leader. The parties repeat this process until a partition only contains one bit. We provide an example of this process in Figure 3.4. Instead of running 27 private `OR` protocols, the example only requires 15 runs. For larger universes, the difference will be even greater. We provide a formal description in Protocol 5, but first, we discuss its underlying algorithms.

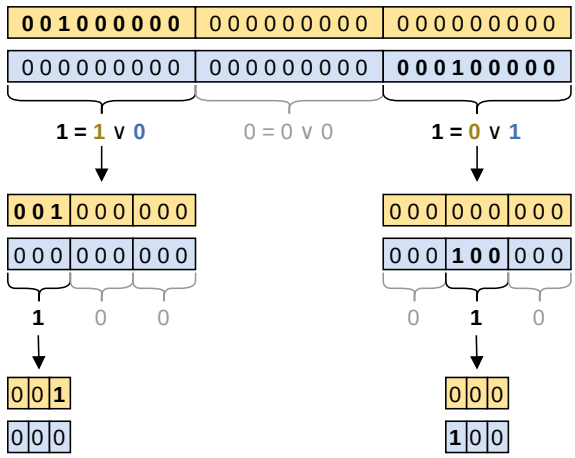


Figure 3.4: Example of the divide-and-conquer approach with two parties, $N = 27$, $T = 2$ and $D = 3$.

3.7.1 Underlying algorithms

We provide pseudocode for the underlying algorithms of the divide-and-conquer approach for multi-party private set unions on bitsets. Algorithm 2 provides the functionality to split a range up into D similarly sized partitions.

Algorithm 2 Selects the indices of D partitions

```
1: procedure Split(min, max, D)
2:     s ← (max−min)/D
3:     indices ← []
4:     i ← min
5:     while i < max do
6:         j ← i + s
7:         Append (⌊i⌉, ⌊j⌉) to indices
8:         i ← j
9:     return indices
```

Algorithm 3 in turn uses the splitting algorithm to run the actual recursive MPSU protocol.

Algorithm 3 Divide-and-conquer approach for one party

```
1: procedure DivideAndConquer(X̂, D)
2:     result ← [0, ..., 0]                          ▷ Bit vector of length |𝒰|
3:     I_prev ← [(0, |𝒰|)]
4:     while |I_prev| > 0 do
5:         I_curr ← [Split(i, j, D)   ∀(i, j) ∈ I_prev]
6:         I_prev ← [ ]
7:         ▷ This for loop can be performed in parallel
8:         for (i, j) ∈ I_curr do
9:             x ← X̂[i] ∨ ··· ∨ X̂[j]
10:            Take part in a private OR protocol with x
11:            Receive result z from the leader 𝒫_1
12:            if j − i = 1
13:                result[i] = 1
14:            else if z = 1
15:                Append (i, j) to I_prev
16:    return result
```

Finally, we provide a derivation for finding that the optimal partition number $D = e$. For $N > 1$ we have the following minimization problem:

$$\min_{D>1} \quad D \log_D N \ . \tag{3.26}$$

We find the optimum by differentiating for D and determining when the derivative is 0:

$$0 = \frac{d}{dD} D \log_D N \,, \tag{3.27}$$

$$= \frac{d}{dD} D \frac{\ln N}{\ln D} \,, \tag{3.28}$$

$$= \frac{\ln(N)(\ln(D) - 1)}{\ln^2 D} \,. \tag{3.29}$$

Since $N > 1$, the only solution is $D = e$.

MPSU protocol for large universes

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the bitset $\hat{X}_i$ of their set X_i:

$$\hat{X}_i[j] = \begin{cases} 1 & \text{if } j \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases}$$

2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ execute DivideAndConquer($\hat{X}_i, D$), computing the bit-wise OR between their bitsets so that the leader $\mathcal{P}_1$ retrieves $\hat{Z}$.

3. The leader $\mathcal{P}_1$ returns the result Z:

$$Z = \{j \in \mathcal{U} \mid \hat{Z}[j] = 1\}$$

Protocol 5: Our multi-party private set union protocol, relying on Algorithms 2 & 3.

3.7.2 Choosing the number of divisions

Consider a vector with $N = 16$ bits, of which only one is 1. When the number of ones $T = 1$ and the number of divisions $D = 2$, we must perform four iterations of the divide-and-conquer approach to reach partitions containing only one bit. Notice that since $T = 1$, we always discard all but one division. In the general case, we require $\log_D N$ iterations of D runs of the OR protocol, so we require $D \log_D N$ private ORs in total. When $T > 1$, we can extend this to a loose upper bound of $TD \log_D N$. After all, in the worst case, each 1 ends up in a separate partition at each iteration.

While the optimal choice of D is Euler's constant e (see previous subsection), this is not practically attainable:

- For ease of implementation, we want D to be an integer.
- For a small D and large $|\mathcal{U}|$, we require many iterations.

- For a constant number of rounds, D cannot be constant.

Instead of choosing $D = e$, we select a suitable number of divisions based on a specified maximum number of iterations R. Let us define the function ORs that returns the expected number of private `OR`s; then choosing D comes down to a minimization problem:

$$\min_{D \geq \sqrt[R]{N}} \text{ORs}(T, N, D) \ . \tag{3.30}$$

The reason that $D \geq \sqrt[R]{N}$ is that splitting a vector into D parts for R iterations allows us to reach exactly $N = D^R$ partitions of size 1 in the final round.

To describe ORs, we analyze the cost of each iteration. The first iteration requires $C_0 = D$ `OR` operations. We can view this as a balls and bins problem, in which T balls are divided among D bins. Only those bins that contained at least one ball continue in the protocol. Assuming that such a bin has a limitless capacity, we express the expected number of filled bins by:

$$\text{spread}(\text{balls}, \text{bins}) = \text{bins}\left(1 - \exp - \frac{\text{balls}}{\text{bins}}\right) \ . \tag{3.31}$$

We derive this function from Equation 1 in [VED21] when $h = 1$. Given this equation, we express the expected cost of iteration i by $C_i = D$ spread(T, C_{i-1}), since each iteration splits the number of filled bins of the previous iteration again into D partitions. This expected cost holds for all but the last iteration, where there may not be as many bits N as the number of partitions we can form. We compensate for this by computing the expected number of partitions that remain as:

$$B = \frac{N}{N - D^{\lfloor \log_D N \rfloor}} \ . \tag{3.32}$$

Now, the expected number of private `OR` operations is:

$$\text{ORs}(T, N, D) = B \, \text{spread}(T, C_{\lfloor \log_D N \rfloor} B) + \sum_{i=0}^{\lfloor \log_D N \rfloor} C_i \ , \tag{3.33}$$

As mentioned before, in the final iteration, we may form more partitions than there are bits. This is only the case when $\log_D N$ is not a whole number. As such, we reduce the optimal choice for D that leads to the least `OR` operations to searching for $D = \sqrt[j]{N}$, where:

$$\min_{j=2,\ldots,R} \text{ORs}(T, N, \sqrt[j]{N}) \ . \tag{3.34}$$

Here, j is the required number of iterations which is less than or equal to the predefined maximum R. To determine j, we evaluate all possible values $j = 2, \ldots, R$. For the MPSU protocol, the other parameters are $T = nk$ and $N = |\mathcal{U}|$. We note that we are optimizing for the average case, where 1s are randomly distributed, but ideally the 1s are bundled together. So, if there is some structure in the set elements, one can achieve performance gains by increasing the probability that ones end up together in the same partition.

3.7.3 Privacy

We argue that assuming our MPSU protocol for small universes is privacy-preserving, the same holds for this protocol. Consider two parties $\mathcal{P}_1$ and $\mathcal{P}_2$ with bitsets:

$$\hat{X}_1 = \left[x_1^1, x_1^2, \ldots, x_1^{|\mathcal{U}|}\right] \ , \quad \hat{X}_2 = \left[x_2^1, x_2^2, \ldots, x_2^{|\mathcal{U}|}\right] \ .$$

Then, in our previous MPSU protocol, these parties would learn:

$$\left[(x_1^1 \vee x_2^1), (x_1^2 \vee x_2^2), \ldots, (x_1^{|\mathcal{U}|} \vee x_2^{|\mathcal{U}|})\right] \ . \tag{3.35}$$

Let us say that the first two bits end up in one partition, then the parties learn:

$$(x_1^1 \vee x_1^2) \vee (x_2^1 \vee x_2^2) = (x_1^1 \vee x_2^1) \vee (x_1^2 \vee x_2^2) \ . \tag{3.36}$$

So the two parties only learn the logical OR of bits they would have learned regardless in our previous MPSU protocol. In other words, the information they learn is a function of the output, rather than their private inputs. Regarding timing attacks, the divide-and-conquer approach does not strictly run in constant time. Instead, the run time is correlated with the size of the output set.

3.7.4 Efficiency

An upper bound for the number of OR operations, which dominate the performance of the protocol, is $nkD \log_D |\mathcal{U}|$. So, when D is constant, both the computational and communication complexities are $O(n^2 k \log |\mathcal{U}|)$ for the leader, and $O(nk \log |\mathcal{U}|)$ for an assistant. The concrete run time scales with the size of the resulting union.

3.7.5 Results

To the best of our knowledge, there are no public implementations of other MPSU protocols. Comparing against the MPSU protocol for small universes is also infeasible, as the run time would exceed hours for larger universes. Instead, we evaluate the run time of this protocol for a growing maximum number of iterations R in Figure 3.5, where the universe has the size of the IPv4 space. Since the decrease in run time tapers off at 8 iterations, we consider this the optimal choice in this instance. In our experiment we do not simulate additional communication delays, however, one might trade-off this delay with a party's computational effort. We note that the actual run time of the protocol does not scale linearly with set size, as the probability increases for two elements to map to the same partition, allowing the protocol to discard more partitions.

3.8 Conclusion

In this work, we instantiate existing MPSI and MPSU protocols with elliptic curve-based private logic protocols to perform fast set operations on any size of universe.

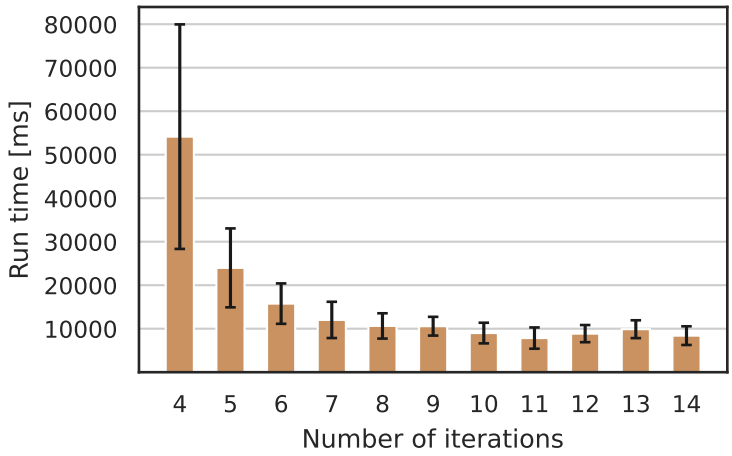


Figure 3.5: Run time of our MPSU protocol for large universes when the number of iteration increases. Here, $n = 5$, $k = 32$ and $|\mathcal{U}| = 2^{32}$. The error bars denote the standard deviation. The decrease in run time tapers off around $R = 8$.

Our novel private logic protocols may also be of independent interest. Most previous MPSI and MPSU protocols either use significantly slower integer-based homomorphic encryption or oblivious transfers that require interactions between all parties. Our protocols, however, enjoy the low computational cost of elliptic curve operations and function in the star topology. Moreover, we propose a novel MPSU protocol for large universes that uses a divide-and-conquer approach to significantly reduce computation at the cost of more interactions. Still, it remains an open question to design an exact elliptic curve-based MPSI or MPSU that does not depend on the size of the universe.

We open-source a proof-of-concept implementation of all protocols. We also demonstrate that the protocols' run times are constant and we underline their security using a simulation-based proof. The MPSI protocol for small universes is two orders of magnitude faster than the protocol by Bay et al. [Bay+21].

References

[ATD20] Aydin Abadi, Sotirios Terzis, and Changyu Dong. "Feather: Lightweight Multi-party Updatable Delegated Private Set Intersection". In: *IACR Cryptol. ePrint Arch.* (2020), p. 407. URL: https://eprint.iacr.org/2020/407.

[BA16] Marina Blanton and Everaldo Aguiar. "Private and oblivious set and multiset operations". In: *Int. J. Inf. Sec.* 15.5 (2016), pp. 493–518. DOI: 10.1007/s10207-015-0301-1. URL: https://doi.org/10.1007/s10207-015-0301-1.

[BAH19] Samiran Bag, Muhammad Ajmal Azad, and Feng Hao. "PriVeto: a fully private two-round veto protocol". In: *IET Inf. Secur.* 13.4 (2019), pp. 311–320. DOI: 10.1049/iet-ifs.2018.5115. URL: https://doi.org/10.1049/iet-ifs.2018.5115.

[Bar20] Elaine Barker. *Recommendation for Key Management: Part 1 – General*. Tech. rep. SP 800-57 Part 1 Rev. 5. NIST, May 2020.

[Bay+21] Aslí Bay et al. "Multi-Party Private Set Intersection Protocols for Practical Applications". In: *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*. Ed. by Sabrina De Capitani di Vimercati and Pierangela Samarati. SCITEPRESS, 2021, pp. 515–522. DOI: 10.5220/0010547605150522. URL: https://doi.org/10.5220/0010547605150522.

[Bay+22] Aslí Bay et al. "Practical Multi-Party Private Set Intersection Protocols". In: *IEEE Trans. Inf. Forensics Secur.* 17 (2022), pp. 1–15. DOI: 10.1109/TIFS.2021.3118879. URL: https://doi.org/10.1109/TIFS.2021.3118879.

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*. Ed. by Moti Yung et al. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 207–228. DOI: 10.1007/11745853_14. URL: https://doi.org/10.1007/11745853%5C_14.

[Boy+19] Colin Boyd et al. "A Blind Coupon Mechanism Enabling Veto Voting over Unreliable Networks". In: *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*. Ed. by Feng Hao, Sushmita Ruj, and Sourav Sen Gupta. Vol. 11898. Lecture Notes in Computer Science. Springer, 2019, pp. 250–270. DOI: 10.1007/978-3-030-35423-7_13. URL: https://doi.org/10.1007/978-3-030-35423-7%5C_13.

[Bra05] Felix Brandt. "Efficient Cryptographic Protocol Design Based on Distributed El Gamal Encryption". In: *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*. Ed. by Dongho Won and Seungjoo Kim. Vol. 3935. Lecture Notes in Computer Science. Springer, 2005, pp. 32–47. DOI: 10.1007/11734727_5. URL: https://doi.org/10.1007/11734727%5C_5.

[Bur+10] Martin Burkhart et al. "SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics". In: *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. USENIX Association, 2010, pp. 223–240. URL: http://www.usenix.org/events/sec10/tech/full%5C_papers/Burkhart.pdf.

[Cha+21] Nishanth Chandran et al. *Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI*. Cryptology ePrint Archive, Report 2021/172. https://ia.cr/2021/172. 2021.

[Cha88] David Chaum. "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability". In: *J. Cryptol.* 1.1 (1988), pp. 65–75. DOI: 10.1007/BF00206326. URL: https://doi.org/10.1007/BF00206326.

[CJS12] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. "Multi-Party Privacy-Preserving Set Intersection with Quasi-Linear Complexity". In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 95-A.8 (2012), pp. 1366–1378. DOI: 10.1587/transfun.E95.A.1366. URL: https://doi.org/10.1587/transfun.E95.A.1366.

[Col21] Yann Collet. *xxHash*. 2021. URL: https://cyan4973.github.io/xxHash/.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 789–800. DOI: 10.1145/2508859.2516701. URL: https://doi.org/10.1145/2508859.2516701.

[Deb+21a] Sumit Kumar Debnath et al. "Post-quantum secure multi-party private set-intersection in star network topology". In: *J. Inf. Secur. Appl.* 58 (2021), p. 102731. DOI: 10.1016/j.jisa.2020.102731. URL: https://doi.org/10.1016/j.jisa.2020.102731.

[Deb+21b] Sumit Kumar Debnath et al. "Secure and efficient multiparty private set intersection cardinality". In: *Adv. Math. Commun.* 15.2 (2021), pp. 365–386. DOI: 10.3934/amc.2020071. URL: https://doi.org/10.3934/amc.2020071.

[ElG84] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2. URL: https://doi.org/10.1007/3-540-39568-7%5C_2.

[Fri07] Keith B. Frikken. "Privacy-Preserving Set Union". In: *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. Lecture Notes in Computer Science. Springer, 2007, pp. 237–252. DOI: 10.1007/978-3-540-72738-5_16. URL: https://doi.org/10.1007/978-3-540-72738-5%5C_16.

[GG10] Ashish Goel and Pankaj Gupta. "Small subset queries and bloom filters using ternary associative memories, with applications". In: *SIGMETRICS 2010, Proceedings of the 2010 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, New York, New York, USA, 14-18 June 2010*. Ed. by Vishal Misra,

Paul Barford, and Mark S. Squillante. ACM, 2010, pp. 143–154. DOI: 10.1145/1811039.1811056.

[Ham15] Mike Hamburg. "Decaf: Eliminating Cofactors Through Point Compression". In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*. Ed. by Rosario Gennaro and Matthew Robshaw. Vol. 9215. Lecture Notes in Computer Science. Springer, 2015, pp. 705–723. DOI: 10.1007/978-3-662-47989-6_34. URL: https://doi.org/10.1007/978-3-662-47989-6%5C_34.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. "Scalable Multi-party Private Set-Intersection". In: *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*. Ed. by Serge Fehr. Vol. 10174. Lecture Notes in Computer Science. Springer, 2017, pp. 175–203. DOI: 10.1007/978-3-662-54365-8_8. URL: https://doi.org/10.1007/978-3-662-54365-8%5C_8.

[HZ06] Feng Hao and Piotr Zielinski. "A 2-Round Anonymous Veto Protocol". In: *Security Protocols, 14th International Workshop, Cambridge, UK, March 27-29, 2006, Revised Selected Papers*. Ed. by Bruce Christianson et al. Vol. 5087. Lecture Notes in Computer Science. Springer, 2006, pp. 202–211. DOI: 10.1007/978-3-642-04904-0_28. URL: https://doi.org/10.1007/978-3-642-04904-0%5C_28.

[IOP18] Roi Inbar, Eran Omri, and Benny Pinkas. "Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters". In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. Ed. by Dario Catalano and Roberto De Prisco. Vol. 11035. Lecture Notes in Computer Science. Springer, 2018, pp. 235–252. DOI: 10.1007/978-3-319-98113-0_13. URL: https://doi.org/10.1007/978-3-319-98113-0%5C_13.

[Ker12] Florian Kerschbaum. "Outsourced private set intersection using homomorphic encryption". In: *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*. Ed. by Heung Youl Youm and Yoojae Won. ACM, 2012, pp. 85–86. DOI: 10.1145/2414456.2414506. URL: https://doi.org/10.1145/2414456.2414506.

[Kol+17] Vladimir Kolesnikov et al. "Practical Multi-party Private Set Intersection from Symmetric-Key Techniques". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 1257–1272. DOI: 10.1145/3133956.3134065. URL: https://doi.org/10.1145/3133956.3134065.

[KS05] Lea Kissner and Dawn Xiaodong Song. "Privacy-Preserving Set Operations". In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, 2005, pp. 241–257. DOI: 10.1007/11535218_15. URL: https://doi.org/10.1007/11535218%5C_15.

[KY03] Aggelos Kiayias and Moti Yung. "Non-interactive Zero-Sharing with Applications to Private Distributed Decision Making". In: *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers*. Ed. by Rebecca N. Wright. Vol. 2742. Lecture Notes in Computer Science. Springer, 2003, pp. 303–320. DOI: 10.1007/978-3-540-45126-6_22. URL: https://doi.org/10.1007/978-3-540-45126-6%5C_22.

[Lin17] Yehuda Lindell. "How to Simulate It - A Tutorial on the Simulation Proof Technique". In: *Tutorials on the Foundations of Cryptography*. Ed. by Yehuda Lindell. Springer International Publishing, 2017, pp. 277–346. DOI: 10.1007/978-3-319-57048-8_6. URL: https://doi.org/10.1007/978-3-319-57048-8%5C_6.

[LW07] Ronghua Li and Chuankun Wu. "An Unconditionally Secure Protocol for Multi-Party Set Intersection". In: *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. Lecture Notes in Computer Science. Springer, 2007, pp. 226–236. DOI: 10.1007/978-3-540-72738-5_15. URL: https://doi.org/10.1007/978-3-540-72738-5%5C_15.

[MN15] Atsuko Miyaji and Shohei Nishida. "A Scalable Multiparty Private Set Intersection". In: *Network and System Security - 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings*. Ed. by Meikang Qiu et al. Vol. 9408. Lecture Notes in Computer Science. Springer, 2015, pp. 376–385. DOI: 10.1007/978-3-319-25645-0_26. URL: https://doi.org/10.1007/978-3-319-25645-0%5C_26.

[NTY21] Ofri Nevo, Ni Trieu, and Avishay Yanai. *Simple, Fast Malicious Multiparty Private Set Intersection*. Cryptology ePrint Archive, Report 2021/1221. https://ia.cr/2021/1221. 2021.

[PSN10] Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl. "Cardinality estimation and dynamic length adaptation for Bloom filters". In: *Distributed Parallel Databases* 28.2-3 (2010), pp. 119–156. DOI: 10.1007/s10619-010-7067-2. URL: https://doi.org/10.1007/s10619-010-7067-2.

[Rua+19] Ou Ruan et al. "New Approach to Set Representation and Practical Private Set-Intersection Protocols". In: *IEEE Access* 7 (2019), pp. 64897–64906. DOI: 10.1109/ACCESS.2019.2917057. URL: https://doi.org/10.1109/ACCESS.2019.2917057.

[SCK12] Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. "Constant-Round Multi-party Private Set Union Using Reversed Laurent Series". In: *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. Lecture Notes in Computer Science. Springer, 2012, pp. 398–412. DOI: 10.1007/978-3-642-30057-8_24. URL: https://doi.org/10.1007/978-3-642-30057-8%5C_24.

[SM18] Katsunari Shishido and Atsuko Miyaji. "Efficient and Quasi-accurate Multiparty Private Set Union". In: *2018 IEEE International Conference on Smart Computing, SMARTCOMP 2018, Taormina, Sicily, Italy, June 18-20, 2018*. IEEE Computer Society, 2018, pp. 309–314. DOI: 10.1109/SMARTCOMP.2018.00021. URL: https://doi.org/10.1109/SMARTCOMP.2018.00021.

[SS09] Yingpeng Sang and Hong Shen. "Efficient and secure protocols for privacy-preserving set operations". In: *ACM Trans. Inf. Syst. Secur.* 13.1 (2009), 9:1–9:35. DOI: 10.1145/1609956.1609965. URL: https://doi.org/10.1145/1609956.1609965.

[VED21] Jelle Vos, Zekeriya Erkin, and Christian Doerr. *Compare Before You Buy: Privacy-Preserving Selection of Threat Intelligence Providers*. Cryptology ePrint Archive, Report 2021/1260. https://ia.cr/2021/1260. 2021.

[VLA21a] Henry de Valence, Isis Lovecruft, and Tony Arcieri. *Testing Equality*. 2021. URL: https://ristretto.group/formulas/equality.html.

[VLA21b] Henry de Valence, Isis Lovecruft, and Tony Arcieri. *The Ristretto Group*. 2021. URL: https://ristretto.group/why_ristretto.html.

3.A Complexities of MPSI protocols

In this section we provide our reasoning for the altered complexities in Table 3.1. We use the notation from Table 3.2. In the final complexities we substitute m with $O(k)$, as explained in Section 3.6.

3.A.I Kissner & Song [KS05]

The authors already provide a computational and communication complexity but we are interested in the complexity per party rather than the total complexity.

1. Each party sends their encrypted polynomial to t other parties, which takes $O(tk)$ bits.
2. Each party sends another encrypted polynomial to one other party, which takes $O(k)$ bits.
3. The leader sends the final encrypted polynomial to all other parties, which takes $O(nk)$ bits.
4. Each party participates in a group decryption (for each coefficient) by sending the their decrypted shares to t other parties, which takes $O(tk)$ bits.

The final communication complexity is $O(nk)$ for the leader and $O(tk)$ for an assistant.

1. Each party generates an encrypted polynomial, which takes $O(k)$.
2. Each party homomorphically multiplies $t + 1$ polynomials, which takes $O(tk^2)$.
3. Each assistant adds two encrypted polynomials together, which takes $O(k)$.
4. Each party participates in a group decryption (for each coefficient), which takes $O(tk)$.

This leads to a computational complexity of $O(tk^2)$ for both the leader and an assistant.

3.A.II Hazay et al. [HV17]

1. Each assistant encodes their set as encrypted polynomial coefficients, which takes $O(k)$.
2. The leader evaluates the $n - 1$ encrypted polynomials with its k elements, which takes $O(nk^2)$.
3. The leader sums up the $n - 1$ ciphertexts per element, which takes $O(nk)$.
4. Several assistants help in the decrypt-to-zero of k ciphertexts, which takes $O(k)$.
5. The leader combines the resulting shares to compute the final intersection, which takes $O(k)$.

So the computational complexity for the leader is $O(nk^2)$ and for an assistant is $O(k)$. At the cost of bandwidth the authors also propose a computation optimization which takes the leader only $O(nk \log_2 k)$.

3.A.III Inbar et al. [IOP18]

1. Each party performs an OT interaction with each other party to share an XOR-secret share, receiving a constant number of bits for each bin in the Bloom Filter, which takes $O(nm)$ bits.
2. Each assistant sends its share of the final aggregated Garbled Bloom Filter to the leader, which takes $O(m)$ bits.

This results in a communication complexity of $O(nm)$ for assistants and the leader alike. The original paper reports a complexity of $O(nhk)$, where we pose $O(hk)$ might have been a substitution for $O(m)$.

1. Each party builds a Bloom Filter and a t-shared Garbled Bloom Filter, which takes $O(nm)$.
2. Each party performs an OT interaction with each other party for every bin in the Bloom Filter, which takes approximately $O(nm)$.

3. Each party XORs their received secret shares from the OT interaction, which takes $O(nm)$.
4. The leader XORs their received secret shares, which takes $O(nm)$.

This results in a computational complexity that is also $O(nm)$ for every party.

3.A.IV Bay et al. [Bay+21]

1. Each assistant sends an encrypted bitset, which takes $O(|\mathcal{U}|)$ bits.
2. The leader sends all assistants at most k aggregated bits, which takes $O(k)$ bits.
3. Each assistant partially decrypts at most k aggregated bits, which takes $O(k)$ bits.

This results in a communication complexity of $O(nk)$ for the leader and $O(|\mathcal{U}|)$ for an assistant.

1. Each assistant generates an encrypted bitset, which takes $O(|\mathcal{U}|)$.
2. The leader aggregates the results using homomorphic addition for the bins corresponding to its set elements, which takes $O(nkh)$.
3. Each assistant partially decrypts at most k aggregated bins, which takes $O(k)$.

As a result, the leader performs $O(nkh)$ operations, while the assistants perform $O(|\mathcal{U}|)$ operations.

3.A.V Debnath et al. [Deb+21b] & Bay et al. [Bay+22]

1. Each assistant sends an encrypted Bloom filter, which takes $O(m)$ bits.
2. The leader sends all assistants at most k aggregated bins, which takes $O(k)$ bits.
3. Each assistant partially decrypts at most k aggregated bins, which takes $O(k)$ bits.

This results in a communication complexity of $O(nk)$ for the leader and $O(k)$ for an assistant.

1. Each assistant generates an encrypted Bloom filter, which takes $O(k + m)$.
2. The leader aggregates the results using homomorphic addition for the bins corresponding to its set elements, which takes $O(nkh)$.
3. Each assistant partially decrypts at most k aggregated bins, which takes $O(k)$.

As a result, the leader performs $O(nkh)$ operations, while the assistants perform $O(k)$ operations.

3.B Complexities of MPSU protocols

3.B.I Frikken [Fri07]

Following the author's complexity analysis, the parties each share $O(nk)$ tuples with two ciphertexts in step 2b of the protocol, so the communication complexity for each individual party is $O(nk)$ bits. Each party must be online at least once in steps 1a, 1b, 1c, 2b, 3 and 4, so the protocol requires at least 6 stages.

1. Each party encodes their set as encrypted polynomial coefficients, which takes $O(k)$.
2. Each party $\mathcal{P}_i$ homomorphically multiplies i encrypted polynomials together, which takes at most $O(nk^2)$.
3. Each party $\mathcal{P}_i$ encrypts $2k$ values and homomorphically multiplies $i-1$ encrypted polynomials together, which takes at most $O(nk^2)$.
4. Each party $\mathcal{P}_i$ homomorphically multiplies i ciphertexts together, which takes at most $O(nk)$.
5. Each party takes part in a secure shuffle protocol with at most nk tuples, which we assume to be linear with the number of parties and tuples.
6. All parties work together to decrypt at most nk tuples, which scales linearly with n and k.

So, the overall computational complexity is $O(nk^2)$ for each party.

3.B.II Seo et al. [SCK12]

In this work, p is the order of the finite field used in secret sharing. It needs to hold that $p \leq |\mathcal{U}|$, so the computational complexity becomes $O(n^4k^2 + n^2k^2 \log|\mathcal{U}|)$. For brevity, we omit the logarithmic term: $\tilde{O}(n^4k^2)$.

Chapter 4

Privacy-Preserving Membership Queries for Federated Anomaly Detection

While multi-party private set intersection have received significant attention from the research community, we are not aware of practical deployments. Instead, deployed private set operations often consider versions of private membership queries. For example, to check whether a password has been found in a leaked password file, without revealing the password nor the complete password file. In this chapter, we apply similar elliptic curve-based techniques as we proposed in the previous chapters in the context of private membership queries.

The use case that we consider in this chapter is that of federated anomaly detection. A specific example is when a payment network system that routes payments between banks wants to perform fraud detection without compromising user privacy. We propose privacy-preserving multi-party *repeated* membership queries for this purpose, which allow the payment network system to check whether all the credentials pertaining to a transaction match the user data stored by both banks, without revealing the query or the user data to the other parties. Our protocol is strictly in the star topology and requires only two rounds for each query, addressing impracticalities 1: High interactivity and 2: Full-mesh topology.

This chapter is an adaptation of the work with the same title that has been published in Proceedings on Privacy Enhancing Technologies 2024, authored by Jelle Vos, Sikha Pentyala, Steven Golob, Ricardo Maia, Dean Kelley, Zekeriya Erkin, Martine De Cock, and Anderson Nascimento. This work was originally a submission to the U.K.-U.S. PETs prize challenge, where it won the 2nd place.

4.1 Introduction

Privacy-preserving membership query protocols allow a centralized entity $\mathcal{S}$ to find out whether a given element is contained in the data sets of other parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ without learning anything about the other elements in those sets. In the multi-party case, such a membership query will only return positively when the queried element is contained in the set of each other party. This work proposes a protocol to perform privacy-preserving membership queries that is particularly efficient when performing multiple parallel and sequential queries on the same sets.

We refer to such queries as repeated membership queries. The key to making this efficient is to use encrypted membership query filters (EMQFs), see Section 4.5.1.

Repeated privacy-preserving membership queries occur in large-scale systems, which are often comprised of many user-facing entities and one or a few centralized entities serving as the backbone. Such federations arise for various reasons: to increase scalability, for political and operational reasons, or simply because that is how these systems functioned historically, and reorganizations are costly. Examples include (1) financial systems such as the global payment system orchestrated by SWIFT and distributed among banks, (2) governmental systems such as tax authorities controlled by national governments but supported by states or municipalities, and (3) health care guided by insurers while provided by many medical institutions. All these systems are prone to fraud because the centralized entities have little knowledge of the users, while the other entities do not have a global view of the system. Typical approaches that address fraud in such a federated setting come at the cost of the users' privacy, and not addressing it is not an option due to its high societal cost. Privacy-preserving membership queries allow us to address fraud while preserving users' privacy.

We demonstrate the effectiveness of our solution in the financial domain. As illustrated in Fig. 4.1, we assume a cross-silo federated architecture in which a *centralized entity* (the backbone) has labeled data to train a machine learning (ML) model for detecting anomalous instances. The other entities in the federation are *data-augmenting entities* (the user-facing entities), which collaborate with the centralized entity to extract feature values to improve the utility of the model. In the financial domain, the centralized entity would, for instance, be a payment network system like SWIFT that holds information about financial transactions, and the data-augmenting entities are partner banks that hold additional information about the ordering and beneficiary accounts appearing in financial transactions. The task is to use the sensitive information residing with all the entities, e.g. the payment network system and the banks, to train a ML model to detect anomalies. A large body of research has already studied ways to train such a model with *output privacy* guarantees: by training the model using differential privacy (DP) techniques, any inferences made with the model can be made public while guaranteeing plausible deniability about the existence of any specific instance in the training data. The challenge is to also provide *input privacy*, which guarantees that the information held by any data-augmenting entities and the information held by any centralized entity is not disclosed to any other entity. We propose a new cryptographic protocol tailored to this setting to efficiently and privately extract a Boolean feature indicating whether the data relating to a specific instance is consistent between the centralized entity and that of one or more data-augmenting entities.

Throughout this paper, we use $\mathcal{S}$ to denote the centralized entity and $\mathcal{P}_1, \ldots, \mathcal{P}_n$ to denote the data-augmenting entities. We assume that $\mathcal{S}$ has a training dataset in which each instance is labeled with a value denoting whether it is anomalous or not. To provide output privacy, $\mathcal{S}$ can train a ML $\mathcal{M}$ over its data with any of a variety of supervised DP model training algorithms that have been proposed in the literature, including for logistic regression, tree ensembles, and neural networks [Aba+16; CM08; CMS11; FI17]. We note that this choice of model is completely free, and we do not expand on it in this work. The emphasis in this paper is on improving the

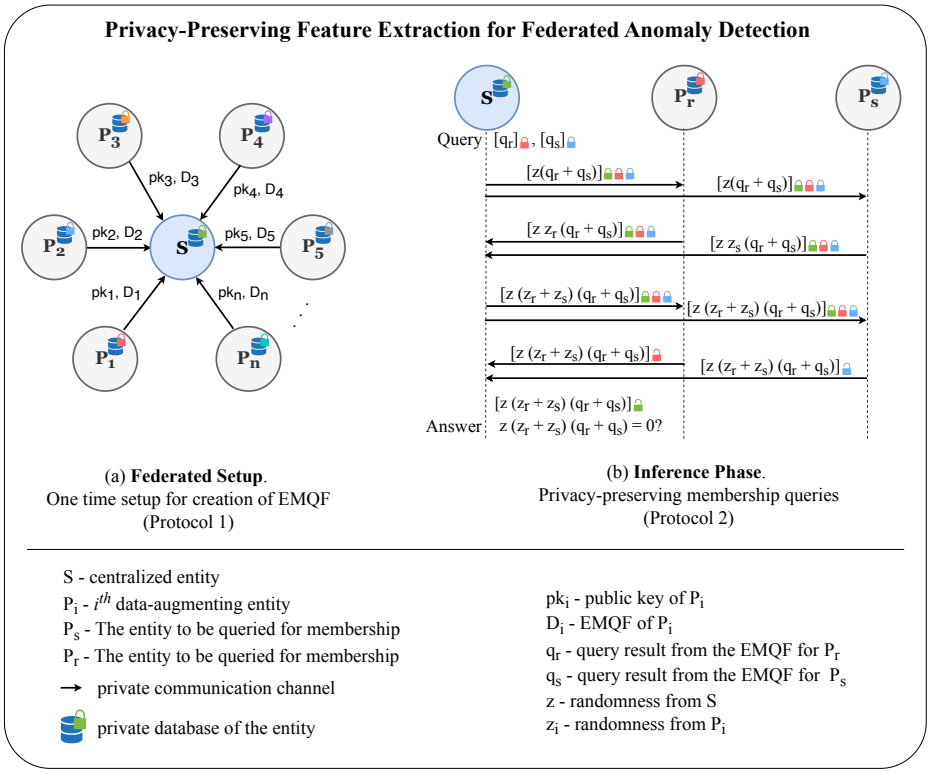


Figure 4.1: (a) Architecture diagram of the federation; (b) A private consistency check involving two data-augmenting clients

utility of $\mathcal{M}$ by extracting feature values that represent whether information held by $\mathcal{S}$ is consistent with information held by one or more of the $\mathcal{P}_i$'s, e.g. whether the information about the ordering and beneficiary accounts listed in a transaction known to a payment network system is consistent with the information known by the respective sending and receiving banks.

From a technical point of view, our private feature extraction protocol is built on top of a novel private set membership protocol that is especially efficient when performing many sequential queries. To make the protocol efficient, we instantiated the cryptosystem over an elliptic curve and implemented it in Rust (with a Python wrapper). The resulting protocol has low computational and communication demands. Moreover, typical protocols for private set membership or private set intersection based on oblivious key-value stores [Gar+21] and built using OT extension protocols [Ash+17] reveal the protocol's output in the clear. This is problematic when the result of the protocol needs to be used in other private computations. We provide an extension of our protocol that overcomes this limitation by outputting ElGamal encryptions of the data and using a custom private equality test that uses its homomorphic properties. We are not aware of similar constructions in the literature.

Motivated by the 2023 PETs prize challenge,[1] which is a collaborative effort of the US and UK governments, we demonstrate the effectiveness of our approach in the financial domain. We show that the private queries significantly increase the precision and recall of the otherwise centralized system and argue that this improvement translates to other use cases as well.

Our contributions can be summarized as follows:

- We propose a new multi-party private set membership protocol that is also of independent interest (e.g., for multi-party private set intersections), in which we formalize the concept of an EMQF.
- We propose a secure equality protocol over elliptic curve-based ElGamal ciphertexts, also of independent interest.
- To the best of our knowledge, we discuss the first concretely efficient distributed anomaly detection system in a federated star topology that preserves both input and output privacy.

After describing related work in Sec. 4.2 and preliminaries in Sec. 4.3, we provide a high level description of our solution in Sec. 4.4. We follow up with a detailed description for private consistency queries in Sec. 4.5 and a privacy proof in Sec. 4.6. In Sec. 4.7 we document experiments that demonstrate the utility and scalability of our method, which was a prize finalist in the 2023 PETs prize challenge.

4.2 Related work

We go over previous works that tackle similar (sub-)problems. First, we discuss previous works for performing private membership queries by studying private set

[1] https://www.drivendata.org/competitions/group/nist-federated-learning/

intersection protocols. After that, we analyze solutions related to our application in the field of privacy-preserving federated learning, which also tackle learning in federations while providing privacy guarantees.

4.2.1 Private Membership Queries

Private membership queries form the basis of private set intersection protocols, which have been studied extensively. In this work, we are interested in the multi-party case, where a querier can check the membership of an element with multiple parties at once and only receive a positive result if each set contains the element. We briefly provide an overview of some of the most recent work on these protocols. We distinguish between multi-party private membership queries and multi-party private set intersections (MPSI). In the latter case, we discuss two categories: protocols based on multiple instances of two-party computation (2PC) and protocols based on homomorphic encryption (HE). Note that while some of the protocols discussed here are efficient when it comes to computation and communication, each individual query has a large cost associated to it. In other words, these protocols are not concretely efficient for performing many queries in parallel or sequentially.

Private set intersections using 2PC This type of MPSI protocols combines multiple executions of two-party computations that inherently perform membership queries to find the intersection of all separate sets. This closely resembles our protocol, except these protocols do not rely on homomorphic encryption. The most recent protocol in this category is that by Nevo et al. [NTY21]. This protocol uses OKVSs encoding secret shares in combination with oblivious transfers to perform the membership checks. A similar approach was proposed before that by Garimella et al. [Gar+21]. These protocols are among the most efficient for multi-party private set intersections over large sets. Another protocol in this category is by Kavousi et al. [KMS21], which relies on oblivious pseudo-random functions (OPRFs) as the two-party computation and functions strictly in the star topology. In this work, each party encodes their set as a garbled Bloom filter. Since OPRFs are efficient building blocks, the Bloom filter can have many bins before computations become prohibitively expensive. Note that in all three works, the party receiving the final output must perform computation and communication with the other parties scaling linearly with their set size every time the protocol is executed.

Private set intersections using HE We discuss the three of the most recent MPSI protocols based on homomorphic encryption. All three protocols function in the star topology, which makes them suitable for in a federation, where there is a centralized entity and multiple other entities. The protocol by Bay et al. [Bay+22], and subsequently by Vos et al. [VCE22] uses encrypted Bloom filters to perform one membership query for each of the elements in the centralized entity's set. While the former uses Paillier encryption, the latter uses elliptic curve-based ElGamal, similar to this work. These protocols are concretely efficient for small set sizes, but they are less efficient than the 2PC-based MPSI protocols as the set

size grows. Hazay et al. [HV17] present a different protocol, in which each party other than the centralized entity represents their set as the roots of an encrypted polynomial. Performing the membership query then involves privately evaluating this polynomial on the queried element. In these three works, the centralized entity may keep the encrypted Bloom filters or encrypted polynomials locally to speed up future queries, but these methods remain efficient. For Bloom filters, this is the case because the filters must be large in size not to introduce false positives, which makes them prohibitively expensive for larger set sizes. For encrypted polynomials, each invocation requires cryptographic operations on every encrypted coefficient, which requires a great deal of computation.

Private membership queries The protocol by Chielle et al. [CGM21] is similar to many HE-based MPSI protocols as it is based on encrypted Bloom filters. However, Bloom filters typically need to grow large in size in order to press the false positive rate. As a result, precise queries involve a large amount of cryptographic operations. This paper uses the BFV leveled-homomorphic encryption scheme. Unfortunately, the protocol leaks information as it exposes all bits of the Bloom filter selected by the hash functions (see [VCE22], Section 6.2). Ramezanian et al. [Ram+20] propose a more complex protocol that decreases this cost by using Cuckoo filters, which are also more common in MPSI protocols. The most recent custom protocol for performing private membership queries is that by Garg et al. [Gar+23]. This protocol provides stronger security than regular membership queries because it is hard for the party holding the queried set to fool the other party of a positive query result. However, this protocol is restricted to two parties.

In this work, we are interested in performing multiple sequential queries on the same sets. We refer to a protocol that is designed for this case as a privacy-preserving repeated membership query protocol. MPSI protocols can be naively turned into such repeated membership query protocols, but this is highly inefficient: While an MPSI protocol allows performing multiple membership queries in parallel, they are not necessarily efficient for performing multiple queries sequentially. We provide an overview incorporating the number of sequential queries as q in Table 4.1. Here, k represents the maximum size of the parties' sets and n is the number of involved parties. We also use t to denote the number of parties that can collude before the security guarantees no longer hold, and h to denote the number of hash functions in a Bloom filter.

In Table 4.1, we give two examples of MPSI protocols (Hazay et al. [HV17] and Vos et al. [VCE22], see Chapter 3) that can be easily adapted (indicated with an asterisk) so that parties only have to encode and send their set once. Their asymptotic communication complexities are identical to the protocol presented in this work. In fact, these adapted protocols can all be seen as instantiations of our protocol with a different EMQF. However, encrypted polynomials and Bloom filters are significantly larger in size and require orders of magnitude more computations to both encode and decode. We provide a concrete comparison of this in Sec. 4.7. Note that the extra factor n incurred in the computation of an assistant is because our work uses a different setup; the involved parties do not share a single public key.

Our solution enjoys the benefits of the HE-based MPSI protocols we discussed, running in the star topology and not having to recompute the set representation at every invocation, along with the benefits of the 2PC-based MPSI protocols, which scale well with the size of the sets. By using elliptic curve-based ElGamal rather than a partially homomorphic encryption scheme such as Paillier, the size of the OKVS stays compact, the cryptographic operations are fast to execute, and the bandwidth cost in subsequent membership queries decreases by an order of magnitude.

Table 4.1: Comparison of our privacy-preserving membership query protocol when expressed as a repeated multi-party private set intersection, derived from [VCE24].

Work	**Communication**			**Computation**	
Authors	Leader	Assistant	Rounds	Leader	Assistant
Hazay [HV17]	$O(qnk)$	$O(qk)$	$4q$	$O(qnk)$	$O(qk)$
Inbar [IOP18]	$O(qnkh)$	$O(qnkh)$	$3q$	$O(qnkh)$	$O(qnkh)$
Ghosh [GN19]	$O(qnk)$	$O(qk)$	$6q$	$O(qnk)$	$O(qk)$
Chandran [Cha+21]	$O(qnk)$	$O(qk)$	$8q$	$O(qnk)$	$O(qk)$
Garimella [Gar+21]	$O(qnk)$	$O(qk)$	$4q$	$O(qk)$	$O(qnk)$
Gordon [GHL22]	$O(qnk + qnt)$	—	$5q$	—	—
Vos [VCE22] (Ch. 3)	$O(qtk)$	$O(qk)$	$3q$	$O(qnkh)$	$O(qk)$
Hazay* [HV17]	$O(qt)$	$O(k + q)$	$1 + 2q$	$O(qnk)$	$O(qk)$
Vos* [VCE22] (Ch. 3)	$O(qt)$	$O(k + q)$	$1 + 2q$	$O(qnkh)$	$O(qk)$
This work	$O(qt)$	$O(k + q)$	$1 + 2q$	$O(qnk)$	$O(qnk)$

4.2.2 Privacy-preserving federated learning

Federated learning (FL) [McM+17] has emerged as a popular paradigm to train global machine learning models over data held by multiple entities, which are typically referred to as clients. Nearly all state-of-the-art FL algorithms and applications assume scenarios in which the data is horizontally partitioned, i.e. each client holds one or more instances. In the cross-silo federated architecture that we consider (see Fig. 4.1), the data is split both horizontally and vertically, making the mainstream FL paradigm difficult to use and analyze [Kai+21]. Orthogonal to this, it is also well understood that FL is not privacy-preserving by default, as information about the clients' training data may leak from the gradients or model parameters (see e.g. [Kai+21; Boe+21; So+21; Elk+22]). Existing works that use a combination of privacy-enhancing technologies in the context of FL to provide end-to-end privacy address, to the best of our knowledge, only the scenario of horizontally distributed data [Jay+18; PRR10; Cha+17; BP20; Tru+19; GLX21].

A relevant technique for cross-silo FL is secure multi-party computation (MPC), an umbrella term for cryptographic approaches that allow two or more parties to jointly compute a specified output from their private information in a distributed fashion without revealing this private information to each other [CDN15]. While MPC typically comes with a substantial computation and communication overhead,

a major advantage is that it can be readily applied to scenarios where the data is partitioned horizontally, vertically, or in any other mixed way. Nearly all of the MPC protocols proposed in the literature for model training (e.g. [Ada+22; Aga+19; MZ17; WGC19; De +21; Guo+20; KS22]) however only protect *input privacy*, i.e. they enable training of a model over data that is distributed among data holders without requiring those data holders to disclose the data. In isolation, these approaches do not provide sufficient protection if the trained model is to be made publicly known, or even if it is only made available for closed-box query access, because information about the model and its training data is leaked through the ability to query the model (see e.g. [FJR15; Tra+16; SRS17; Car+19]). Formal privacy guarantees, in this case, can be provided by differential privacy [DR+14]. It is, however, well known that local differential privacy, where each client adds noise to their data before sending it out, causes severe utility loss. In contrast, global differential privacy, where each client sends its data to a trusted curator responsible for adding the noise, introduces a single point of failure (namely the curator) and violates input privacy.

Input and output privacy can simultaneously be achieved in FL by combining MPC with differential privacy by replacing the trusted curator from the global DP paradigm with an MPC protocol (run across multiple mutually distrustful parties) to generate noise to perturb the model parameters, providing differential privacy guarantees. Existing methods leveraging this idea can handle data that is arbitrarily partitioned. However, such solutions are not directly applicable to our scenario because they assume that all feature values are readily available in the federation [Pen+22]. In our case, however, some features carry a high signal but have yet to be constructed by combining information from the centralized and data-augmenting entities.

While using a general-purpose MPC protocol for such joint feature extraction provides the necessary privacy guarantees, it requires significant computation and communication since MPC generic solutions for such tasks heavily depend on the circuit size (which is very high for our specific joint feature extraction task).

This work proposes a custom cryptographic protocol based on elliptic curve-based homomorphic ElGamal and oblivious key-value stores (OKVS) [Gar+21]. With almost all of the attention in the privacy-preserving machine learning (PPML) literature going to the model training phase, our proposal fills a critical gap concerning data preprocessing, namely private feature extraction. A recent work by Kadhe et al. [Kad+23] uses generic MPC and specifically avoids this private feature extraction step. This makes their protocol significantly more expensive in computation and communication.

We discuss three more recent works that tackle similar problems. Asif et al. [Asi+23] use a similar method to ours, where $\mathcal{S}$ performs membership queries on $\mathcal{P}_i$'s data, but the data is encoded as unencrypted Bloom filters. The authors wrongfully assume that this provides privacy to the users whose data is encoded. This form of information leakage is discussed in the work by Vos et al. ([VCE22], Sec. 4.2.1). The work titled HyFL [Zha+23] solves the same problem but in a weaker threat model, where data is only encrypted against outside attackers, but $\mathcal{S}$ decrypts $\mathcal{P}_i$'s data and trains on it in plain text. Finally, a recent work titled Starlit [Aba+24] extracts a similar feature as in our work, except that it is computed

by performing a one-time private set intersection between the $\mathcal{S}$'s data and $\mathcal{P}_s$ and $\mathcal{P}_r$'s data. This reveals significantly more private information than our protocol: $\mathcal{S}$ learns the result of performing membership queries with each row in their dataset on each individual $\mathcal{P}_i$'s dataset. This is essentially the worst-case leakage of our protocol.

4.3 Preliminaries

We briefly discuss cryptographic primitives used in our protocols and introduce the definition of differential privacy.

Oblivious key-value stores Conceptually, an OKVS is a dictionary that outputs a random-looking value for each key. If the key was encoded in the OKVS, then the value always corresponds to the value that went with it. We choose to work with the PaXoS [Pin+20] OKVS with the xxh3 statistical hash function [Col21]. Note that this hash function does not have to be cryptographically secure, as the security of OKVSs only relies on the statistical indistinguishability of the encoded values. The OKVS function $\mathsf{decode}(D, q)$ returns the value corresponding to the key q in OKVS D. Regardless if the key was encoded in the OKVS, the properties of the OKVS ensure that the returned value is indistinguishable from randomness.

Curve25519 In our protocols, we work over Curve25519 [Ber06]. We denote the identity by O, the scalar group by $\mathbb{Z}_q$, and the curve group by $E(\mathbb{Z}_q)$. Here, $q = 2^{255} - 19$ and $|E(\mathbb{Z}_q)| > 2^{252}$.

All parties have access to a generator G. While Curve25519 is defined as a Montgomery curve, it is birationally equivalent to a twisted Edwards curve. Unless specified, we use the twisted Edwards model. We also introduce two functions $\mathsf{ToMontgomery}(x)$ and $\mathsf{ToEdwards}(x, s)$, which switch x between the two curve models. Here, s denotes the sign of the twisted Edwards point, as the Montgomery point only contains the X-coordinate of the point.

Elligator maps The main challenge when encoding curve points in an OKVS is that the OKVS requires those points to be indistinguishable from random bits. The typical compressed representation of curve points certainly does not satisfy this requirement. For example a compressed Montgomery point is simply its X coordinate, which must satisfy the strict curve equation. As a result, not every set of random bits is interpretable as the scalar representation of X. Instead, we use the Elligator2 map [Ber+13] to map between random-looking bits and curve points. Consequently, we work over Montgomery and twisted Edwards curves. We use the Montgomery model to apply the Elligator2 map given by the function $\psi : \mathbb{Z}_q \mapsto E(\mathbb{Z}_q)$ and its inverse ψ^{-1}. The inverse only returns a representative for half of the points in $E(\mathbb{Z}_q)$; otherwise, it returns $\perp$.

Differential privacy A randomized algorithm $\mathcal{F}$ provides (ϵ, δ)-DP if for all pairs of neighboring datasets D and D' (i.e. datasets that differ in one entity), and

for all subsets S of $\mathcal{F}$'s range:

$$\mathrm{P}(\mathcal{F}(D) \in S) \leq e^{\epsilon} \cdot \mathrm{P}(\mathcal{F}(D') \in S) + \delta \quad \text{[DR+14]}. \tag{4.1}$$

The parameter $\epsilon \geq 0$ denotes the *privacy budget* or privacy loss, while $\delta \geq 0$ denotes the probability of violation of privacy, with smaller values indicating stronger privacy guarantees in both cases. In our context, D and D' are datasets with instances, and $\mathcal{F}$ is an algorithm to induce an ML model from a dataset, or a method to compute a statistic (like the mean) over a dataset. An DP algorithm $\mathcal{F}$ is usually created out of an algorithm $\mathcal{F}^*$ by adding noise that is proportional to the sensitivity of $\mathcal{F}^*$, in which the sensitivity measures the maximum impact a change in the underlying dataset can have on the output of $\mathcal{F}^*$.

4.4 Solution outline

4.4.1 Threat model

We model all parties as polynomial-time Turing machines (PPT).

For the elliptic curve cryptography, we assume the decisional Diffie-Helman problem to hold over Curve25519 [Ber06] and Elligator2 [Ber+13] to be statistically indistinguishable from randomness.

We implement authenticated and private communication channels between the centralized entity $\mathcal{S}$ and the data-augmenting clients $\mathcal{P}_1, \ldots, \mathcal{P}_n$ by using the authenticated encryption mode of operation EAX and AES, which we treat as a pseudorandom function. To do so, we predistribute symmetric keys among the respective parties. Alternatively, the parties can run Diffie-Helman key exchange protocols to establish such common keys.

We assume the adversaries to be honest-but-curious. That means they follow the protocol specifications, but try to obtain as much information as possible about private information from their inputs, messages exchanged, and internal randomness. Our protocols (Sec. 4.4.3) are designed to prevent adversaries from learning such information (see Sec. 4.6 for the proofs). We work with static adversaries. Our solutions can be generalized to stronger adversarial models (fully malicious/active adversaries) at the cost of having reduced efficiency. See Appendix 4.B for an initial discussion.

In our solution, the centralized entity $\mathcal{S}$'s training data never leaves $\mathcal{S}$, not even in encrypted form. The privacy of the centralized entity's data is guaranteed even if the result of the classification (the predicted probability that an instance is anomalous) is made public. This privacy guarantee follows directly from the fact that the model is generated from the data with an algorithm that provides differential privacy (DP) guarantees (Sec. 4.4.2). This means that the probability that the algorithm generates a specific model from the data is very similar to the probability of generating that model if a particular instance had been left out of the data. The latter implies that what the model has memorized about individual instances is negligible. Obviously, if the result of the classification is not made public by $\mathcal{S}$, no information whatsoever leaks about the centralized entity's data (a result that follows from our secure distributed feature extraction protocol in Sec. 4.5).

In our solution, data never leaves the data-augmenting clients in plaintext form. In the feature extraction protocol in Sec. 4.5, the data-augmenting clients encrypt their data as ElGamal ciphertexts and encode the ciphertexts in oblivious key-value stores (OKVS), which they send to $\mathcal{S}$. The centralized entity $\mathcal{S}$ and the data-augmenting clients perform computations over this data while it is stays encrypted. At the end of the protocol, (1) the centralized entity and the data-augmenting clients can jointly decrypt the result (Sec. 4.4.3) and open it to $\mathcal{S}$, or (2) use a protocol extension to compute linear functions (such as generalized linear machine learning models) on the encrypted data. We use the former approach, so $\mathcal{S}$ learns one bit of information, which is 1 if any inconsistency is detected.

We do not consider side channel attacks in our proposal, but Protocols 6 and 8 have been designed using constant-time primitives, and the only variable-time operations do not reveal information about the inputs. However, we do not investigate the security of our solution against these attacks, and give no guarantees about our current implementation's resistance to them.

Moreover, our threat model does not take into account model inversion attacks. We perceive this as a separate issue, requiring its own countermeasures, as highlighted by [FJR15].

4.4.2 Model training

We recall that the entities in our solution are the centralized entity $\mathcal{S}$, and the data-augmenting entities $\mathcal{P}_1, \ldots, \mathcal{P}_n$. $\mathcal{S}$ has a training dataset in which each instance is labeled whether it is an anomalous instance or not. $\mathcal{S}$ trains a classifier $\mathcal{M}$ over this training data. For a query instance $\boldsymbol{x}$, i.e. a new instance that needs to be classified as anomalous or not, the model $\mathcal{M}$ outputs a predicted probability $\mathcal{M}(\boldsymbol{x}) \in [0, 1]$ that the instance is anomalous.

To prevent leakage of information from the predicted probabilities about the instances in the training data, we use an ML model training algorithm that provides differential privacy. In this way, our solution provides formal guarantees that the trained model $\mathcal{M}$, and hence predictions made with $\mathcal{M}$, are negligibly affected by the inclusion of any particular instance in the training data, thereby offering output privacy through plausible deniability [DR+14]. A variety of (ϵ, δ)-DP ML model training algorithms have been proposed in the literature, including for logistic regression, tree ensembles, and neural networks [Aba+16; CM08; CMS11; FI17]. Our overall solution is general enough to allow for any (ϵ, δ)-DP ML model training algorithm to be used by $\mathcal{S}$ to train its model $\mathcal{M}$. In Sec. 4.7, we compare the performance of several DP model training algorithms.

4.4.3 Inference

During inference, $\mathcal{S}$ has to infer the probability to which each new instance $\boldsymbol{x}$ is anomalous. Our solution leverages information held by the data-augmenting entities $\mathcal{P}_1, \ldots, \mathcal{P}_n$ to improve the accuracy of the predictions made by model $\mathcal{M}$. To this end, $\mathcal{S}$ and the data-augmenting entities work together to perform a consistency check that yields $B(\boldsymbol{x}) = 1$ if there is any inconsistency between

fields from x as known to $\mathcal{S}$ versus as known by the data-augmenting entities, and $B(\boldsymbol{x}) = 0$ if everything is consistent.

In the financial domain, $\mathcal{S}$ could be a payment network system interacting with banks $\mathcal{P}_1, \ldots, \mathcal{P}_n$ to check the validity of a financial transaction $\boldsymbol{x}$. The consistency check in this case would involve verifying that the names and addresses of the sender and receiver accounts in the transaction $\boldsymbol{x}$ as known to $\mathcal{S}$ match the names and addresses of the account holders as known by the sending bank $\mathcal{P}_s$ and receiving bank $\mathcal{P}_r$. In Sec. 4.5 we describe a protocol for performing such a consistency check without requiring $\mathcal{S}$, $\mathcal{P}_s$, or $\mathcal{P}_r$ to disclose their data to each other in an unencrypted manner. At the end of the protocol, we open the value of $B(\boldsymbol{x})$ to $\mathcal{S}$. We note that in this way, $\mathcal{S}$ learns only whether there was an inconsistency or not, and not from which field in $\boldsymbol{x}$ or from which data-augmenting client an inconsistency originated. We also note that the protocol in Sec. 4.5 is generic and supports any number of data-augmenting entities in a single query and not just two as illustrated in the example above. Moreover, the protocol in Sec. 4.5 supports any disjunction of equality constraints, i.e. it applies to any situation where one must check whether a database contains an entry (such as a single field or a combination of fields) that matches exactly.

The final inference result for $\boldsymbol{x}$ is computed by $\mathcal{S}$ as $\max(\mathcal{M}(\boldsymbol{x}), B(\boldsymbol{x}))$, in which $\mathcal{M}(\boldsymbol{x})$ denotes the probability predicted by the model trained by a DP algorithm (see Sec. 4.4.2) and $B(\boldsymbol{x})$ denotes the Boolean consistency feature jointly extracted by $\mathcal{S}$ and the data-augmenting clients. In the next section, we explain how to compute $B(\boldsymbol{x})$ in a privacy-preserving manner.

4.5 Private consistency queries

The underlying functionality of the private consistency queries is that of a private membership query. After all, we are testing whether each involved data-augmenting entity has a row in their dataset that matches our query. Suppose first that such a consistency query only involves the centralized entity $\mathcal{S}$ and one data-augmenting entity $\mathcal{P}_i$. It is easy to see that we can perform private consistency queries if we can realize a protocol that returns an encryption of zero (or rather the additive identity O) if the membership check passes and randomness otherwise. We refer to this primitive as an encrypted membership query filter. The key insight in our protocol is that we can let each data-augmenting entity perform an expensive one-time setup that scales linearly with the size of their dataset, after which queries take a significantly shorter time.

4.5.1 Encrypted membership query filters

We introduce encrypted membership query filters (EMQFs), which are non-interactive encrypted filters for querying set membership. When queried, with high likelihood, an EMQF only returns an encryption of the identity when the element is encoded in it. It implements and satisfies the following functions and properties:

- Build$(pk, R) \mapsto$ EMQF: Constructs an EMQF encoding R using encryption key pk.
- QueryEMQF, $q \mapsto C$: Queries the EMQF on element q returning a ciphertext.

Correctness We require that $\Pr[\text{Dec}(\text{QueryEMQF}, q, sk) \neq O \mid q \in R] \leq \mu$ and $\Pr[\text{Dec}(\text{QueryEMQF}, q, sk) = O \mid q \notin R] \leq \mu$, where sk is the secret key for the pk used to build the EMQF and μ denotes a negligible value.

Privacy It must hold that $\text{Build}(pk, R) \overset{c}{\equiv} \text{Build}(pk, R')$ given $|R| = |R'|$.

A naive way to instantiate an EMQF using partially-homomorphic encryption is to encrypt every row of the set. Then, for a query, privately subtract every row with the query. However, this requires returning more than one ciphertext. Hazay et al. [HV17] implicitly provide an instantiation by encoding set elements as the roots of a compact but expensive-to-query encrypted polynomial and Vos et al. [VCE22] encode the set as a large but cheap-to-query encrypted inverted Bloom filter. Both use partially-homomorphic encryption.

The idea of our paper is to encode a set as an oblivious key-value store (OKVS), which is both compact and cheap to query. We choose to work over an elliptic curve group to speed up computation, decrease the key size, and to minimize the size of the OKVS. As explained next, encoding these ciphertexts is not straightforward.

4.5.2 Encoding ciphertexts in the OKVS

The security of the oblivious key-value stores depends entirely on the requirement that its values are computationally indistinguishable from randomness. Note, however, that elliptic curve points by default are easily distinguishable from a uniformly random bitstring by checking whether it conforms to the curve equations. Instead, we rely on the Elligator2 map denoted by ψ to encode and decode curve points. In Algorithm 4 and 5 we show how to use this mapping to represent a point as a bitstring and vice versa. We refer to Sec. 4.3 for a description of the building blocks.

Algorithm 4 Encodes a twisted Edwards point p in 32 bytes indistinguishable from randomness.

```
1: procedure ToBytes(p)
2:     p_E ← ToMontgomery(p)
3:     s ∈_R {+, −}                          ▷ Choose + or - representative for p
4:     B ∈ ψ^{-1}(p_E, s)
5:     if B = ⊥ return ⊥
6:     if FromBytes(B) = p return B
7:     B[31] ← B ∨ 128                       ▷ Encode the sign in the MSB
8:     return B
```

A data-augmenting entity can now safely generate an OKVS that realizes an EMQF using Protocol 6, where steps 2 and 3 represent the Build(pk, R) function. The EMQF's privacy is implied by the OKVS's obliviousness property. This protocol also generates a key pair. After that, the data-augmenting entity sends the OKVS

Algorithm 5 Decodes a twisted Edwards point from 32 bytes B.

```
1: procedure FromBytes(B)
2:     s ← B[31] ≫ 7                ▷ Extract the sign from the MSB
3:     return ToEdwards(ψ(B), s)
```

and the public key to $\mathcal{S}$. The centralized entity generates a keypair in the same way, but it does not need to generate an OKVS.

Input: Set R_i containing the rows of $\mathcal{P}_i$'s database.
Output: Public key $pk_i \in E(\mathbb{Z}_q)$, secret key $sk_i \in \mathbb{Z}_q$, OKVS D_i.

1. $\mathcal{P}_i$ randomly generates $sk_i \in_R \mathbb{Z}_q$ and computes $pk_i \leftarrow sk_i\, G \in E(\mathbb{Z}_q)$.
2. $\mathcal{P}_i$ generates $v_j \leftarrow \mathsf{ToBytes}(r_j\, G) \,||\, \mathsf{ToBytes}(r_j\, pk_i)$ where $r_j \in_R \mathbb{Z}_q$ for $j = 1, \ldots, |R_i|$. If ToBytes returns $\perp$, resample.
3. $\mathcal{P}_i$ encodes OKVS D_i where the keys are the rows in R_i, and the values are v_j for $j = 1, \ldots, |R_i|$.
4. $\mathcal{P}_i$ sends pk_i and D_i to $\mathcal{S}$.

Protocol 6: One-time setup for each data-augmenting entity: generating keys and an OKVS.

4.5.3 Performing a query

Next, we explain how the centralized entity can perform a membership query with multiple data-augmenting entities. We present this in Protocol 7 for the case where each query involves two data-augmenting entities, but we note that the protocol can easily be extended to an arbitrary number of entities. This case involving one centralized entity and two data-augmenting entities resembles that of a federation where the centralized entity routes transactions between two data-augmenting entities.

Protocol 7 uses a variant of ElGamal ciphertexts to perform arithmetic under encryption. The goal of this protocol is for the centralized party to learn whether element q_i is included in both the sender's and receiver's EMQF. Putting encryption aside, the protocol starts by querying the sender's $\mathcal{P}_s$ and receiver's $\mathcal{P}_r$ EMQF on q_i, which is as simple as decoding and calling FromBytes on the OKVS in our OKVS-based EMQF. The EMQF of $\mathcal{P}_i$ outputs an encryption of the identity O when the query matches a row in $\mathcal{P}_i$'s database, otherwise the resulting ciphertext encrypts a random curve point. In steps 2–4 of the protocol, the centralized entity $\mathcal{S}$ computes the sum of the OKVS outputs, lets $\mathcal{P}_s$ and $\mathcal{P}_r$ multiply the result by a random scalar, and then sums those results again. Note that if both OKVS output encrypted the identity O, then the result of these steps is still O. In steps 5–6, the entities collaboratively decrypt the result such that only $\mathcal{S}$ receives the output.

Protocol 7 does not use standard ElGamal ciphertexts, as each EMQF encodes ciphertexts related to different secret keys. Instead, in step 2, $\mathcal{S}$ creates a threshold

ciphertext (a, b, c, d) that keeps separate terms for each of those secret keys. To decrypt, $\mathcal{P}_s$ must multiply a by sk_s, $\mathcal{P}_r$ must do so for b, and $\mathcal{S}$ for c. Next to that, the OKVS encode pairs of curve points in Montgomery form, while the operations in this protocol require them to be in twisted Edwards form. For these reasons, we present the protocol in terms of curve points rather than ElGamal ciphertexts. We note that whenever curve points are sent between the entities, they can be compressed to a single X coordinate for space efficiency.

Our OKVS-based EMQF is correct because the OKVS always returns an encryption of the identity for elements in the set. In the case where the element is not in the set, $\mu \approx 2^{-252}$ as explained in Sec. 4.6.1. This protocol, however, can be understood without consideration of the specific instantiation of the EMQF or the OKVS. In Sec. 4.7.4, we consider the aforementioned alternatives for EMQFs and compare them to our OKVS-based EMQF using PaXoS [Pin+20].

Input: EMQFs D_i relating to R_i and queries q_i for $i \in \{s, r\}$.
Output: True if $[q_s \in R_s] \wedge [q_r \in R_r]$, or false (high probability).

1. $\mathcal{S}$ computes $\hat{x}_i \,||\, \hat{y}_i \leftarrow \mathsf{decode}(D_i, q_i)$ for $i \in \{s, r\}$. It transforms them into a set of Edwards points:
$$x_s \leftarrow \mathsf{FromBytes}(\hat{x}_s)\ ,\ y_s \leftarrow \mathsf{FromBytes}(\hat{y}_s)\ ,$$
$$x_r \leftarrow \mathsf{FromBytes}(\hat{x}_r)\ ,\ y_r \leftarrow \mathsf{FromBytes}(\hat{y}_r)\ .$$
2. $\mathcal{S}$ generates $z \in_R \mathbb{Z}_q$ and computes:
$$a \leftarrow z x_s\ ,\quad b \leftarrow z x_r\ ,\quad c \leftarrow zG\ ,\quad d \leftarrow z(y_s + y_r + pk_{\mathcal{S}})\ .$$
It sends (a, b, c, d) to all entities $\mathcal{P}_i$ for $i \in \{s, r\}$.
3. $\mathcal{P}_i$ for $i \in \{s, r\}$ generates $z_i \in_R \mathbb{Z}_q$ and computes:
$$\hat{a}_i \leftarrow z_i a\ ,\quad \hat{b}_i \leftarrow z_i b\ ,\quad \hat{c}_i \leftarrow z_i c\ ,\quad \hat{d}_i \leftarrow z_i d\ .$$
They then send $(\hat{a}_i, \hat{b}_i, \hat{c}_i, \hat{d}_i)$ to $\mathcal{S}$.
4. $\mathcal{S}$ computes:
$$\alpha \leftarrow \hat{a}_s + \hat{a}_r\ ,\quad \beta \leftarrow \hat{b}_s + \hat{b}_r\ ,\quad \gamma \leftarrow \hat{c}_s + \hat{c}_r\ ,\quad \delta \leftarrow \hat{d}_s + \hat{d}_r\ .$$
It sends α to $\mathcal{P}_s$ and β to $\mathcal{P}_r$.
5. $\mathcal{P}_s$ computes $\hat{\alpha} \leftarrow sk_s \alpha$, $\mathcal{P}_r$ computes $\hat{\beta} \leftarrow sk_r \beta$, and they send $\hat{\alpha}$ and $\hat{\beta}$ to $\mathcal{S}$.
6. $\mathcal{S}$ checks if $\delta \stackrel{?}{=} \hat{\alpha} + \hat{\beta} + sk_{\mathcal{S}}\, \gamma$.

Protocol 7: Checks a record's consistency between $\mathcal{S}$ and two data-augmenting entities $\mathcal{P}_s$ and $\mathcal{P}_r$.

4.5.4 Keeping the output encrypted

In the protocol above, $\mathcal{S}$ receives the Boolean result of the computed feature, revealing whether an inconsistency was detected. While in Sec. 4.6 we provide an argument why the leaked Boolean feature is permissible with regard to the

privacy of a data-augmenting entity and the users they are serving, in this section we propose an extension of our approach to further minimize the leakage. We note, however, that the extended approach does not allow our inference step (see Sec. 4.4.3) to output a probability score between 0 and 1. Instead, with the adjusted method, our inference step outputs 0 or 1. This is an inherent limitation of the functionality.

In the extended approach, we change the inference computation from $\max(\mathcal{M}(\boldsymbol{x}), B(\boldsymbol{x}))$ to $(\mathcal{M}(\boldsymbol{x}) \geq t) \vee B(\boldsymbol{x})$ for some pre-defined threshold t. Apart from $\mathcal{M}(\boldsymbol{x}) \geq t$, the inference can be entirely computed under encryption by simply adding $\mathcal{M}(\boldsymbol{x}) \geq t$ in encrypted form to the ElGamal ciphertext in step 2 of Protocol 7.

$\mathcal{S}$ can also perform inference entirely in the encrypted domain by training a DP model on both the Boolean feature and some other features. It would do so by omitting steps 5–6 of Protocol 7 and instead performing a secure equality operation that checks if the ElGamal ciphertext encrypts the identity O, returning a ciphertext encrypting O in the positive case and G otherwise.. We present a custom secure equality protocol for this purpose in Protocol 8. Unlike typical equality protocols, it does not require full decryptions (we only check if the decryption is the identity) and no conversions to secret shares. It functions in the multi-party setting and scales linearly with the number of parties. After running this protocol, $\mathcal{S}$ can run any quantized linear model over the resulting ElGamal ciphertext (linear over the Boolean feature, the other features are plaintexts). Alternatively, $\mathcal{S}$ can collaborate with the other parties to evaluate a non-linear model by introducing more interactions. The three parties proceed to finish the protocol using steps 5–6 in Protocol 7 to decrypt the final result.

The intuitive understanding of the above protocol is that each party either adds an even or odd number of encryptions of O to the set C. So long as one party does not collude, it is not clear from the decrypted ciphertexts whether the remaining party added an even or odd number of identity encryptions. The security is determined by the number of ciphertexts k that each party adds. Let us look at a toy-sized example with two parties and $k = 5$. We disregard encryption and instead work with coins. Party $\mathcal{P}_1$ creates collection $C \leftarrow \{c\}$, containing coin c, of which we do not know if it is head or tails, which we wish to find out. Let c_0 and c_1 be coins that we know to be head H and tails T, respectively. Now, $\mathcal{P}_1$ flips k random coins that it can observe. If there is an odd number of tails, it switches c_0 and c_1, otherwise it leaves them be. Next, $\mathcal{P}_1$ adds the coins to C. At this point, C may contain $\{H, T, c, T, T, H\}$, $c_0 = T$, and $c_1 = H$. Party $\mathcal{P}_2$ performs the same process, after which we may have $C = \{T, T, H, H, T, H, T, H, c, T, H\}$, and c_0 and c_1 remain unchanged because $\mathcal{P}_2$ rolled an even number of tails. When all parties are finished, $\mathcal{P}_1$ inspects C, counting the number of tails t. It returns coin $c_{t \pmod 2}$. If $c = H$, $t = 5$ in our example, so $\mathcal{P}_1$ returns coin $c_1 = H$. If $c = T$, $t = 6$, and $\mathcal{P}_1$ returns $c_0 = T$.

We explain the correctness of this protocol in more detail in Sec. 4.6, along with our choice of k and its impact on the protocol's security. The key idea here is that if k is large enough, the set of 'coins' looks uniform, regardless of how c is distributed. Note that unlike in Protocol 7, this protocol becomes significantly slower when the number of involved entities grows. The reason is that the set of ciphertexts grows by k ciphertexts for each entity. For applications such as verifying bank

Input: Ciphertext c encrypting either the identity O or another point in $E(\mathbb{Z}_q)$.
Output: Ciphertext c' encrypting O if c encrypted O, otherwise c' encrypts G.

1. The first party $\mathcal{P}_1$ creates a set $C \leftarrow \{c\}$ and ciphertexts c_0 and c_1 encrypting O and G, respectively.
2. Each party $\mathcal{P}_i$ for $i = 1, \ldots, p$, in turn, does the following:
   - It flips k coins $r_{i,j} \in_R \{0, 1\}$.
   - It swaps and randomizes c_0 and c_1 if $r_i = \sum_{j=1}^{k} r_{i,j} = 1 \pmod 2$, setting $c_0 \leftarrow c_{r_i}$ and $c_1 \leftarrow c_{1-r_i}$.
   - For $j = 1, \ldots, k$, it appends a ciphertext encrypting O to C if $r_{i,j} = 0$. Else, a ciphertext encrypting randomness.
   - It multiplies each ciphertext in C by some random scalar from $\mathbb{Z}_q$.
   - It shuffles set C and sends the elements to party $\mathcal{P}_{i+1}$.
3. Parties $\mathcal{P}_1, \ldots, \mathcal{P}_p$ collaboratively decrypt the ciphertexts in C and count the number of non-identity elements as t.
4. The first party $\mathcal{P}_1$ outputs ciphertext $c' \leftarrow c_{t \pmod 2}$ (without decrypting it).

Protocol 8: Secure equality protocol between multiple parties $\mathcal{P}_1, \ldots, \mathcal{P}_p$ for threshold additively homomorphic encryptions of points in $E(\mathbb{Z}_q)$.

transactions this is not a problem as they only involve three entities in the equality protocol.

4.6 Privacy analysis

At the core of feature extraction is Protocol 7. We note that, in practice, this protocol can perform multiple queries in parallel. Here, we first provide a proof of correctness of this protocol and then show that it is secure in the semi-honest model. We also provide a short security argument for the secure equality operation presented in Protocol 8. We focus on our OKVS-based EMQF but the analysis works similarly for any other EMQF.

4.6.1 Proof of correctness

Claim 1. *Protocol 7 returns true when $q_s \in R_s \wedge q_r \in R_r$.*

Proof. Working backwards through the protocol:

$$\begin{aligned}
\delta &= sk_s\alpha + sk_r\beta + sk_{\mathcal{S}}\gamma \ , \\
\hat{d}_s + \hat{d}_r &= sk_s(\hat{a}_s + \hat{a}_r) + sk_r(\hat{b}_s + \hat{b}_r) + sk_{\mathcal{S}}(\hat{c}_s + \hat{c}_r) \ , \\
\cancel{(z_s + z_r)}d &= sk_s\cancel{(z_s + z_r)}a + sk_r\cancel{(z_s + z_r)}b + sk_{\mathcal{S}}\cancel{(z_s + z_r)}c \ , \\
\not{z}(y_s + y_r + pk_{\mathcal{S}}) &= sk_s\not{z}x_s + sk_r\not{z}x_r + sk_{\mathcal{S}}\not{z}G \ .
\end{aligned}$$

Since $q_s \in R_s$ and $q_r \in R_r$, then $y_s = sk_s x_s$ and $y_r = sk_r x_r$ by the functionality of an OKVS. Moreover, the setup implies $pk_{\mathcal{S}} = sk_{\mathcal{S}} G$:

$$y_s + y_r + pk_{\mathcal{S}} = sk_s x_s + sk_r x_r + sk_{\mathcal{S}} G \ , \tag{4.2}$$

$$y_s + y_r + pk_{\mathcal{S}} = y_s + y_r + pk_{\mathcal{S}} \ . \tag{4.3}$$

Claim 2. *Protocol 7 returns false with overwhelming probability when $q_s \notin R_s \vee q_r \notin R_r$.*

Proof. From (4.2) it follows that:

$$y_s + y_r + pk_{\mathcal{S}} \neq sk_s x_s + sk_r x_r + sk_{\mathcal{S}} G \ ,$$

must hold with overwhelming probability. Let us assume that $q_s \notin R_s$ (the argument follows the same when $q_r \notin R_r$). Then, $x_s \neq sk_s a$ with probability $1 - |E(\mathbb{Z}_q)|^{-1}$, where $|E(\mathbb{Z}_q)| > 2^{252}$. As a result, (4.3) only holds with negligible probability. □

4.6.2 Proof of privacy

In Protocol 6 the banks only encode an OKVS and generate an ElGamal keypair. The security of this keypair, which $\mathcal{S}$ also generates, is implied by the decisional Diffie-Hellman assumption (or more precisely, by the discrete log problem). The security of the OKVS is defined by its indistinguishability from randomness. We achieve this by encoding curve points as strings that are indistinguishable from random bytes, as proposed in Algs. 4 and 5.

Given that Protocol 7 is correct (see above) and the ideal functionality is deterministic, what remains is to show that the protocol privately computes the ideal functionality in the semi-honest model [Lin17]. We do so by showing that there exists a simulator that given the input and output can replicate the view of a party without having access to the data of other parties. To be precise, the family of simulated views is computationally indistinguishable from those of actual protocol executions. Our protocol relies on the Diffie-Hellman assumption:

Lemma 14. *The decisional Diffie-Hellman assumption implies that, for a generator point G, unknown scalars $a, b \in_R \mathbb{Z}_q$, and random point $C \in_R E(\mathbb{Z}_q)$: $(a\,G, b\,G, ab\,G) \overset{c}{\equiv} (a\,G, b\,G, C)$.*

We provide two privacy proofs: one that proves $\mathcal{S}$'s view $\mathsf{view}_{\mathcal{S}}$ to be simulatable and one for a $\mathcal{P}_i$'s view $\mathsf{view}_{\mathcal{P}}$. We keep these proofs short. We direct the reader to the work by Vos et al. [VCE22] for a more detailed proof of a comparable protocol. To simplify notation, we do not explicitly pass the source of randomness as an input to the simulator. For the purpose of our arguments, we consider the OKVSs D_i for $i \in Q$ to be public. Given that their contents are statistically indistinguishable from randomness, this only leaks their size. This first proof shows that the protocol remains private when $\mathcal{S}$ is corrupted.

Claim 3. *There exists a simulator $\mathsf{Sim}_{\mathcal{S}}$ for PNS in Protocol 7, s.t.:*

$$\{\mathsf{Sim}_{\mathcal{S}}(1^\lambda, q_s, q_r, o)\}_{q_s \in Q,\, q_r \in Q,\, o \in \{0,1\}} \overset{c}{\equiv}$$
$$\{\mathsf{view}_{\mathcal{S}}(q_s, q_r, \lambda)\}_{q_s \in Q,\, q_r \in Q,\, o \in \{0,1\}} \ ,$$

for security parameter $\lambda = 128$, queries q_s and q_r from query space Q, and output o.

Proof. Function $\text{view}_{\mathcal{S}}$ returns inputs q_s and q_r, output o, and all incoming messages . Simulator $\text{Sim}_{\mathcal{S}}$ generates an indistinguishable view by outputting the inputs and output, and randomly sampling messages $\hat{a}_s, \hat{b}_s, \hat{c}_s, \hat{d}_s, \hat{a}_r, \hat{b}_r, \hat{c}_r, \hat{d}_r, \hat{\alpha}, \hat{\beta} \in_R E(\mathbb{Z}_q)$. These messages are indistinguishable from those received in actual executions:

- In step 3, $\hat{a}_i = z_i a = z_i p_a G$ for some p_a unknown to $\mathcal{S}$. Given Lemma 14, $\hat{a}_i$ is computationally indistinguishable from randomness, even when given $a = p_a G$ and $z_i G$ (the latter is not actually given). The same argument applies to $\hat{b}_i$, $\hat{c}_i$, and $\hat{d}_i$.
- In step 5, $\mathcal{S}$ receives $\hat{\alpha} = sk_s \alpha = sk_s p_\alpha G$ and $\hat{\beta} = sk_r \beta = sk_r p_\beta G$. Given Lemma 14, α is computationally indistinguishable from randomness, even when given $pk_r = sk_r G$ and $\beta = p_\beta G$. The same argument applies to $\hat{\beta}$. □

Next, we prove that the protocol remains private when a data-augmenting entity is corrupted.

Claim 4. *There exists a simulator* $\text{Sim}_{\mathcal{P}}$ *for* $\mathcal{P}_s$ *in Protocol 7, s.t.:*

$$\{\text{Sim}_{\mathcal{P}}(1^\lambda)\} \stackrel{c}{\equiv} \{\text{view}_{\mathcal{P}_i}(\lambda)\} ,$$

for security parameter $\lambda = 128$ *(the data-augmenting entities do not output anything).*

Proof. Function $\text{view}_{\mathcal{P}_i}$ returns all incoming messages of bank $\mathcal{P}_i$. Simulator $\text{Sim}_{\mathcal{P}}$ generates an indistinguishable view by randomly sampling messages $a, b, c, d, \alpha \in_R E(\mathbb{Z}_q)$. These messages are indistinguishable from those received in actual executions:

- In step 2, $\mathcal{P}_i$ receives $a = zx_s = zp_a G$, $b = zx_r = zp_b G$, $c = zG$, and $d = z(y_s + y_r + pk_{\mathcal{S}}) = zp_d G$. Given Lemma 14, a is computationally indistinguishable from randomness, even when given $c = zG$ and $p_a G$ (which may be guessed by $\mathcal{P}_i$). The same argument applies to b and d. Since z is random, $c = zG$ is statistically indistinguishable from randomness.
- In step 4, $\mathcal{P}_i$ receives $\alpha = \hat{a}_s + \hat{a}_r$, which is indistinguishable from randomness since $\hat{a}_r$ is unknown to $\mathcal{P}_i$ given that the queried banks are not colluding. □

We note that one might also give a proof that proves that the protocol remains private when two of the three parties collude. This would require a more sophisticated simulator, which looks similar to that in the work by Vos et al. [VCE22].

4.6.3 Security of the equality protocol

Finally, we prove the security of our secure equality protocol 8.

Claim 5. *Protocol 8 correctly and privately computes an equality.*

Proof. Verifying correctness of the secure equality protocol (Protocol 8) comes down to verifying its behavior depending on whether a party's coin tosses come out to an even or odd number of 1s. We study the case where there is one party, but the argument extends trivially to multiple parties.

- If $r_1 = 0$, then c_0 encrypts O and c_1 encrypts G. If c encrypts O, then $c' = c_0$. Otherwise, $c' = c_1$. Both are correct.

- If party $\mathcal{P}_i$ has $r_i = 1$, then c_0 encrypts G and c_1 encrypts O. If c encrypts O, then $c' = c_1$. Otherwise, $c' = c_0$. Both are correct.

Next, we analyze the security of Protocol 8. We do not consider the security of the ElGamal scheme, which we discussed previously. In the protocol, each party randomizes and shuffles the set of ciphertexts C. As a result, the only meaningful information that is revealed when the set is decrypted is the number of identity points. We refer to the number of non-identity points as t.

If a party would only perform random coin flips, the number of non-identity points t is given by $P(\boldsymbol{t} = t) = \binom{k}{t}0.5^k$. However, since we are inserting ciphertext c, this changes to $P(\boldsymbol{t} = t) = P(c = O)\binom{k-1}{t}0.5^{k-1} + P(c \neq O)\binom{k-1}{t-1}0.5^{k-1}$, where we use $c \neq O$ to denote that c does not encrypt O. Using this, we derive the posterior probability that $c \neq O$ given the number of points t.

$$
\begin{aligned}
P(c \neq O | \boldsymbol{t} = t) &= \frac{P(\boldsymbol{t} = t | c \neq O) P(c \neq O)}{P(\boldsymbol{t} = t)}, \\
&= \frac{\binom{k-1}{t-1}\cancel{0.5^{k-1}} P(c \neq O)}{(1 - P(c \neq O))\binom{k-1}{t}\cancel{0.5^{k-1}} + P(c \neq O)\binom{k-1}{t-1}\cancel{0.5^{k-1}}}, \\
&= \frac{\binom{k-1}{t-1} P(c \neq O)}{(1 - P(c \neq O))\binom{k-1}{t} + P(c \neq O)\binom{k-1}{t-1}}.
\end{aligned}
$$

The strongest attack is to guess $c = O$ when $P(c \neq O | \boldsymbol{t} = t) < \frac{1}{2}$ and $c \neq O$ otherwise. The expected guessing chance is then:

$$
\sum_{t=0}^{k} P(\boldsymbol{t} = t) \underbrace{\left(\frac{1}{2} + \left|\frac{1}{2} - P(c \neq O | t = t)\right|\right)}_{\text{Guess based on the posterior}}. \tag{4.4}
$$

An adversary who does not have access to the protocol's result can only guess using its knowledge about the prior probability of c, so it will succeed with probability $\frac{1}{2} + \left|\frac{1}{2} - P(c = O)\right|$. Using (4.4), we formulate the advantage of an adversary using our Protocol 8, and restrict it to 2^{-40}, which we deem negligible:

$$
\left(\sum_{t=0}^{k} P(\boldsymbol{t} = t)\left(\frac{1}{2} + \left|\frac{1}{2} - P(x = 1 | t = t)\right|\right)\right) - \left(\frac{1}{2} + \left|\frac{1}{2} - P(c \neq O)\right|\right) \leq 2^{-40}. \tag{4.5}
$$

The choice of k then depends on the probability of anomalies occurring. Let us consider $P(c \neq O) \leq 0.05$ as an example. Then, the first k for which (4.5) holds is $k = 44$. □

4.7 Performance analysis

We empirically evaluated the utility and scalability of our solution for the detection of anomalies among millions of financial transactions in a federated setup with

a payment network system (the centralized entity) and partner banks (the data-augmenting clients). To this end, we implemented our solution in *Flower*, a well-known framework for federated learning [Beu+20]. *Flower* assumes a star topology in which all data holders, including our centralized entity $\mathcal{S}$ and the data-augmenting clients, are connected to an aggregator that can only perform simple tasks like averaging and message passing. Since it does not support client peer-to-peer communication, we route all communication between $\mathcal{S}$ and the data-augmenting clients through the aggregator. We note that this is an implementation detail rather than a requirement originating from our protocol. We implemented Protocol 7 in Rust (with a Python wrapper).[2]

4.7.1 Experimental setup

Data We use synthetic data provided by SWIFT to participants in the 2023 PETs prize challenge.[3] The data consists of two parts:

1. Transaction data held by $\mathcal{S}$ (payment network system): a dataset of financial transactions that are labeled as anomalous (positive) or not. This dataset is split into a train dataset with 2,990,349 negative and 3,521 positive instances, and a test dataset with 1,002,395 negative and 1,279 positive instances. Each transaction known to $\mathcal{S}$ has, apart from other information, details of the ordering and beneficiary accounts, and the financial institutions involved in making the transaction. These financial institutions are the centralized entity's partner banks which hold information about the ordering and beneficiary accounts.
2. Bank account data held by the data-augmenting clients $\mathcal{P}_1, \ldots, \mathcal{P}_n$: information about bank accounts. To test scalability, we distribute this data among a varying number of clients.

Features for model training by $\mathcal{S}$ We train model $\mathcal{M}$ (see Sec. 4.4.2) on the following features that are held by $\mathcal{S}$:

- Unexpected currency. Most transactions in the train data have the same InstructedCurrency and SettlementCurrency. The few transactions that involve two different currencies are all anomalous. We include `SameCurrency` as a feature.
- Unusual timestamps. We found the Timestamp and the SettlementDate to be strong indicators of anomalous transactions in the training data. We encode this as a feature `InterimTime` which is the Timestamp subtracted from the SettlementDate.

Consistency checks for Boolean feature extraction by $\mathcal{S}$ and the banks We assume that $\mathcal{S}$ has a list of unique IDs pertaining to $\mathcal{P}_1, \ldots, \mathcal{P}_n$. In our implementation, $\mathcal{S}$ receives this information from the clients in a setup phase. For a transaction x involving a valid sending bank $\mathcal{P}_s$ and receiving bank $\mathcal{P}_r$ (i.e., $\mathcal{S}$ holds their IDs), $\mathcal{S}$ and the banks engage in a cryptographic protocol to compute a Boolean feature $B(x)$ derived from:

[2] The code can be found at DOI `10.4121/4e1739c5-f743-47cc-aa01-df52481e3fb3` or on GitHub.
[3] `https://www.drivendata.org/competitions/group/nist-federated-learning/`

- Information from sending bank $\mathcal{P}_s$ indicating...
  - whether the account ID of the ordering entity as listed in the transaction x is a valid ID known to the ordering bank.
  - whether the name of the ordering entity as listed in the transaction x is the same as the name known to the ordering bank.
  - whether the street address of the ordering entity as listed in x is the same as the street address known to the ordering bank.
  - whether the country/city/zip of the ordering entity as listed in x is the same as that known to the ordering bank.
  - whether the ordering bank has flagged the ordering entity's account for any reason (e.g. account closed, account frozen,...).
- Information from receiving bank $\mathcal{P}_r$ indicating the same as above, but for the account of the beneficiary entity.

The Boolean feature $B(x)$ is 0 if the account information appears correct and 1 if there is any indication of inconsistency or unusual account information in either the sender or the receiver account.

During the setup phase, each data-augmenting client sets up a single OKVS and key pair using Protocol 6. This protocol generalizes to any type of data as its only requirement is that the data is hashable. In our implementation, we let bank $\mathcal{P}_i$ encode R_i in the OKVS, which contains the `["Account", "Name", "Street", "CountryCityZip"]` columns. The bank omits any flagged entries from R_i (that is, where `flag != "0"`). $\mathcal{S}$ performs a different setup, only generating a single key pair.

During inference, $\mathcal{S}$ must check whether the ordering bank's dataset contains a row for the ordering user, and whether the beneficiary bank contains a row for the beneficiary. It does so using one invocation of Protocol 7. The selected fields are `["OrderingAccount", "OrderingName", "OrderingStreet", "OrderingCountryCityZip"]` and `["BeneficiaryAccount", "BeneficiaryName", "BeneficiaryStreet", "BeneficiaryCountryCityZip"]`.

4.7.2 Utility-privacy tradeoffs

The utility results in Tab. 4.2 are obtained by fitting models on the train dataset and evaluating them on the test dataset in terms of AUPRC (area under the precision-recall curve). We compare two kinds of models: Random Forest (RF) and Logistic Regression (LR). The models in Tab. 4.2 are trained on the feature set consisting of `SameCurrency` and `InterimTime`. However, the LR uses a discretized version of the `InterimTime` feature, as we explain in more detail below. In Tab. 4.2, we provide predictions made by all the models themselves, as well as when they are augmented with the Boolean feature representing the consistency check with the data-augmenting clients, denoted <model>$_{okvs}$ (Sec. 4.4.3).

The models are trained with algorithms that provide DP guarantees, under varying privacy budgets ϵ, corresponding to the different columns in Tab. 4.2. For comparison, in the last column we include results for models trained with an infinite privacy budget, i.e. with no DP guarantee at all. These models are trained with sklearn [Ped+11], using default values for the hyperparameters, with the exception of the use of 20 trees and max_depth = 10 for RF.

Table 4.2: Utility-privacy tradeoff of models augmented with the consistency feature extracted jointly by $\mathcal{S}$ and the banks, averaged over 5 runs. Higher ϵ implies less privacy. The AUPRC results are independent of the number of clients. Anomaly detection consistently improves the AUPRC, even when the base model was already performing well.

Model	$\epsilon = 0.5$	$\epsilon = 1.0$	$\epsilon = 5.0$	$\epsilon = \infty$
RF	0.648	0.645	0.669	0.963
RF_{EMQF}	0.716	0.727	0.743	0.979
LR	0.869	0.908	0.915	0.943
LR_{EMQF}	0.892	0.925	0.935	0.964

with DP ($\epsilon < \infty$):
– RF-DP: Fletcher et al. [FI17]
– LR-DP: DP-SGD [Aba+16]
without DP ($\epsilon = \infty$):
– RF, LR: sklearn [Ped+11]

For ease of reference, in the text below we denote the models that we trained with a DP algorithm by appending "-DP" to indicate that they are differentially private. To train the RF-DP models in Tab. 4.2 we used the diffprivlib library [Hol+19], while for LR-DP we used the implementation of DP-SGD in TensorFlow Privacy [Mar+21]. Similar to the non-private setting ("$\epsilon = \infty$") from the last column in Tab. 4.2, the RF-DP models are trained with 20 trees and max_depth = 10. The way in which trees are constructed in this RF-DP approach [FI17] is quite different from the standard RF algorithm in sklearn that we used in the non-private approach. While in the standard RF algorithm each node in each tree is selected by evaluating it against the data, in the RF-DP approach, intermediate nodes and threshold values for these nodes are generated at random, to limit the number of queries needed against the data and stretch the privacy budget further. While in the non-private setting we obtained our best results with RF, this was no longer the case with RF-DP because the `InterimTime` feature only really pays off for well chosen thresholds. As mentioned earlier, the RF algorithm in the non-private setting was able to find and pick up those thresholds, while the RF-DP approach with all its random guessing of thresholds was not. As a result, in the federated setting, the LR-DP (on DP-SGD [Aba+16]) approach took over in terms of better utility, and, as we observed, was most stable across different runs.

DP discretization of `InterimTime` We observed the `InterimTime` to be crucial for identifying anomalous transactions. Based on our observations in the non-private setting (the last column in Tab. 4.2), non-DP RF yields high AUPRC because the underlying decision tree learning algorithm has a built-in technique to find good thresholds for dynamic discretization of the `InterimTime` feature during tree construction. The DP training algorithms cannot detect such thresholds with the same ease. To mitigate this, we statically discretize the `InterimTime` feature into bins. We replace the `InterimTime` feature in each transaction with its corresponding bin number and one-hot-encode the bin numbers in the training set. In Appendix 4.A, we explain how to achieve this while satisfying differential privacy.

Utility of consistency checks As demonstrated in Tab. 4.2, there is a clear boost in accuracy when augmenting the centralized entity's model with the EMQF-based feature extraction protocol. The lower the accuracy of the model trained by $\mathcal{S}$,

Table 4.3: Efficiency and scalability results on development data. The efficiency of the clients stays consistent as the partitions change, while $\mathcal{S}$'s efficiency increases. The client communication cost grows superlinearly as communication depends on the number of client *pairs* in our experiments.

	Time			Memory		Communication	
	Total	$\mathcal{S}$	$\mathcal{P}_i$	$\mathcal{S}$	$\mathcal{P}_i$	$\mathcal{S}$	$\mathcal{P}_i$
$\mathcal{S}$ + 2 clients	1596s	1198s	228s	3.50GB	1.95GB	1052B	1584B
$\mathcal{S}$ + 4 clients	1581s	1173s	234s	3.92GB	2.01GB	1200B	3168B
$\mathcal{S}$ + 9 clients	2701s	2215s	243s	4.36GB	1.85GB	2236B	7128B

the greater the benefit of the data-augmenting feature extraction. Predicting anomalous instances solely from the feature extraction protocol (i.e. without training on the centralized entity's data) yielded an AUPRC of .294.

4.7.3 Efficiency and scalability

We performed efficiency and scalability experiments on a desktop Intel i7 6700k at 4.2GHz, 64GB memory, and GTX1080 GPU. The results in Tab. 4.3 are based on training a model, and running the consistency checks using three different client partitioning scenarios. In these scenarios, the average number of accounts per client is 561,935; 280,968; and 124,874; respectively. The runtimes, memory usages, and communication costs for $\mathcal{S}$ and clients change with the partition. Included in these metrics is the federated set-up, done in a privacy-preserving manner with the protocols from Sec. 4.4.3.

It can be seen in Tab. 4.3 that as the number of clients increases, the efficiency of the $\mathcal{S}$ decreases. This is expected since $\mathcal{S}$'s computation cost is largely dependent on the number of sender-receiver client pairs, which grows superlinearly with the amount of clients. However, the results also show that the total client runtime and memory resources remain fairly constant, and are mostly a function of the amount of client data rather than number of partitions. This can be explained by the fact that each clients' computational load is proportional to its data size. The total clients' communication cost on the other hand scales superlinearly with the amount of clients, since in our experiments, like in $\mathcal{S}$, this computation depends more on the amount of sender-receiver pairs.

Computational cost of the setup We evaluated the run time of the setup by measuring how long it takes for a bank to create an OKVS. Since many parts of the PaXoS [Pin+20] OKVS algorithm can be parallelized, we ran this experiment on 8 threads. We present the results in Fig. 4.2. The run time scales linearly with the bank's dataset size. We note that while it takes approximately 90 seconds to generate an OKVS for 250,000 rows, this operation has to be computed only once (or whenever the dataset needs to change).

Cost of consistency checks We also evaluated the run time of performing 128 consistency checks, split by the entities that take part in the protocol. We present the results in Fig. 4.3, where solid bars represent the measured computation

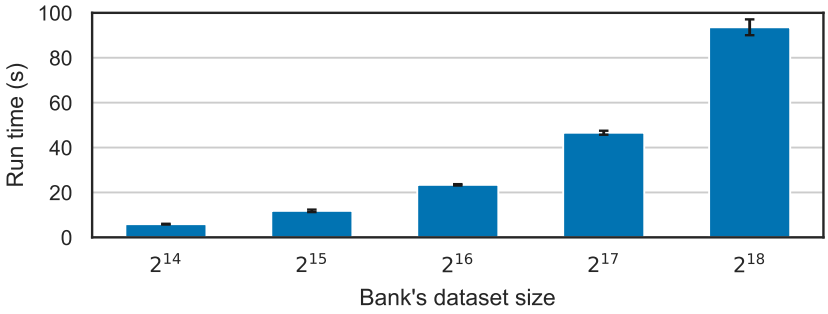


Figure 4.2: Time required for a bank to generate the OKVS using 8 threads on an M1 processor. Averaged over 10 runs, the error bars indicate the standard deviation. The run time scales linearly with the size of the bank's dataset.

time averaged over 10 runs, and transparent bars represent the time that a party would spend waiting to receive messages. This figure assumes that the latency is 100ms and we do not consider throughput constraints, given that all entities only exchange a small amount of compressed curve points which are made up of 32 bytes each. The run time for the centralized entity here scales linearly, while the data-augmenting entities perform a constant amount of work. Even at more than 250,000 rows, the centralized entity only spends 80 ms per query.

4.7.4 Comparison with other EMQFs

In Sec. 4.5.2 we discussed how there are multiple potential instantiations for a primitive that only returns encryptions of the identity when queried on elements in the set that it represents. Such a primitive allows moving a large fraction of computation and communication to a one-time (or each time that the set is updated) setup. We now compare the PaXoS OKVS with two alternatives that are used in MPSI protocols. Specifically, we compare PaXoS with the encrypted polynomials in the protocol by Hazay et al. [HV17] and the encrypted Bloom filters in the protocol by Vos et al. [VCE22] (Chapter 3). For all schemes, we use the same elliptic curve-based ElGamal ciphertexts. We measure the average time it takes to encode a set, to decode (query on an element), and the size of the OKVS or set representation. We present the results in Table 4.4 for a moderately-sized set of $k = 2^{14} = 16,384$ elements over 10 runs.

Notice how encrypted polynomials are the most compact but take the longest to encode. In fact, their size is optimal because the representation is exactly as large as the number of elements it contains multiplied by the size of a single ciphertext. Encrypted Bloom filters are significantly larger, even for large false positive probabilities μ that are shown to lead to security problems in Chapter 2. For the other schemes, the probability of a query returning a false positive is negligible. Finally, the PaXoS OKVS is efficient to encode and differs only a factor of 2.4 with the encrypted polynomial in terms of its size. This factor can be further reduced by increasing the number of hash functions [Gar+21]. One can also

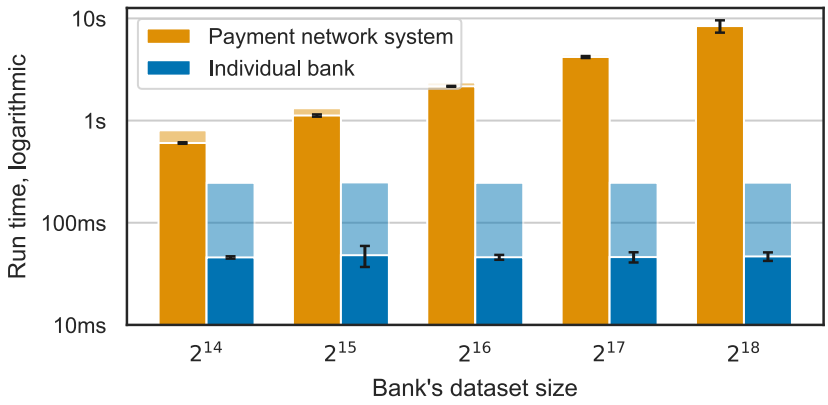


Figure 4.3: Time required for each party to complete 128 consistency checks on an M1 processor with one thread. Averaged over 10 runs, the error bars indicate the standard deviation. The run time is constant for the data-augmenting entities and scales linearly for the centralized entity. The transparent bars indicate time spent waiting for messages to arrive when the latency is 100ms, ignoring throughput constraints.

consider other OKVSs to instantiate our EMQF, e.g. trading off size and the time that it takes to encode or decode. Van Baarsen & Lu [vBP24] provide another way of encoding curve points in an OKVS.

Table 4.4: Comparing EMQFs with $k = 2^{14}$ over 10 runs.

EMQF	**Encode**	**Decode**	**Size**
Encrypted polynomial	22.2s	1.03s	1 MB
Encrypted BF, $\mu = 0.1\%$	7.79s	0.02ms	18.8 MB
Encrypted BF, $\mu = 1\%$	5.19s	0.01ms	12.6 MB
PaXoS with ciphertexts	2.25s	0.03ms	2.4 MB

4.7.5 Comparison with MPSI protocols

Standard MPSI protocols are not designed to be more efficient when some sets are fixed. We show that our work outperforms even fast MPSI protocols like the one by Kolesnikov et al. [Kol+17] by orders of magnitude. We use the popsicle library by Galois Inc. [Gal19], which implements this protocol using the same elliptic curve library as in our implementation, assigning 1 thread per party. We report total run time, including setup time, and the bytes sent by $\mathcal{P}_i$ and $\mathcal{S}$ after setup. The implementation requires $\mathcal{S}$'s set to have the same size as the others' (same in the original), so the run time does not change whether the server queries $p = 1$ or $p = k$ elements in parallel. We report the cost per sequential query on an M1 processor.

Note that in [Kol+17], run time would be twice as low since each party has 2 threads. Their actual numbers are significantly lower, but even using their Table 3,

Table 4.5: Efficiency when performing p parallel queries

	$k = 2^{16}$			$k = 2^{18}$		
	Time	$\mathcal{P}_i$ sent	$\mathcal{S}$ sent	Time	$\mathcal{P}_i$ sent	$\mathcal{S}$ sent
Ours, $p = 1$	2.1 s	160 B	320 B	8.3 s	160 B	320 B
Ours, $p = 128$	2.7 s	20 kB	40 kB	8.9 s	20 kB	40 kB
Ours, $p = 512$	4.7 s	80 kB	160 kB	11.0 s	80 kB	160 kB
Kolesnikov et al.	79.6 s	956 MB	665 MB	327.2 s	3.84 GB	2.67 GB

the communication cost is 3 orders of magnitude higher than ours, while the run times are similar. Also, their protocol requires a full-mesh topology, incurring additional delays when communication is routed through a central entity.

4.8 Limitations

Protocol 7 discloses a single bit of private data from the data-augmenting entities to the central entity. This bit signals inconsistencies between data held by $\mathcal{S}$ and the data from data-augmenting entities $\mathcal{P}_i$. The implications of this single bit of information can be substantial depending on the nature of the data set held by $\mathcal{S}$. A zero bit immediately implies that the data entry held by $\mathcal{S}$ aligns with the data held by $\mathcal{P}_i$. In the case the leaked bit is one, there is also the potential for information leakage. within the specific context of the PETS prize, there exist multiple transactions with identical names (or addresses or banks). $\mathcal{S}$ can utilize the private set membership protocol results for these transactions with overlapping data to discern inconsistencies in these specific fields and recover them. For example, if two queries are made with the same name but the result of the query is different for both, then $\mathcal{S}$ will learn that the other fields are causing the discrepancy. Whether this release of information is acceptable depends on the particular application of our proposed protocols and needs a case-by-case analysis. The leakage of this information needs to be weighed against the societal cost of not detecting anomalous data.

The scenario described above is not unique to our proposed solution. It can occur in any situation where $\mathcal{S}$ trains a model with features sourced from $\mathcal{P}_i$ and where anomalies strongly correlate with data discrepancies between $\mathcal{S}$ and $\mathcal{P}_i$. In that case, learning the classification outcome of a transaction (anomalous or not) already provides significant insight regarding any potential data mismatch, exactly as in the case of our disclosed bit. For this reason, we loosely define the following requirements to decide whether our protocol can be applied in a given use case:

- There is a centralized entity with a global view of the system, but whose view can be enriched by incorporating data from data-augmenting entities.
- Data-augmenting entities only update their data at a low frequency, ensuring that OKVSs can be reused.
- All involved parties have an incentive to act semi-honestly. E.g., through legal obligations or financial incentives.

- The output of a private membership query may be revealed to the centralized entity. Or, when using Protocol 8, the output of the computation that follows it.

The release of information described above can be prevented by having the banks locally randomize their information and obtaining local differential privacy guarantees at the cost of reducing the utility of the final model. In the case the same query is repeated multiple times by $\mathcal{S}$, the privacy budget needs to be adjusted accordingly by using differential privacy's sequential composition property. This approach will typically not work, however, given that anomalies only make up a small proportion of the entire set of data. The reason is that the randomization will make the signal very noisy, significantly increasing the number of false positives.

Finally, we want to briefly state that while our protocol solves this specific privacy aspect, in practice one must take into account the wider context in which the protocol is deployed. One should be cautious about applying automated anomaly detection in general, as there are issues beside potential privacy violations that may negatively impact users. Nevertheless, our protocol fills an important gap in situations where anomaly detection is applied in federations.

4.9 Conclusion

Motivated by the PETS prize challenge, we propose an efficient solution for federated anomaly detection. Unlike traditional federated learning scenarios, our solution works for a case where the data is horizontally and vertically partitioned. Moreover, our solution is based on an efficient private feature extraction protocol - where features used in the training of a machine learning model are computed based on information distributed across different parties. Our proposed framework has applications beyond the specific scenario presented in the PETS competition. It proves valuable in any situation where inconsistencies across distributed data sets serve as important information for anomaly detection and does so while preserving privacy.

Despite the extensive literature on privacy-preserving machine learning focusing on protocols for private training of machine learning models over distributed datasets, our approach addresses a commonly neglected issue: privacy-preserving feature extraction. We privately compute features calculated over the distributed data set and subsequently use these features for training ML models. To accomplish this, we introduce an innovative private set membership protocol, combining the efficiency of oblivious key-value stores with inputs encrypted using elliptic curve-based ElGamal. By combining these two building blocks, the entities can perform membership queries with low computational overhead and bandwidth costs, while the only communication happens between the centralized entity and the involved data-augmenting entities.

References

[Aba+16] Martín Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 308–318. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145/2976749.2978318.

[Aba+24] Aydin Abadi et al. "Starlit: Privacy-Preserving Federated Learning to Enhance Financial Fraud Detection". In: *IACR Cryptol. ePrint Arch.* (2024), p. 90. URL: https://eprint.iacr.org/2024/090.

[Ada+22] Samuel Adams et al. "Privacy-preserving training of tree ensembles over continuous data". In: *Proceedings on Privacy Enhancing Technologies* 2 (2022), pp. 205–226.

[Aga+19] Anisha Agarwal et al. "Protecting Privacy of Users in Brain-Computer Interface Applications". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 27.8 (2019), pp. 1546–1555.

[Ash+17] Gilad Asharov et al. "More efficient oblivious transfer extensions". In: *Journal of Cryptology* 30.3 (2017), pp. 805–858.

[Asi+23] Hafiz Asif et al. *Anomaly Detection via Privacy-Enhanced Two-Step Federated Learning*. Tech. rep. Rutgers University, Apr. 2023. URL: https://rutgers.app.box.com/s/q84zjo3edv5d1e1eu67ypihiw8cb2djq.

[Bay+22] Aslí Bay et al. "Practical Multi-Party Private Set Intersection Protocols". In: *IEEE Trans. Inf. Forensics Secur.* 17 (2022), pp. 1–15. DOI: 10.1109/TIFS.2021.3118879. URL: https://doi.org/10.1109/TIFS.2021.3118879.

[Ber+13] Daniel J. Bernstein et al. "Elligator: elliptic-curve points indistinguishable from uniform random strings". In: *2013 ACM SIGSAC Conference on Computer and Communications Security*. 2013, pp. 967–980. DOI: 10.1145/2508859.2516734. URL: https://doi.org/10.1145/2508859.2516734.

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*. Ed. by Moti Yung et al. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 207–228. DOI: 10.1007/11745853_14. URL: https://doi.org/10.1007/11745853%5C_14.

[Beu+20] Daniel J. Beutel et al. "Flower: A Friendly Federated Learning Research Framework". In: *CoRR* abs/2007.14390 (2020). arXiv: 2007.14390. URL: https://arxiv.org/abs/2007.14390.

[Boe+21] Franziska Boenisch et al. "When the curious abandon honesty: Federated learning is not private". In: *arXiv preprint arXiv:2112.02918* (2021).

[BP20] David Byrd and Antigoni Polychroniadou. "Differentially private secure multi-party computation for federated learning in financial applications". In: *ICAIF '20: The First ACM International Conference on AI in Finance, New York, NY, USA, October 15-16, 2020*. Ed. by Tucker Balch. ACM, 2020, 16:1–16:9. DOI: 10.1145/3383455.3422562. URL: https://doi.org/10.1145/3383455.3422562.

[Car+19] Nicholas Carlini et al. "The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks". In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Ed. by Nadia Heninger and Patrick Traynor. USENIX Association, 2019, pp. 267–284. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/carlini.

[CDN15] Ronald Cramer, Ivan Damgard, and Jesper Nielsen. *Secure Multiparty Computation and Secret Sharing*. New York: Cambridge University Press Print, 2015.

[CGM21] Eduardo Chielle, Homer Gamil, and Michail Maniatakos. "Real-time Private Membership Test using Homomorphic Encryption". In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 1282–1287. DOI: 10.23919/DATE51398.2021.9473968. URL: https://doi.org/10.23919/DATE51398.2021.9473968.

[Cha+17] Melissa Chase et al. "Private collaborative neural network learning". In: *Cryptology ePrint Archive* (2017).

[Cha+21] Nishanth Chandran et al. "Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim et al. ACM, 2021, pp. 1182–1204. DOI: 10.1145/3460120.3484591. URL: https://doi.org/10.1145/3460120.3484591.

[CM08] Kamalika Chaudhuri and Claire Monteleoni. "Privacy-preserving logistic regression". In: *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*. Ed. by Daphne Koller et al. Curran Associates, Inc., 2008, pp. 289–296. URL: https://proceedings.neurips.cc/paper/2008/hash/8065d07da4a77621450aa84fee5656d9-Abstract.html.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. "Differentially Private Empirical Risk Minimization". In: *J. Mach. Learn. Res.* 12 (2011), pp. 1069–1109. DOI: 10.5555/1953048.2021036. URL: https://dl.acm.org/doi/10.5555/1953048.2021036.

[Col21] Yann Collet. *xxHash*. 2021. URL: https://cyan4973.github.io/xxHash/.

[De +21] Martine De Cock et al. "High performance logistic regression for privacy-preserving genome analysis". In: *BMC Medical Genomics* 14(23) (2021).

[DR+14] Cynthia Dwork, Aaron Roth, et al. “The algorithmic foundations of differential privacy”. In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014), pp. 211–407.

[Elk+22] Ahmed Roushdy Elkordy et al. “How Much Privacy Does Federated Learning with Secure Aggregation Guarantee?” In: *arXiv preprint arXiv:2208.02304* (2022).

[FI17] Sam Fletcher and Md Zahidul Islam. “Differentially Private Random Decision Forests using Smooth Sensitivity”. In: *Expert Systems with Applications* 78 (2017), pp. 16–31.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. “Model inversion attacks that exploit confidence information and basic countermeasures”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 1322–1333.

[Gal19] Galois, Inc. *swanky: A suite of rust libraries for secure computation*. https://github.com/GaloisInc/swanky. 2019.

[Gar+21] Gayathri Garimella et al. “Oblivious Key-Value Stores and Amplification for Private Set Intersection”. In: *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*. Ed. by Tal Malkin and Chris Peikert. Vol. 12826. Lecture Notes in Computer Science. Springer, 2021, pp. 395–425. DOI: 10.1007/978-3-030-84245-1_14. URL: https://doi.org/10.1007/978-3-030-84245-1%5C_14.

[Gar+23] Sanjam Garg et al. “Credibility in Private Set Membership”. In: *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part II*. Ed. by Alexandra Boldyreva and Vladimir Kolesnikov. Vol. 13941. Lecture Notes in Computer Science. Springer, 2023, pp. 159–189. DOI: 10.1007/978-3-031-31371-4_6. URL: https://doi.org/10.1007/978-3-031-31371-4%5C_6.

[GHL22] S. Dov Gordon, Carmit Hazay, and Phi Hung Le. “Fully Secure PSI via MPC-in-the-Head”. In: *Proc. Priv. Enhancing Technol.* 2022.3 (2022), pp. 291–313. DOI: 10.56553/popets-2022-0073. URL: https://doi.org/10.56553/popets-2022-0073.

[GLX21] Xiaolan Gu, Ming Li, and Li Xiong. “PRECAD: Privacy-Preserving and Robust Federated Learning via Crypto-Aided Differential Privacy”. In: *arXiv preprint arXiv:2110.11578* (2021).

[GN19] Satrajit Ghosh and Tobias Nilges. “An Algebraic Approach to Maliciously Secure Private Set Intersection”. In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Springer, 2019, pp. 154–185. DOI: 10.1007/978-3-030-17659-4_6. URL: https://doi.org/10.1007/978-3-030-17659-4%5C_6.

[Guo+20] Chuan Guo et al. “Secure multiparty computations in floating-point arithmetic”. In: *arXiv:2001.03192* (2020).

[Hol+19] Naoise Holohan et al. “Diffprivlib: the IBM differential privacy library”. In: *ArXiv e-prints* 1907.02444 [cs.CR] (July 2019).

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. “Scalable Multi-party Private Set-Intersection”. In: *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*. Ed. by Serge Fehr. Vol. 10174. Lecture Notes in Computer Science. Springer, 2017, pp. 175–203. DOI: 10.1007/978-3-662-54365-8_8. URL: https://doi.org/10.1007/978-3-662-54365-8%5C_8.

[IOP18] Roi Inbar, Eran Omri, and Benny Pinkas. “Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters”. In: *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*. Ed. by Dario Catalano and Roberto De Prisco. Vol. 11035. Lecture Notes in Computer Science. Springer, 2018, pp. 235–252. DOI: 10.1007/978-3-319-98113-0_13. URL: https://doi.org/10.1007/978-3-319-98113-0%5C_13.

[Jay+18] Bargav Jayaraman et al. “Distributed learning without distress: privacy-preserving empirical risk minimization”. In: *Advances in Neural Information Processing Systems 31*. 2018, pp. 6346–6357.

[Kad+23] Swanand Ravindra Kadhe et al. “Privacy-Preserving Federated Learning over Vertically and Horizontally Partitioned Data for Financial Anomaly Detection”. In: *CoRR* abs/2310.19304 (2023). DOI: 10.48550/ARXIV.2310.19304. arXiv: 2310.19304. URL: https://doi.org/10.48550/arXiv.2310.19304.

[Kai+21] Peter Kairouz et al. “Advances and open problems in federated learning”. In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[KMS21] Alireza Kavousi, Javad Mohajeri, and Mahmoud Salmasizadeh. “Efficient Scalable Multi-party Private Set Intersection Using Oblivious PRF”. In: *Security and Trust Management - 17th International Workshop, STM 2021, Darmstadt, Germany, October 8, 2021, Proceedings*. Ed. by Rodrigo Roman and Jianying Zhou. Vol. 13075. Lecture Notes in Computer Science. Springer, 2021, pp. 81–99. DOI: 10.1007/978-3-030-91859-0_5. URL: https://doi.org/10.1007/978-3-030-91859-0%5C_5.

[Kol+17] Vladimir Kolesnikov et al. “Practical Multi-party Private Set Intersection from Symmetric-Key Techniques”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani Thuraisingham et al. ACM, 2017, pp. 1257–1272. DOI: 10.1145/3133956.3134065. URL: https://doi.org/10.1145/3133956.3134065.

[KS22] Marcel Keller and Ke Sun. “Secure quantized training for deep learning”. In: *International Conference on Machine Learning*. 2022, pp. 10912–10938.

[Lin17] Yehuda Lindell. “How to Simulate It - A Tutorial on the Simulation Proof Technique”. In: *Tutorials on the Foundations of Cryptography*. Ed. by Yehuda Lindell. Springer International Publishing, 2017, pp. 277–346. DOI: 10.1007/978-3-319-57048-8_6. URL: https://doi.org/10.1007/978-3-319-57048-8%5C_6.

[Mar+21] Martín Abadi et al. *TensorFlow Privacy*. https://www.tensorflow.org/responsible_ai/privacy/guide. Sept. 2021.

[McM+17] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *AISTATS*. 2017, pp. 1273–1282.

[MZ17] Payman Mohassel and Yupeng Zhang. “SecureML: A system for scalable privacy-preserving machine learning”. In: *IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 19–38.

[NTY21] Ofri Nevo, Ni Trieu, and Avishay Yanai. “Simple, Fast Malicious Multiparty Private Set Intersection”. In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim et al. ACM, 2021, pp. 1151–1165. DOI: 10.1145/3460120.3484772. URL: https://doi.org/10.1145/3460120.3484772.

[Ped+11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[Pen+22] Sikha Pentyala et al. *Training Differentially Private Models with Secure Multiparty Computation*. Cryptology ePrint Archive, Report 2022/146. https://ia.cr/2022/146. 2022.

[Pin+20] Benny Pinkas et al. “PSI from PaXoS: Fast, Malicious Private Set Intersection”. In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Springer, 2020, pp. 739–767. DOI: 10.1007/978-3-030-45724-2_25. URL: https://doi.org/10.1007/978-3-030-45724-2%5C_25.

[PRR10] Manas A Pathak, Shantanu Rane, and Bhiksha Raj. “Multiparty Differential Privacy via Aggregation of Locally Trained Classifiers”. In: *Advances in Neural Information Processing Systems 23*. 2010, pp. 1876–1884.

[Ram+20] Sara Ramezanian et al. “Private membership test protocol with low communication complexity”. In: *Digit. Commun. Networks* 6.3 (2020), pp. 321–332. DOI: 10.1016/j.dcan.2019.05.002. URL: https://doi.org/10.1016/j.dcan.2019.05.002.

[Smi11] Adam Smith. "Privacy-preserving statistical estimation with optimal convergence rates". In: *Proceedings of the 43th Annual ACM symposium on Theory of Computing*. 2011, pp. 813–822.

[So+21] Jinhyun So et al. "Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning". In: *arXiv preprint arXiv:2106.03328* (2021).

[SRS17] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. "Machine learning models that remember too much". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 587–601.

[Tra+16] Florian Tramèr et al. "Stealing machine learning models via prediction APIs". In: *25th USENIX Security Symposium*. 2016, pp. 601–618.

[Tru+19] Stacey Truex et al. "A hybrid approach to privacy-preserving federated learning". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. 2019, pp. 1–11.

[vBP24] Aron van Baarsen and Sihang Pu. *Fuzzy Private Set Intersection with Large Hyperballs*. Cryptology ePrint Archive, Paper 2024/330. https://eprint.iacr.org/2024/330. 2024. URL: https://eprint.iacr.org/2024/330.

[VCE22] Jelle Vos, Mauro Conti, and Zekeriya Erkin. "Fast Multi-party Private Set Operations in the Star Topology from Secure ANDs and ORs". In: *IACR Cryptol. ePrint Arch.* (2022), p. 721. URL: https://eprint.iacr.org/2022/721.

[VCE24] Jelle Vos, Mauro Conti, and Zekeriya Erkin. "SoK: Collusion-resistant Multi-party Private Set Intersections in the Semi-honest Model". In: *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024. DOI: 10.1109/SP54263.2024.00079. URL: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00079.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. "SecureNN: 3-Party Secure Computation for Neural Network Training". In: *Proceedings on Privacy Enhancing Technologies* 2019.3 (2019), pp. 26–49. DOI: 10.2478/popets-2019-0035. URL: https://doi.org/10.2478/popets-2019-0035.

[Zha+23] Haobo Zhang et al. *A Privacy-Preserving Hybrid Federated Learning Framework for Financial Crime Detection*. 2023. arXiv: 2302.03654 [cs.LG].

4.A Differentially-private discretization of `InterimTime`

To avoid any privacy leakage, we make the process of binning the `InterimTime` feature differentially private. To do so, we first compute the DP mean of the

`InterimTime` feature for benign transactions.[4] A privacy budget of ϵ_1 is spent towards such computation. We then split the feature value range into two regions based on the above computed mean. Each region contains a distinct peak of benign samples. For each region, we compute the DP min and max value (percentile) of the `InterimTime` feature for the benign transactions and then generate 100 uniformly distributed bins for each percentile. The privacy of computation of percentile is due to [Smi11] and a privacy budget of ϵ_2 is spent for computation of one percentile. Once the bins are generated for both the regions, these are one-hot-encoded. Each value of the feature `InterimTime` is then mapped to the corresponding one-hot-encoded bin number that it falls into.

The total privacy spent for computation of DP statistics for the binning process is $(\epsilon_1 + 2 \cdot \epsilon_2)$ which follows from sequential composition of DP. In order to keep the privacy budget equal to ϵ, we divide it between ϵ_1, ϵ_2, and ϵ_m, where ϵ_m is used in training the model with discretized bins. The total ϵ is allocated as $\epsilon_1 = \epsilon \cdot 1/50$, $\epsilon_2 = \epsilon \cdot 9/100$, and $\epsilon_m = \epsilon \cdot 4/5$, thus $\epsilon = \epsilon_1 + 2 \cdot \epsilon_2 + \epsilon_m$.

4.B Changes in the malicious model

In the semi-honest model, the parties always follow the protocol specification. In the malicious model, a party is allowed to divert from this behavior. In the financial fraud detection application, one possible attack that is not addressed in our protocol might be for the data-augmenting entities involved in a transaction to collude with a user who makes a fraudulent transaction, covering their tracks by returning a fresh encryption of zero instead of randomizing the ciphertext sent by the centralized entity. One can prevent this attack by letting the data-augmenting entities provide a zero-knowledge proof that proves that the randomization is performed correctly. Note, however, that data-augmenting entities can also perform this attack by encoding the fraudulent data into the OKVS to begin with. This is not covered in the malicious model, as a party is free to choose their own inputs.

We may also assume that the centralized entity acts maliciously. For example, it could ask the data-augmenting entities to decrypt a different ciphertext than the one originating from the previous steps in the protocol execution. This too can be prevented using zero-knowledge proofs, where the centralized entity proves to the data-augmenting entities that the ciphertext they are asked to decrypt indeed corresponds to the randomized sum that they expect.

Beyond this, one can argue that the centralized entity should be limited in the number of queries that it makes: if it can perform arbitrarily many queries, then it can learn the data-augmenting entity's entire datasets. This can easily be done by data-augmenting entities, who can keep count of the number of protocol executions. Note, however, that this 'leakage' can also be seen as deliberate. For example, when it comes to the financial fraud detection application, the centralized entity must query a large number of transactions, potentially querying the same row multiple times. These all constitute valid transactions, and so the centralized

[4] We use diffprivlib [Hol+19] to compute DP mean, min, and max, and we provide bounds for clipping that are independent of the data and depend on the given problem.

entity is expected to learn whether these entities are in the datasets; the leakage is inherent to this use case.

Part B

Automatic generation of HE circuits

Chapter 5

Depth-Aware Arithmetization of Common Primitives in Prime Fields

Fully homomorphic encryption is a promising solution to secure computation because it is analogous to conventional computation. That said, current techniques still suffer from impracticalities such as 3: Arithmetization and 4: Compute-intensive. Programmers must currently translate high-level circuits into arithmetic circuits by hand or use tools that significantly limit the choice of homomorphic encryption parameters, for example by being constrained to Boolean circuits. Besides, homomorphic encryption is still orders of magnitude slower than performing computations on plaintext values.

In this chapter, we propose automatic techniques for arithmetization. By making these techniques *depth-aware*, they also lead to reduced computational requirements when evaluated using homomorphic encryption. Next to that, they exploit the algebra of prime fields, which allows one to select potentially faster parameters than when only considering Boolean circuits. When evaluated using schemes like BFV or BGV, one can compute many evaluations in parallel, leading to lower amortized run times than when using other schemes like TFHE that support fast operations on single values.

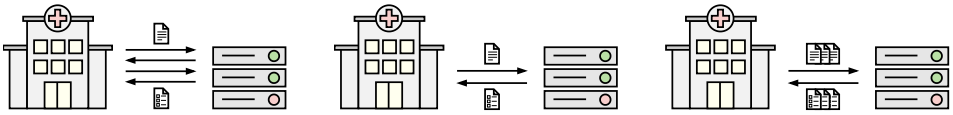
This chapter is an adaptation of the work with the same title submitted to USENIX Security 2025, authored by Jelle Vos, Mauro Conti, and Zekeriya Erkin.

5.1 Introduction

Since the advent of cloud computing, outsourced computation has become a ubiquitous tool for organizations to fulfill their computational tasks without operating and maintaining a large computational infrastructure. However, classical outsourced computation does not provide any cryptographic guarantees to the confidentiality of data sent to the cloud. As a result, the sensitive nature of some of these data hinder outsourcing computations on them. Secure outsourced computation offers the benefits of outsourced computation while providing exactly those cryptographic guarantees.

Homomorphic encryption schemes like BFV [Bra12; FV12] and BGV [BGV11] allow us to encrypt elements in $\mathbb{F}_{p^d}$ and perform non-interactive secure outsourced computation because they allow an evaluator to manipulate the encrypted data using additions and multiplications. However, a remaining challenge is to translate

complex computations into efficient circuits comprised of only additions and multiplications. We refer to this sub-problem as *arithmetization*. In this paper, we show how to do so for BFV and BGV, by introducing a concept called depth-aware arithmetization. As shown in Figure 5.1, these techniques can be used to speed up secure outsourced computation of medical data.



(a) Two-party computation techniques do not outsource the computation and they require multiple interactions.

(b) TFHE allows one medical file to be processed in 1 second after sharing a bootstrapping key.

(c) Our work using BGV allows 128 medical files to be processed in 44 seconds after sharing a key-switching key.

Figure 5.1: Secure outsourced computation of medical analyses. Our work allows batching files so they are processed more quickly.

One approach to arithmetization is to consider only Boolean circuits, so $p = 2$. This approach allows the use of existing Boolean circuit synthesis techniques, but it may result in circuits with a large number of multiplications. For example, an equality check between two 256-bit inputs requires just 8 squarings in $\mathbb{F}_{257}$ (see Sec. 5.5), but it takes 255 multiplications in $\mathbb{F}_2$.

Other works do consider arithmetization in circuits with $p > 3$, but these works either focus on minimizing the number of multiplications, known as the multiplicative size [Cow+21], or the multiplicative depth [GMT23], which is the largest number of multiplications in any path through the circuit. Minimizing the multiplicative size stands to reason because multiplications are significantly more expensive to compute than additions in all of the techniques mentioned above. However, the multiplicative depth cannot be completely ignored, because it determines the size of the cryptographic parameters of the evaluated circuit: BFV and BGV ciphertexts contain noise that grows linearly during homomorphic additions and exponentially during multiplications, and for successful decryption, this noise must stay under some bound. Parameters for these schemes are therefore chosen to be large enough so that the noise has enough room to grow, remaining under the noise limit with high probability. Large parameters negatively impact the efficiency of each homomorphic multiplication.

Inspired by the depth-size trade-off in permutation circuits proposed by Halevi & Shoup [HS14], we propose a new type of arithmetization called depth-aware arithmetization, which considers *both* a circuit's multiplicative size and multiplicative depth in the arithmetization of every high-level operation. In doing so, depth-aware arithmetization allows one to reduce the size of the parameters needed in BFV and BGV, resulting in a lower computational cost. Specifically, we study the arithmetization of deterministic high-level functions while minimizing both the multiplicative size and depth of the generated circuit. We restrict these circuits to be deterministic (so constants are truly constant) and do not allow intermediate revealing of values. We also restrict the algebraic structure to a prime field $\mathbb{F}_p$, in which any function can be expressed as an arithmetic circuit.

As a second consideration, we take into account that squaring is typically a more efficient operation than performing arbitrary multiplications. We do so by defining a metric called the multiplicative cost, which is the number of multiplications between distinct non-constant inputs plus the total squaring cost, which is the number of squarings multiplied by $0.5 \leq \sigma \leq 1.0$.

Our work is not the first to reduce the depth of arithmetic circuits. Some works [CAS17; ACS20; Lee+20; YM24] take in arbitrary arithmetic circuits and reduce their depth while increasing their size. We do not consider these generic depth reduction methods in this work for two reasons. Firstly, these methods ignore the function that is being computed, but since we have this knowledge, we exploit it. Secondly, these methods reduce the depth by distributing products of sums, while increasing the multiplicative size. However, opportunities for distributing products of sums do not arise in the circuits generated in this paper (this is more common in Boolean circuits).

The rationale behind our work is to propose algorithms for generating efficient circuits for several common primitives. These primitives can be composed into more complex circuits. In each section, we first discuss how to obtain anchor points: the points that minimize the multiplicative cost (with the multiplicative depth as a secondary objective) or the multiplicative depth (with the multiplicative cost as a secondary objective). After that, we discuss how to obtain the other solutions in the depth-cost front. At the end of each section, we perform a case study, where we use the primitive for a common practical use case.

BFV and BGV offer performance gains because they allow packing multiple inputs into a single ciphertext, enabling computations on a single thread to implicitly parallelize across all inputs. An alternative that does not allow parallelization but allows for faster individual computations comes from schemes like TFHE [Chi+20]. However, BFV and BGV offer performance gains when amortized. For example, Iliashenko & Zucca [IZ21] already showed that comparison circuits for BFV/BGV can outperform CKKS and TFHE when amortized. We show that the same holds for even more complex circuits, comparing to TFHE with optimized parameters. One might think that another alternative is the use of two-party computation, but this requires both parties to perform a significant amount of computations, so it does not successfully outsource the computation. We summarize this in Figure 5.1.

We note that our techniques may also be of use in other domains. For example, in some arithmetic garbling schemes, the multiplicative depth also plays an important role in the efficiency of a circuit [AIK11]. Arithmetization is also a fundamental problems in these protocols.

The structure of our paper is as follows. In Section 5.2, we describe our notation. In Section 5.3, we briefly review related work. In Section 5.4, we discuss the depth-aware arithmetization of sums and products. After that, in Section 5.5, we provide a MaxSAT formulation for generating exponentiation circuits. We use the exponentiation circuits to arithmetize the equality operator. In Section 5.6, we vary a parameter in two existing techniques to generate circuits for univariate polynomial evaluation. We use these circuits for arithmetizating the comparison operator. In Section 5.7, we generate circuits for AND and OR operations. We study veto voting circuits, which are essentially OR operations. In Section 5.8, we compose these primitives into larger circuits. We conclude in Section 5.9.

5.2 Notation and conventions

In this work, we typically denote circuits by upper-case letters and symbolic variables by lower-case letters. Since multiplications with constants are much cheaper to compute than other multiplications, we denote the former as e.g. $3C$ or $3 \cdot 5$, while we denote the latter using a $\times$ operator.

An arithmetic circuit $C = (V, E)$ is a directed acyclic graph consisting of variable & constant nodes, which form the leaves of the graph, and arithmetic operations. The roots of the graph are the outputs of the circuit. In this work, we consider only addition and multiplication operations, but we note that arithmetic circuits are used in various different contexts, which may allow for a larger set of arithmetic operations such as subtraction.

In many cases, we will write e.g. $C = X + Y$ when we know that C only has a single root, and it is an addition node. In this work, we do not work with the set of edges E, so we use $X \in C$ to actually denote $X \in V$. In other words, we only consider C's nodes. Putting these two shorthands together, we write $[X \times Y \in C] = \{Z \in C \mid Z = X \times Y\}$ to denote the set of all multiplications in C.

We define several metrics for arithmetic circuits below. These metrics only consider multiplications. For this reason, arithmetic circuits that also allow subtraction do not affect the results in this work.

Definition 6 (Multiplicative size)**.** The multiplicative size of a circuit or several subcircuits is the number of multiplications in these potentially-overlapping (sub)circuits:

$$\text{size}(C_1, \ldots, C_n) = |[X \times Y \in C_1] \cup \cdots \cup [X \times Y \in C_n]| \ .$$

Definition 7 (Multiplicative cost)**.** The multiplicative cost of a circuit is a weighted sum of the cost of all multiplications in a circuit. One might define the cost of squaring operations to be a factor σ of that of arbitrary multiplications, yielding:

$$\begin{aligned}\text{cost}(C_1, \ldots, C_n) = \sigma|[X \times X \in C_1] \cup \cdots \cup [X \times X \in C_n]| \\ + |[X \times Y \in C_1 \mid X \neq Y] \cup \ldots \cup [X \times Y \in C_n \mid X \neq Y]| \ .\end{aligned}$$

Definition 8 (Multiplicative depth)**.** The multiplicative depth of a circuit C is the largest number of multiplications in any path through the circuit:

$$\text{depth}(C) = \begin{cases} 0 & \text{If } C \text{ is a leaf} \\ \max(\text{depth}(X), \text{depth}(Y)) & \text{If } C = X + Y \\ 1 + \max(\text{depth}(X), \text{depth}(Y)) & \text{If } C = X \times Y \end{cases}$$

For any circuit C, there exist an infinite number of different circuits C' that perform the same computation. We denote such an equivalence as $C = C'$. An interesting question to answer is for some circuit C, what is an equivalent circuit C' that minimizes some metric (such as the ones defined above). We denote the minimal multiplicative size, cost, or depth, that can be achieved by any equivalent circuit to some circuit C as $\text{size}^*(C)$, $\text{cost}^*(C)$, and $\text{depth}^*(C)$, respectively.

5.3 Related work

We briefly go over previous works in the same order as this work, and describe their relation.

5.3.1 Arithmetization of Sums & Products

Products can be trivially expressed in an arithmetic circuit. While the multiplicative size of a product cannot be reduced, depth-aware arithmetization may rebalance a multiplication tree to reduce the multiplicative depth. This has been studied before, such as in the EVA and Ramparts compilers [Cho+21; Arc+19]. However, these compilers rebalance multiplication trees without regard for the multiplicative depths of the operands, so the result is suboptimal. We provide a simple algorithm for optimally rebalancing multiplication trees and a closed-form expression for the resulting multiplicative depth. There are also works [CAS17; ACS20; Lee+20; YM24] that show how to reduce the multiplicative depth of a circuit beyond multiplication trees by distributing products.

5.3.2 Arithmetization of Exponentiations

The problem of arithmetizing exponentiations (repeated multiplication) is equivalent to the problem of arithmetization of repeated additions. In cryptography, exponentiation circuits have been studied extensively. As a result, methods like square & multiply (also known as double & add) [HVM04], window methods [HVM04], and ones based on heuristics [Ber+89] are widely deployed. While these methods are highly efficient in generating circuits, they only optimize for the multiplicative size, meaning that the circuits themselves are not necessarily efficient. Besides that, these methods do not consider that squaring can be cheaper than arbitrary multiplications, and they ignore the cyclic nature of $\mathbb{F}_p$.

Abbas & Gustafsson [AG23] propose a depth-aware arithmetization method for exponentiations based on a mixed-integer linear program (MILP) formulation. They also show how to adapt the formulation to consider that squaring is cheaper than arbitrary multiplications. While the formulation allows one to find optimal arithmetic circuits, it is slow in practice. In Section 5.5, we translate this MILP to a MaxSAT formulation that is significantly faster to solve. We also provide several optimizations.

5.3.3 Arithmetization of Polynomial Evaluation

The arithmetization of polynomial evaluation has been studied in many previous works, but the work by Paterson & Stockmeyer [PS73] is of particular interest because it specifically considers minimizing the number of non-scalar multiplications (i.e. the multiplicative size). Paterson & Stockmeyer provide two methods, which we discuss in more detail in Section 5.6, and we show how to tweak them to obtain a depth-size trade-off.

Iliashenko et al. [IZ21; INZ21] show that for many common integer functions, it is possible to choose a convenient p such that the polynomial is efficiently

computable. The key idea is that the polynomial has a sparse structure of equally-spaced monomials apart from the leading term. This choice of p is quite restrictive. For example, for some of the functions it must hold that p is a Mersenne prime. In our work, we want to allow any choice of p.

Comparisons between two elements in $\mathbb{F}_p$ have also been studied in other works. Let us focus on $x < y$, from which the other comparisons follow easily. The approach taken by the T2 compiler [GMT23] performs an equality check for each positive case of the comparison. In other words, $\sum_{x'=0}^{p-1} \sum_{y'=x'+1}^{p-1} (x = x' \cdot y = y')$, which has optimal depth but requires a large amount on non-scalar multiplications. Iliashenko & Zucca [IZ21] show how to generate efficient circuits that only work for half of the elements in $\mathbb{F}_p$. These circuits have significantly lower multiplicative size, but a higher depth. In this work, we show how to trade off multiplicative cost and depth. We also use our formulation for finding efficient exponentiation circuits to reuse the powers that must be precomputed for polynomial evaluation, which allows us to find slightly smaller comparison circuits.

5.3.4 ORs & ANDs

ANDs are typically arithmetized using a product $x_1 \wedge x_2 \wedge \cdots \wedge x_k = x_1 \times x_2 \times \cdots \times x_k$, which leads to a circuit of depth $\lceil \log_2 k \rceil$. The OR operation follows using DeMorgan's law, which does not introduce further non-scalar multiplications. An alternative arithmetization [BI20] uses a summation and an IsNonZero check to compute such operations on many inputs.

5.4 Arithmetization of Sums & Products

Let us consider the class of arithmetic circuits consisting of only multiplications. In such a circuit, one can only reduce the number of multiplications by eliminating common subexpressions, possibly introducing a trade-off between the circuit's multiplicative depth and size. When such an arithmetic circuit does not contain common subexpressions, we cannot reduce its multiplicative size, but we may still reduce its multiplicative depth. An example can be seen in Figure 5.2. The left subfigure shows a depth-3 product, whereas the right subfigure shows a rearranged product of depth 2. This is the minimal depth that such a circuit can achieve, because a binary tree of depth d can only contain $2^d - 1$ operations, so a product of $n = 4$ distinct inputs requiring $n - 1 = 3$ binary multiplications requires $d \geq \log_2 n = 2$. This simple optimization called rebalancing has been implemented in multiple homomorphic encryption compilers [Cho+21; Arc+19].

General arithmetic circuits which also contain additions are harder to analyze. In those cases, reducing the depth beyond rebalancing requires distributing multiplications of sums. It is still possible to determine the minimal depth of such a circuit by relating it to the number of multiplicands. For this reason, we define a metric that we call the multiplicative breadth, which essentially counts the number of multiplicands in a single term (i.e. the maximum degree) when the computed polynomial is fully expanded.

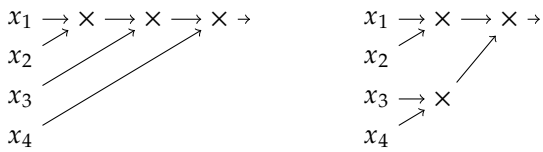


Figure 5.2: Two circuits that compute $x_1 \times x_2 \times x_3 \times x_4$. Left, an inefficient circuit of depth 3. Right, an optimal circuit that uses a binary tree to compute the product in depth 2.

Definition 9 (Multiplicative breadth)**.** The multiplicative breadth of a node in an arithmetic circuit is the largest number of multiplicands in any path of the circuit up to that node. The breadth of a node is given by:

$$\text{breadth}(C) = \begin{cases} 1 & \text{If } C \text{ is a leaf} \\ \max(\text{breadth}(X), \text{breadth}(Y)) & \text{If } C = X + Y \\ \text{breadth}(X) + \text{breadth}(Y) & \text{If } C = X \times Y \end{cases}$$

The breadth of an arithmetic circuit does not change when the circuit is rebalanced, therefore it relates to the circuit's minimal multiplicative depth. Since each multiplication can only take two operands, we have that:

$$\text{depth}^*(C) = \lceil \log_2 \text{breadth}(C) \rceil \,. \tag{5.1}$$

Conversely, it always holds that $\text{breadth}(C) \leq 2^{\text{depth}(C)}$.

In our work we do not consider depth reduction of general arithmetic circuits, but we rather study how to arithmetize several high-level operations. For this reason, we do not consider distributing multiplications of sums. As such, we can consider additions as 'optimization fences' beyond which we do not change the circuit. Even in this limited model, we show that the rebalancing operation described above can be improved by taking into account the depth of the operands. Algorithm 6 shows how to perform depth-aware rebalancing, effectively answering the question of how to optimally perform depth-aware products of distinct multiplicands.

Algorithm 6 Depth-aware product of distinct multiplicands

```
1:  procedure PRODUCT(C_1, ..., C_n)
2:      Let Q be an empty priority queue
3:      for i = 1, ..., n do
4:          Insert C_i into Q with priority depth(C_i)
5:      while |Q| ≥ 2 do
6:          Pop X and Y from Q                          ▷ Returns lowest depth
7:          d ← 1 + max(depth(X), depth(Y))
8:          Insert X × Y into Q with priority d
9:      Pop C from Q                                    ▷ There is only one C in Q
10:     return C
```

We can adapt the equation before to derive a closed-form expression of the depth of the circuit resulting from depth-aware arithmetization of a product. Since

we do not modify the subcircuits, we model them as having maximal breadth for their depth, yielding:

$$\operatorname{depth}(\textsc{Product}(C_1, \ldots, C_n)) = \left\lceil \log_2 \sum_{i=1}^{n} 2^{\operatorname{depth}(C_i)} \right\rceil . \tag{5.2}$$

Since the multiplicative size (and the cost) of such a product is $n - 1$, there is no depth-cost trade-off.

5.5 Arithmetization of Exponentiations

Exponentiations are a crucial primitive in many high-level operations. In this section, we show how to perform optimal depth-aware arithmetization of the map x^t, for a constant exponent t. Our main tool is a MaxSAT solver [MDM14], which we use to solve a reformulation of the mixed-integer linear programming (MILP) formulation by Abas & Gustafsson [AG23]. Such a solver attempts to find a variable assignment that satisfies a set of hard clauses and as many soft clauses as possible (possibly dropping some). We assign a weight to some of these soft clauses.

We first describe how to generate a minimum-cost circuit, after which we use an adapted formulation to find a minimum-depth anchor point. Having this anchor point and a lower bound on the cost of the exponentiation circuit allows us to efficiently generate the entire front. We conclude by applying our exponentiation circuits for performing equality checks.

5.5.1 Finding a Minimum-Cost Circuit

Finding minimum-cost exponentiation circuits has been studied under the name of 'addition chains' (as multiplication chains are effectively addition addition chains in the exponent). The aim is typically to find minimum-length chains, which correspond to minimizing the multiplicative size of exponentiation circuits, but some works also consider the multiplicative cost [AG23; McL21]. Much theoretical work has been done [Sch75] and many heuristics have been proposed [McL21; Ber+89]. Variants of the problem have also been studied, such as addition sequences [DLS81], which compute multiple exponentiations, reusing intermediate computations. Because exponentiations are so crucial in determining the efficiency of other high-level operations, we are looking for optimal solutions. We propose a MaxSAT formulation that is amenable to computing addition sequences and to consider precomputations provided by other computations (see Section 5.6.2).

We adapt the MILP formulation by Abbas & Gustafsson [AG23] into a MaxSAT formulation that is significantly more efficient to solve in practice. Let Boolean variables x_i represent that number i is covered in the addition chain, and let $y_{i,j}$ represent that the chain computes $i + j$. Abbas & Gustafsson define the following constraints:

1. If $y_{i,j} = 1$, then $x_i = 1$, $x_j = 1$, and $x_{i+j} = 1$.
2. Cutting away: $x_{\lceil \frac{k}{2} \rceil} \vee \cdots \vee x_{k-1} = 1$.

To minimize the size of the addition chain, we want to maximize the number of x_i that are 0. I.e. we want to maximize $\bigwedge_{i \in I} \neg x_i$. The authors also suggest replacing this objective with an objective that maximizes the number of $y_{i,j}$ that are 0, which allows taking into account that squaring is cheaper operation. In other words, it allows us to minimize the multiplicative cost.

We define $P = \{(i, j) \in [1, t]^2 : i \leq \min(j, t - j)\}$, which is the set of all ordered pairs (i, j) such that $i + j \leq t$. Our basic MaxSAT formulation is as follows:
Hard clauses:

$$
\begin{aligned}
(x_t), & \\
(\neg y_{i,j} \vee x_i), & \quad \forall (i, j) \in P \\
(\neg y_{i,j} \vee x_j), & \quad \forall (i, j) \in P \\
\left(\neg x_k \vee \bigvee_{(i,j) \in P : i+j=k} y_{i,j} \right), & \quad \forall k \in [2, t]
\end{aligned}
$$

Soft clauses:

$$
\begin{aligned}
\text{weight } 1 \ (\neg y_{i,j}), & \quad \forall (i, j) \in P : i \neq j \\
\text{weight } \sigma \ (\neg y_{i,j}). & \quad \forall (i, j) \in P : i = j
\end{aligned}
$$

We can add several cuts to this formulation to make solving it faster in practice. We add three kinds of cuts:

- Bounds from original [AG23]
- The bounds derived by Thurber & Clift [TC21]. Given an upper bound on the cost of the chain, we can use these to find lower bounds for the ith element in the chain. For our MaxSAT formulation, let $T_i(c_{\max})$ return a set of pairs (l, u) such that the ith element is bounded from below by l and from above by u for a chain with cost at most $c_{\max}$. We also have that $c_{\max} \geq \sigma s_{\min}$.
- Knowing a lower bound $s_{\min}$ on the size of the chain, we can add a cardinality constraint that $\sum_{i=2}^{t} x_i \geq s_{\min}$. This constraint can be turned into a set of clauses using multiple different techniques. We find a sequential counter approach [Sin05] to work well in practice.[1]

We can add these cuts using the following hard clauses:

$$
\begin{aligned}
\left(\bigvee_{m=\lceil \frac{k}{2} \rceil}^{k} x_m \right), & \quad \forall k \in [2, t] \\
\left(\bigvee_{m=l}^{u} x_m \right), & \quad \forall (l, u) \in T_t(c) \\
\left(\sum_{i=2}^{t} x_i \geq s_{\min} \right), & \quad \text{encoded with [Sin05]}
\end{aligned}
$$

[1] Our implementation supports the choices offered by PySAT [IMM18].

To determine $s_{\min}$ we combine three lower bounds reported by Schönhage [Sch75], where $\nu(t)$ is the Hamming weight of t:

$$s_{\min}(t) \geq \lceil \log_2(t) \rceil \,, \tag{5.3}$$

$$s_{\min}(t) \geq \lceil \log_2(t) + \log_2(\nu(t)) - 2.13 \rceil \,, \tag{5.4}$$

$$s_{\min}(t) \geq \lceil \log_2(t) + \log_3(\nu(t)) - 1 \rceil \,. \tag{5.5}$$

Finally, in a finite field, we must take into account its cyclic nature (or the resulting exponentiation circuit cannot be considered optimal). For example, $x^{62} \equiv x^{128} \pmod{67}$, but the shortest addition chain for 62 has 8 multiplications, while 128 requires 7 multiplications. We solve this problem by generating an exponentiation circuit for several $t' = t + i\phi(p)$, with $i = 1, 2, \ldots$, and selecting the most efficient.

The challenge in the solution provided above is in determining when to stop increasing i. To do so, we use monotonically growing lower bound c_{mono} on the multiplicative cost of the exponentiation circuit:

$$c_{\text{mono}}(t') = \sigma \lceil \log_2 t' \rceil \,. \tag{5.6}$$

If $c_{\text{mono}}(t')$ is greater or equal to the current best cost, we can terminate the search. Next to that, when we find a circuit with a lower multiplicative cost than before, we can lower $c_{\max}(t')$, making the formulation faster to solve and allowing us to skip targets t' for which $\sigma s_{\min}(t) \geq c_{\max}(t')$.

5.5.2 Finding a Minimum-Depth Anchor Point

One very common method for arithmetizing exponentiations is the square & multiply method, which produces a circuit as shown in Figure 5.3. As seen in the figure, this method actually produces minimum-depth circuits, seeing as a multiplication can at most double the exponent in either of its inputs, so:

$$\text{depth}^*(X^t) = \lceil \log_2 t \rceil \,. \tag{5.7}$$

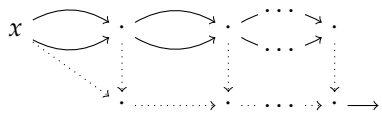


Figure 5.3: Square & multiply method for computing x^t.

While square & multiply produces a minimum-depth circuit, it does not necessarily produce a minimum-depth anchor point (i.e. a circuit that is minimal in depth and secondarily minimal in cost). In other words, a circuit may exist with the same depth but a lower multiplicative cost. To find such an anchor point, we make another call to the MaxSAT formulation, but this time we provide the following constraints:

- The maximum depth is $\lceil \log_2 t \rceil$.
- The maximum cost is $c_{\max} = \sigma\lfloor \log_2 t \rfloor + \nu(t) - 1$, corresponding to the cost of square & multiply.

What remains, is to modify the MaxSAT formulation to incorporate a bound on the depth $d_{\max}$ of the exponentiation circuit. We introduce the following sets of hard clauses:

$$\begin{aligned}
(d_{k,m+1} \vee \neg d_{i,m} \vee \neg y_{i,j}), &\quad \forall (i,j) \in P,\ \forall m \in [0, d_{\max}] \\
(d_{k,m+1} \vee \neg d_{j,m} \vee \neg y_{i,j}), &\quad \forall (i,j) \in P,\ \forall m \in [0, d_{\max}] \\
(\neg d_{k,d_{\max}+1}), &\quad \forall k \in [2, t] \\
(d_{1,0}). &
\end{aligned}$$

These clauses encode the depth of an exponent as a Boolean vector, such that the highest-index Boolean that is true represents the depth of that exponent. By forcing the $d_{\max} + 1$th Boolean to be false, we ensure that the depth limit is not exceeded. This is a different encoding than the one used by Abbas & Gustafsson [AG23], which uses integers to denote the depth (as they use a MILP solver).

5.5.3 Finding Circuits on the Depth-Cost Front

We can generate circuits on the depth-cost front using the same method that we described for finding an anchor point given a minimal-depth circuit with suboptimal cost. We do so by incrementally going through all such circuits, from least to highest depth. For the the maximum cost, we can use the current best cost. We present our approach in Algorithm 7, in which we call our MaxSAT formulation as AddChain$(t, d_{\max}, c_{\max}, \sigma, s_{\min})$, which returns a circuit satisfying the constraints or $\perp$ if no circuit could be found.

Algorithm 7 Depth-aware product of distinct multiplicands

```
1:  procedure GenExpFront(C)
2:      Find and yield C such that cost(C) = cost*(C)
3:      d ← ⌈log2 t⌉
4:      c ← σ⌊log2 t⌋ + ν(t) − 1
5:      while c < cost*(C) and d < depth(C) do
6:          Compute s_min using (5.3), (5.4), and (5.5)
7:          C′ ← AddChain(t, d, c, σ, s_min)
8:          if C′ ≠ ⊥
9:              yield C′
10:             c ← cost(C′)
11:         d ← d + 1
```

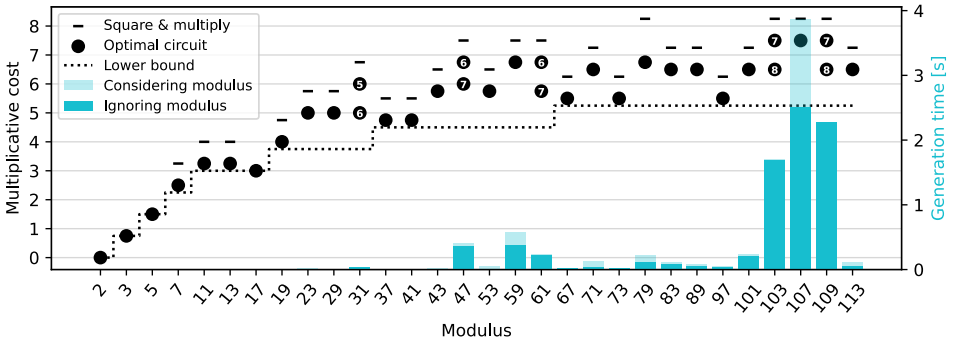


Figure 5.4: Equality circuits generated using square & multiply and our MaxSAT formulation, where $\sigma = 0.75$. Square & multiply is only optimal when p is of the form $2^k + 1$. When we find a depth-cost trade-off, we denote the depth in the markers. The run time of our algorithm is hard to predict, but it increases with the modulus p. In some cases, ignoring the modulus makes a large difference in generation time, but the result is not guaranteed to be optimal.

5.5.4 Case Study: Equality Checks

As explained by Iliashenko & Zucca [IZ21], equality checks can be arithmetized as $[x = y] = 1 - (x - y)^{p-1}$. The cost of such an operation is almost exclusively determined by the exponentiation circuit, as it is the only operation requiring multiplications. In Figure 5.4, we plot the multiplicative cost of the optimal exponentiation circuits we found using our MaxSAT formulation for different prime moduli p and for fixed $\sigma = 0.75$. We also show how long it took to generate these circuits, with and without consideration of the cyclic nature of $\mathbb{F}_p$. For the moduli in Figure 5.4, the circuits generated by ignoring or considering the modulus are the same, but it is significantly more efficient to ignore the modulus. One can interpret the 'considering modulus' generation time as the time it takes to prove optimality.

5.6 Arithmetization of Polynomial Evaluation

For many high-level operations there is not a straightforward arithmetization. For example, checking if a field element is within a given range can be expressed as a large number of equality operations but this is inefficient. In these situations, it is typical to interpolate a polynomial and to find an efficient circuit to evaluate it. In this section, we show how to perform depth-aware arithmetization for univariate polynomial evaluation. These cover many common operations including comparisons, which we highlight in our case study at the end of this section.

When it comes to the multiplicative cost of polynomial evaluation circuits, we know that the multiplicative cost of a degree-d polynomial is at least as high as that of an exponentiation circuit with target t. Next to that, Paterson &

Stockmeyer [PS73] provide an asymptotic bound:

$$\text{cost}^*\left(x^d\right) \leq \text{cost}^*\left(\sum_{i=0}^{d} c_i x^i\right) \leq O(\sqrt{d}) \ . \tag{5.8}$$

In fact, Paterson & Stockmeyer already provide two algorithms that generate circuits with the same asymptotic complexity. We discuss these two algorithms later on.

The multiplicative depth of polynomial evaluation circuits can also be bounded. To achieve the minimal depth, we can simply compute all monomials and evaluate the polynomial using a linear combination. So:

$$\text{depth}^*\left(\sum_{i=0}^{d} c_i x^i\right) = \lceil \log_2(d) \rceil \ . \tag{5.9}$$

This is an equality because we cannot evaluate x^d with fewer multiplications. In Paterson & Stockmeyers's methods, this is equivalent to choosing $k = d$. Our key idea for generating circuits that trade off multiplicative depth and cost is to vary this parameter k.

5.6.1 Baby-Step Giant-Step

The baby-step giant-step method was one of the two algorithm proposed by Paterson & Stockmeyer [PS73], but we refer to it with this name because it is colloquially known as such in the cryptography community. It is also known as the two-level evaluation method [Deg+24].

The algorithm, parameterized by an integer $1 \leq k \leq d$, starts by precomputing the monomials $X^2, X^3, \ldots, X^k$. It will later use these precomputed powers to evaluate a $k-1$-degree polynomial without performing any more multiplications. In this work, we also want to minimize the multiplicative depth, so we do not use sequential multiplications to compute these powers. Instead, we start by computing X^2 and use it to compute X^3 and X^4. We then use X^4 to compute $X \times X^4 = X^5, X^2 \times X^4 = X^6, \ldots, X^4 \times X^4 = X^8$, etc. Given these precomputed powers, the key idea behind this algorithm is the following identity:

$$\left[\sum_{i=0}^{d} c_i X^i\right] \leftarrow X^k \left[\sum_{i=0}^{d-k} q_i X^i\right] + \left[\sum_{i=0}^{k-1} r_i X^i\right] \ , \tag{5.10}$$

where the rightmost polynomial can be evaluated using only additions and constant multiplications. In other words, the polynomial can be evaluated by taking approximately $\frac{d}{k}$ giant steps after computing k baby steps. Paterson & Stockmeyer show that this method requires approximately $2\sqrt{d}$ multiplications for the right choice of k. This makes it asymptotically optimal in terms of the multiplicative cost and size. Due to its sequential nature, the circuits generated by this method are typically larger in depth than the circuits generated by the other two methods that we discuss.

5.6.2 Paterson & Stockmeyer's method

Paterson & Stockmeyer also propose a method that evaluates polynomials of a specific degree in $\sqrt{2d} + O(\log d)$ non-constant multiplications for the right choice of k. This method is defined for monic polynomials (i.e. the leading coefficient is 1) of degree $d = (2^n - 1)k$, but it can be adapted to evaluate any polynomial by extending it to the next monic polynomial of the correct degree (or using a constant multiplication if it is a non-monic polynomial of the correct degree). We can then remove this added monomial from the final result by computing it and subtracting it or by adapting the coefficients.

Paterson & Stockmeyer's method [PS73] works by reducing the evaluation of a degree-$(2^n - 1)k$ monic polynomial to the evaluation of two monic polynomials of degree $(2^{n-1} - 1)k$ and a polynomial of degree $k - 1$ using the following identity:

$$\left[X^{(2^n-1)k} + \sum_{i=0}^{(2^n-1)k-1} c_i X^i\right] \leftarrow \left(X^{2^{n-1}k} + \left[\sum_{i=0}^{k-1} c'_i X^i\right]\right)\left[X^{(2^{n-1}-1)k} + \sum_{i=0}^{(2^{n-1}-1)k-1} q_i X^i\right] + \left[X^{(2^{n-1}-1)k} + \sum_{i=0}^{(2^{n-1}-1)k-1} r_i X^i\right], \quad (5.11)$$

where the square brackets group together the terms of a polynomial. The coefficients of these smaller polynomials can be obtained using a Euclidean division. Note that the polynomial of degree $k - 1$ can be computed using the precomputed powers without any multiplications. Note that where the previous method only precomputes monomials $X^2, X^3, \ldots, X^k$, this method must also precompute monomials $X^{2k}, X^{4k}, X^{2^{n-1}k}$, which requires $n - 1$ squarings.

As described previously, the method can be extended to any polynomial of degree-d by padding it with a monomial $(2^n - 1)k \geq d$, which is of the correct degree. However, we must compensate for this added monomial in the final result. If it holds that $i = (2^n - 1)k \mod \phi(p) \leq d$, where $\phi()$ is the totient function, then we can easily compensate for it by decrementing the i-th coefficient. Otherwise, we must compute the monomial separately and subtract it at the end.

For the case that we must compute the padding monomial separately, we slightly modify the MaxSAT formulation described in Section 5.5 to take into account that the polynomial evaluation circuit already precomputes a large number of monomials. We ensure that these monomials count for free towards the cost of the addition chain, while still considering their depth. We do so by adding new variables z_k that represent using previously-computed power X^k. When they are enabled, they incorporate the fixed depth of the precomputed power. Given precomputed powers $t_1, \ldots, t_n$ with depths $d_1, \ldots, d_n$, we add the following hard clauses:

$$(d_{t_i,d_i}, \neg z_{t_i}), \quad \forall i \in [1, n]$$

Next to that, we adapt the following hard clause in the original formulation to allow x_k to be true when z_k is:

$$\left(\neg x_k \vee z_k \vee \bigvee_{(i,j)\in P: i+j=k} y_{i,j}\right). \quad \forall k \in \{t_1, \ldots, t_n\}$$

We also have to remove the cuts described in Section 5.5.1 from the formulation, as they do not apply to depth-constrained circuits.

5.6.3 Our work: Divide & conquer

We propose a new method for evaluating univariate polynomials of any degree inspired by Paterson & Stockmeyer's method. While our method does not achieve as small of a multiplicative cost, it achieves a low multiplicative depth. It is essentially a simplified version of Paterson & Stockmeyer's method that retains the divide & conquer strategy. The key idea is to split evaluation of a degree-$2^n k$ polynomial into the evaluation of two degree-$2^{n-1}k$ polynomials:

$$\left[\sum_{i=0}^{d} c_i X^i\right] \leftarrow X^{2^{n-1}k}\left[\sum_{i=0}^{d-(2^{n-1}k-1)} q_i X^i\right] + \left[\sum_{i=0}^{2^{n-1}k-1} r_i X^i\right], \tag{5.12}$$

where $d \leq 2^n k$. This method requires the same precomputations as Paterson & Stockmeyer's method.

We briefly analyze the cost and depth of the circuits generated by our method. Let $N(d)$ denote the cost of computing a degree-d polynomial using our method when we have already computed the precomputations. We have:

$$\begin{aligned}
N(2^n k) &\leq \begin{cases} 0 & \text{If } n = 0 \\ 1 + 2N(2^{n-1}k) & \text{If } n > 0 \end{cases}, && (5.13)\\
&\leq 1 + 2(1 + 2N(2^{n-2}k)), && (5.14)\\
&= 3 + 4N(2^{n-2}k), && (5.15)\\
&\leq 2^i - 1 + 2^i N(2^{n-i}k), && (5.16)\\
&\leq 2^n - 1 + 2^n 0, && (5.17)\\
&= 2^n - 1. && (5.18)
\end{aligned}$$

If it takes $k-1$ multiplications to compute $X^2, \ldots, X^k$ and $n-1$ squarings to compute $X^{2k}, X^{4k}, \ldots, X^{2^{n-1}k}$, then the total cost of our circuit C is:

$$\text{cost}(C) \leq k + n + 2^n - 3 \leq k + \log_2\left(\left\lceil \frac{d}{k} \right\rceil\right) + \left\lceil \frac{d}{k} \right\rceil. \tag{5.19}$$

The depths of precomputations X^i for $i = 2, \ldots, k$ are $\lceil \log_2 i \rceil$, and the depths of $X^{2^i k}$ for $i = 1, \ldots, n-1$ are $\lceil \log_2 k \rceil + i$. As a result, the depth of the circuit is:

$$\text{depth}(C) \leq \lceil \log_2 k \rceil + n \leq \lceil \log_2 k \rceil + \left\lceil \frac{d}{k} \right\rceil. \tag{5.20}$$

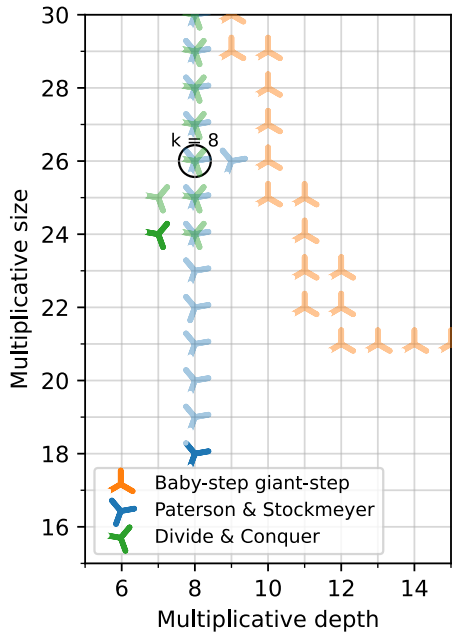


Figure 5.5: Polynomial evaluation circuits for computing $x \pmod 7$ in $\mathbb{F}_{127}$. This is a degree-126 polynomial, so Paterson & Stockmeyer's parameter for minimizing multiplicative size is $k = \sqrt{0.5 \cdot 126} \approx 8$. However, the optimum occurs when $k = 9$. Divide & conquer leads to a lower depth.

From this analysis it is clear that choosing a large value of k reduces the depth significantly.

5.6.4 Finding Circuits on the Depth-Cost Front

The three methods described above all achieve a different depth-cost trade-off when varying k. Our depth-aware arithmetization method for polynomial evaluation is to simply try all three methods on all values $1 \leq k \leq d$. It turns out that, while it is possible to compute the optimal k for reducing the multiplicative cost, there are cases where other values of k achieve a lower cost. In Figure 5.5 we highlight such a situation. In this figure, we show all circuits computing $x \pmod 7$ in $\mathbb{F}_{127}$ that we can generate by varying k. While Paterson & Stockmeyer show that $k = 8$ minimizes the multiplicative cost, it turns out that we can achieve a significantly better circuit using $k = 9$.

5.6.5 Case Study: Comparisons

We show that our depth-aware arithmetization method allows to generate a front of circuits that trade off multiplicative depth and cost, even for complex operations such as comparisons. We use the technique proposed by Iliashenko &

Zucca [IZ21] for performing comparisons between half of the elements in the field $\mathbb{F}_p$ using a univariate polynomial evaluation. By computing the leading term of the polynomial separately, the remainder of the polynomial can be decomposed so that its degree is only $\frac{p-1}{2}$.

Another method for generating such circuit is implemented in the T2 compiler [GMT23], in which the comparison is implemented as a number of equality checks:

$$[X < Y] = \sum_{a=\frac{p+1}{2}}^{p} [(X - Y) = a] = \sum_{a=\frac{p+1}{2}}^{p} 1 - (X - Y - a)^{p-1} \,. \tag{5.21}$$

We provide an optimistic implementation of this technique in which we use the minimal-cost exponentiation circuit to implement the equality checks.

We also provide an optimistic implementation of the work by Iliashenko & Zucca [IZ21], in which we only use the Paterson & Stockmeyer method with their choice of k with the intent of minimizing the multiplicative cost. One problem is that their proposed way to compute the final term requires a certain polynomial degree, but it is not possible for all p to find a certain k. Instead, we use our method for finding the optimal addition chain given precomputed powers to compute the leading term of the univariate polynomial.

In Table 5.1 we provide an overview of different methods for generating comparison circuits. We find that our work consistently finds circuits in the depth-size front, but the other methods do so too. We mark values on the front in bold. For example, while the T2 compiler finds circuits with large size, their depth is minimal. We find that the method by Iliashenko & Zucca does not outperform ours, unless we apply common subexpression elimination. In some cases, this allows the method to find circuits on the front. We evaluate the run time of these circuits using fhegen [Mon+23] to generate parameters and execute the circuits using HElib on a Threadripper 7970X CPU, using only one thread to compile and evaluate the circuits. The machine has 4x64GB DDR5 RAM, but only a fraction was used.

Table 5.1: Comparison circuits for different moduli p with squaring cost $\sigma = 1.0$. Run times are in seconds, and were averaged over 10 iterations. Our circuits often outperform previous techniques. *We used CSE to reduce these circuits' size.

Method	$p = 29$			$p = 43$			$p = 61$			$p = 101$			$p = 131$		
	Depth	Size	Time	Depth	Size	Time	Depth	Size	Time	Depth	Size	Time	Depth	Size	Time
T2 [GMT23]*	**5**	**84**	1.16	**6**	**147**	4.86	7	210	6.79	**7**	**400**	11.71	8	520	16.95
IZ21 [IZ21]	7	12	0.44	8	18	0.68	9	14	0.55	**8**	**16**	**0.60**	11	29	1.33
IZ21 [IZ21]*	7	12	0.44	8	16	0.59	**9**	**13**	0.52	**8**	**16**	0.63	11	26	1.18
Our work	**6**	**11**	0.41	**7**	**12**	**0.44**	**7**	**15**	0.59	**8**	**16**	0.62	**8**	**20**	**0.84**
	7	**10**	**0.38**				**8**	**14**	**0.51**						

5.7 Arithmetization of ANDs and ORs

Finally, we study the depth-aware arithmetization of AND and OR operations. The typical arithmetization of an AND operation is to treat it as a product:

$$X_1 \wedge \cdots \wedge X_k = X_1 \times \cdots \times X_k \ . \tag{5.22}$$

As shown in Section 5.4, there is a single optimal circuit C_1 to compute this product. It has the following properties:

$$\text{cost}(C_1) = k - 1 + \text{cost}(X_1, \ldots, X_k) \ , \tag{5.23}$$

$$\text{depth}(C_1) = \left\lceil \log_2 \sum_{i=1}^{k} 2^{\text{depth}(X_i)} \right\rceil \ . \tag{5.24}$$

OR operations are sometimes arithmetized as follows:

$$X_1 \vee \cdots \vee X_k = (X_1 + \cdots + X_k)^{p-1} \ , \tag{5.25}$$

where x^{p-1} maps $0 \mapsto 0$ and $\{1, \ldots, p-1\} \mapsto 1$. Note that this arithmetization is only guaranteed to work when $k < p$, otherwise the result of the summation might wrap around the modulus. Let circuit C_2 be a circuit that evaluates this arithmetization, which first sums the operands and then uses another circuit C_{exp} for exponentiation by $p - 1$. Then, $C_2(C_{\text{exp}})$ has the following properties:

$$\text{cost}(C_2(C_{\text{exp}})) = \text{cost}(C_{\text{exp}}) + \text{cost}(X_1, \ldots, X_k) \ , \tag{5.26}$$

$$\text{depth}(C_2(C_{\text{exp}})) = \text{depth}(C_{\text{exp}}) + \max_{i=1,\ldots,k} \text{depth}(X_i) \ . \tag{5.27}$$

While this method allows varying the depth and size using different circuits for C_{exp}, this only provides minimal variance.

DeMorgan's law provides a bidirectional transformation between AND and OR circuits that does not increase the size or depth because it only requires negation, which does not require non-scalar multiplications:

$$X_1 \wedge \cdots \wedge X_k = \overline{\overline{X_1} \vee \cdots \vee \overline{X_k}} \ . \tag{5.28}$$

So, either of the two arithmetizations above can be used for AND and OR operations at the same depth and size cost. In fact, they can be composed to achieve a hybrid arithmetization. This allows one to trade off depth and size. It also allows reaching smaller sizes than what could be reached by a non-hybrid arithmetization.

We cannot prove that minimizing the depth and size of the hybrid arithmetization described above coincides with minimizing the depth and size of all potential arithmetic circuits for ANDs and ORs. That said, we argue that our method is a useful heuristic.

5.7.1 Finding a Minimum-Cost Circuit

It is easy to see that if $k < p$, then $\text{cost}(C_2(C_{\text{exp}})) < \text{cost}(C_1) \iff \text{cost}(C_{\text{exp}}) < k-1$. So in this case, it is easy to decide the minimum-cost circuit. Let $N(k)$ represent

the minimal multiplicative cost of a circuit for the hybrid arithmetization of an AND or OR operation with k operands, and let c denote the multiplicative cost of C_{exp}. We have:

$$N(k) = \min(c, k-1) \quad \text{if } k \leq p-1 \,. \tag{5.29}$$

When $k \geq p$, we must consider a hybrid arithmetization. Notice that the cost of the smallest hybrid circuit $C_3(C_{\text{exp}})$ grows monotonically with k. So, if it holds that $\text{cost}(C_{\text{exp}}) \leq p-1$, we can perform $C_2(C_{\text{exp}})$ on $p-1$ operands (e.g. $X_1, \ldots, X_{p-1}$) to obtain a new problem with $k-(p-1)$ operands. It turns out that $\text{cost}^*\left(C_{\text{exp}}\right) \leq p-1$ always holds.

Using the strategy described above, we get that:

$$\begin{aligned} N(k) &= c + N\left(k-(p-1)+1\right) \,, && (5.30)\\ &= 2c + N\left(k-p-(p-1)+1\right) \,, && (5.31)\\ &\vdots && (5.32)\\ &= rc + N\left(k-r(p-1)+r\right) \,, && (5.33)\\ &= rc + N\left(k+r(2-p)\right) \,. && (5.34) \end{aligned}$$

We reach the base case when $k + r(2-p) \leq p-1$. This happens when $r = \lceil \frac{p-1-k}{2-p} \rceil$, so we have:

$$\text{cost}^*\left(C_3(C_{\text{exp}})\right) = N(k) = \left\lfloor \frac{k}{p} \right\rfloor c + \min\left(c, k - \left\lfloor \frac{k}{p} \right\rfloor p - 1\right) \tag{5.35}$$

Notice that increasing c always increases the total multiplicative cost, apart from the case where $k < p$ and $c \geq k-1$, in which case c does not influence the result. We conclude that to minimize $N(k)$, c needs to be minimal.

5.7.2 Finding Circuits on the Depth-Cost Front

In minimizing the multiplicative depth of the circuit, we define a useful metric called fullness. This metric captures both the depth of the circuit and how many multiplications can still be absorbed by the multiplication tree in the outer layer of the circuit without increasing the circuit's depth.

Definition 10 (Fullness)**.** The fullness is defined as:

$$\begin{aligned} \text{fuln}(X+Y) &= 2^{\max(\text{depth}(X),\text{depth}(Y))} \\ \text{fuln}(X \times Y) &= \text{fuln}(X) + \text{fuln}(Y) \\ \text{fuln}(v) &= 1 \end{aligned}$$

Notice that:

$$\text{depth}(C) = \lceil \log_2 \text{fuln}(C) \rceil \,.$$

To find a minimum-depth anchor point, we put forward a recursive algorithm that finds a circuit for performing an AND operation while satisfying the constraint that the fullness is at most f, and the cost is less than c. We present it in Algorithm 8,

in which cost(C) ignores the cost of subcircuits $X_1, \ldots, X_k$. The algorithm also inputs E, which is a collection of exponentiation circuits that are on the Pareto front, and p, the order of the prime field.

Our recursive algorithm is essentially a bounded search. We use the bounds derived above to decide whether certain branches are not worth exploring. By starting with $f = 2^d$ for $d = \lceil \log_2 \text{fuln}(X_1) \rceil$, where X_1 is the operand with the highest fullness, we can iteratively increment d until the algorithm finds a circuit. This first circuit is a minimum-depth anchor point because the algorithm outputs the minimal cost circuit for this fullness bound f.

We can keep going in the fashion described above, incrementing d, to generate the entire depth-cost front. Since it is easy to compute $\text{cost}^*\big(C_3(C_{\text{exp}})\big)$, we know when to stop the search. Note that while we describe the algorithm to compute a circuit for an AND operation, the algorithm for OR operations follows almost identically: For OR operations, one must apply DeMorgan's law.

5.7.3 Case Study: Veto Voting

We study the problem of veto voting, where multiple parties submit a Boolean value, indicating whether they veto or not. If no one vetoes, the result should be false. If anyone vetoes, the result should be true. This is exactly an OR operation. We consider the setting where we do not know a bound on the possible number of vetoes.

In Figure 5.6, we demonstrate the circuits that our algorithm generates for two values of p when the number of operands grows. It is clear that for almost every number of operands, there exists a cost-depth trade-off. What is more, there is also a trade-off between different values of p. Whereas a larger value of p allows one to find circuits with fewer multiplications when the number of operands grows, there are still cases where one might favor a smaller p as it provides a better depth-cost trade-off. For example, when there are 13 operands, $p = 7$ permits a depth-4 circuit at 10 multiplications, while $p = 13$ requires 12 multiplications. Finally, notice that there are only a few cases where computing an OR operation using a C_1 circuit is necessary to achieve a minimum depth. In many other cases, we can achieve the same minimum depth with far fewer multiplications.

5.8 Depth-Aware Composition

In the previous sections, we put forward methods for the depth-aware arithmetization of several common primitives, but many interesting circuits emerge as the composition of these primitives. In this section, we show how to perform depth-aware arithmetization for high-level circuits that compose multiple primitives.

Suppose we have a circuit $X^{31} < Y^{31}$. We can generate a front for the exponentiation circuits of X^{31} and Y^{31}, but at that point we are stuck, because our method for arithmetizing comparisons inputs subcircuits rather than two fronts of circuits. For composition, we propose the following heuristic: we generate a new Pareto front in which we try all possible combinations of input arithmetizations. This is a

Algorithm 8 Finds an `AND` circuit with fullness $\leq f$ and minimal cost $< c$, returning $\perp$ if it cannot be found.

procedure AND($X_1, \ldots, X_k, f, c, E, p$)
  Ensure that $\text{fuln}(X_1) \geq \cdots \geq \text{fuln}(X_k)$
  if $k = 1$ ▷ *Base cases*
    if $\text{fuln}(X_1) \leq f$ and $c > 0$
      return X_1
    return $\perp$
  if $f < 1$ or $c \leq 0$
    return $\perp$
  if $\text{cost}^*(X_1 \wedge \cdots \wedge X_k) \geq c$
    return $\perp$
  $C_{\text{out}} = \perp$
  for $C_{\text{exp}} \in E$ **do**
    $C = X_1 \times \cdots \times X_k$ ▷ *C_1 circuit*
    if $\sum_{i=1}^{k} \text{fuln}(X_i) \leq f$ and $\text{cost}(C) < c$
      $C_{\text{out}} \leftarrow C, c \leftarrow \text{cost}(C)$
    $f_{\text{exp}} \leftarrow 2^{\lceil \log_2 f \rceil - \text{depth}(C_{\text{exp}})}$ ▷ *Max fuln for C_2*
    if $k < p$
      $C \leftarrow \overline{C_{\text{exp}}(\overline{X_1} + \cdots + \overline{X_k})}$ ▷ *C_2 circuit*
      if $\bigwedge_{i=1}^{k} \text{fuln}(X_i) \leq f_{\text{exp}}$ and $\text{cost}(C) < c$
        $C_{\text{out}} \leftarrow C, c \leftarrow \text{cost}(C)$
      continue
    if $\bigwedge_{i=1}^{k} \text{fuln}(X_i) \leq f_{\text{exp}}$ ▷ *C_2 works for all X_i*
      if $\text{cost}(C_{\text{exp}}) \geq c$
        continue
      cache $\leftarrow \{\}$
      for $i = 1, \ldots, k-1$ **do**
        $C' \leftarrow \overline{C_{\text{exp}}(\overline{X_i} + \cdots + \overline{X_{i+p-2}})}$
        $X \leftarrow C', X_1, \ldots, X_{i-1}, X_{i+p-1}, \ldots, X_k$
        if $\{\text{fuln}(x) \mid x \in X\} \in$ cache
          continue
        Add $\{\text{fuln}(x) \mid x \in X\}$ to cache
        $C \leftarrow \text{AND}(X, f, c - \text{cost}(C_{\text{exp}}), E, p)$
        if $C \neq \perp$
          $C_{\text{out}} \leftarrow C, c \leftarrow \text{cost}(C)$
    else ▷ *We can isolate X_i that must use C_1*
      Find t s.t. $\text{fuln}(X_t) > f_{\text{exp}}, \text{fuln}(X_{t+1}) \leq f_{\text{exp}}$
      if $t = 0$ or $t \geq c$
        continue
      $f_{\text{new}} \leftarrow f - \sum_{i=1}^{t} \text{fuln}(X_i)$
      $C' \leftarrow \text{AND}(X_{t+1}, \ldots, X_k, f_{\text{new}}, c - t, E, p)$
      if $C' \neq \perp$
        $C \leftarrow C' \times X_1 \times \cdots \times X_t$
        $C_{\text{out}} \leftarrow C, c \leftarrow \text{cost}(C)$
  return C_{out}

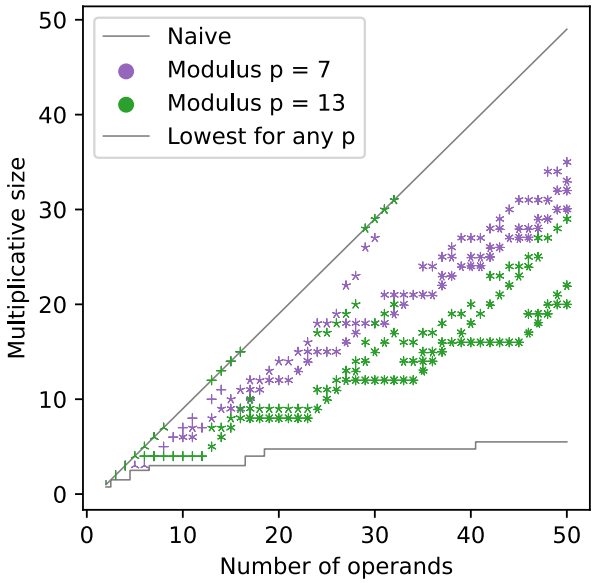


Figure 5.6: Circuits computing an OR operation with $\sigma = 1.0$, for a growing number of operands. The number of ticks on a marker indicates the depth of the circuit. Depending on the depth that one wants to achieve and the number of operands, it is better to choose $p = 7$ or $p = 13$.

heuristic because we do not change the arithmetizations of the inputs, even when this could lead to a lower cost or depth.

Because of this heuristic, composing two optimal sub-circuits does not necessarily lead to an optimal composition. For example, the condition $X^{10} = 0$ is equivalent to $X = 0$, meaning that straightforward composition leads to a circuit that is unnecessarily large. On the other hand, it often infeasible to arithmetize a complex high-level circuit to optimality because its search space is astronomical.

While the method described above offers a generic solution of dealing with composition, it can be inefficient. For example, when we want to arithmetize $X^{31} + Y^{31}$, we know that it is never better to choose a subcircuit for X^{31} with a lower depth as the subcircuit for Y^{31} and vice versa. In those cases, we do not exhaustively try all combinations, but we iteratively increment a depth limit and choose the lowest-cost subcircuits that still satisfy the depth limit.

Finally, one might consider heuristics that cut away even more solutions. For example, increasing a circuit's depth by one layer while saving one multiplication may not be worth it in practice. We do not implement such a heuristic, and leave it to future work.

To highlight the effectiveness of our methods, we apply them to a practical example that composes all of the primitives described in this work. Specifically, we evaluate them on the *cardio* circuit as proposed by Carpov et al. [Car+16] and used as a benchmark in other works [VJH21]. The circuit computes a number of predicates relating to a person's cardiac health and returns how many evaluate to

true. These predicates involve comparisons, such as checking whether a person's weight is smaller than its height - 90. We also consider a variant of this circuit that we call *cardio-elevated*, which only returns if any of the risk factors were true. In other words, we compute an OR over all the predicates.

In Table 5.2 we present the results of our methods applied to the *cardio* and *cardio-elevated* circuits for a fixed value $p = 257$, since all values fit under this modulus. We report the fronts that our methods generated for different costs of squaring σ, and how long these fronts took to generate (after implementing several run time optimizations). We do not take the cyclic nature of $\mathbb{F}_p$ into account for the exponentiations in padding the polynomials to make it run in reasonable time, and since these are unlikely to produce significantly better results for $p = 257$. We show that the *cardio* circuit can be evaluated in 419 multiplications.

Table 5.2: Fronts generated by our methods for the *cardio* and *cardio-elevated* circuits, displaying the cost-depth trade-off.

Cardio risk assessment				**Cardio elevated risk**			
Depth	$\sigma = 1.0$	$\sigma = 0.75$	$\sigma = 0.5$	Depth	$\sigma = 1.0$	$\sigma = 0.75$	$\sigma = 0.5$
Gen. (s)	*83*	*90*	*84*		*81*	*75*	*84*
11	419.0	389.0	359.0	14	428.0	396.0	364.0
12		384.0	349.0	15	·	389.0	350.0
13		374.0	329.0	16	·	383.0	338.0
				·	·	·	·
				19	427.0	·	·
				·		·	·
				21		380.0	333.0

Notice that if we solely optimize multiplicative cost/size, the resulting circuits may be wasteful in terms of the multiplicative depth. A good example is in the *cardio-elevated* circuit when $\sigma = 1.0$: If we would only focus on multiplicative cost, we would save 1 multiplication at the cost of 5 layers of depth. Moreover, optimizing both multiplicative cost *and* depth allows one to save multiplicative cost on branches of the circuit that do not contribute to the multiplicative depth, unlike what happens when optimizing for depth in isolation.

Finally, we measure the run times of our circuits using the same machine described in Section 5.6, and compare these to the equivalent circuits implemented in TFHE-rs [2]. We use unsigned 256-bit high-level integers to generate these circuits in TFHE. The results are in Table 5.3, in which we marked the fastest amortized run time in bold. We conclude that the lower-cost circuits are not always better, and that exploring the trade-off between cost and depth enables finding more efficient circuits in practice.

[2]We used version 0.11: https://docs.zama.ai/tfhe-rs

Table 5.3: Run times averaged over 10 iterations; $\sigma = 0.75$. Our circuits outperform TFHE when the run time is amortized.

Circuit	Slots	Run time (s)	Amortized (s)
Cardio risk assessment			
Depth-11	128	43.98	**0.34**
Depth-12	128	50.14	0.39
Depth-13	128	50.87	0.40
TFHE	1	0.97	0.97
Cardio elevated risk			
Depth-14	128	57.26	0.45
Depth-15	128	57.56	0.45
Depth-16	128	56.53	**0.44**
Depth-21	128	73.78	0.58
TFHE	1	0.49	0.49

5.9 Conclusion

In this work, we introduced the concept of depth-aware arithmetization, in which we generate arithmetic circuits for high-level operations while considering the trade-off between multiplicative depth and multiplicative cost. We proposed methods for the depth-aware arithmetization of exponentiations, polynomial evaluation, and AND/OR operations. In turn, these primitives allow one to perform equality checks, comparisons, and perform operations such as veto voting. They may also be composed into larger circuits.

Our methods have limitations. For example, they can take minutes to arithmetize circuits with only a handful of comparisons. Moreover, they are not necessarily optimal: we only provide optimal methods for exponentiation circuits.

There is still room for future work. One may look for:

- Faster methods for generating optimal addition chains with depth constraints and/or precomputed values.
- An optimal method for polynomial evaluation, although this may be as hard as solving a system of multivariate polynomials.
- Other polynomial evaluation methods, e.g. mixing or generalizing the methods that we use in this work.
- An optimal method for AND/OR operations, or a proof that our current approach is optimal.
- Efficient ways of composing arithmetized primitives.
- Methods for arithmetizing multiple polynomial evaluations at once, reusing the precomputed powers across evaluations.

- Experimenting with different plaintext moduli p, which allows for a trade-off between the number of slots and the size of the generated circuits.

Our work paves the way to make several secure computation tasks more efficient and more user-friendly. For example, these algorithms can be used to automatically generate efficient arithmetic circuits for high-level circuits in the context of homomorphic encryption, where this is currently inefficient, or left as an exercise for the protocol designer. These techniques can also be used in the context of other secure computation techniques, such as arithmetic garbling.

References

[ACS20] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. "Faster Homomorphic Encryption is not Enough: Improved Heuristic for Multiplicative Depth Minimization of Boolean Circuits". In: *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*. Ed. by Stanislaw Jarecki. Vol. 12006. Lecture Notes in Computer Science. Springer, 2020, pp. 345–363. DOI: 10.1007/978-3-030-40186-3_15. URL: https://doi.org/10.1007/978-3-030-40186-3%5C_15.

[AG23] Muhammad Abbas and Oscar Gustafsson. "Integer Linear Programming Modeling of Addition Sequences With Additional Constraints for Evaluation of Power Terms". In: *CoRR* abs/2306.15002 (2023). DOI: 10.48550/ARXIV.2306.15002. arXiv: 2306.15002. URL: https://doi.org/10.48550/arXiv.2306.15002.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. "How to Garble Arithmetic Circuits". In: *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. Ed. by Rafail Ostrovsky. IEEE Computer Society, 2011, pp. 120–129. DOI: 10.1109/FOCS.2011.40. URL: https://doi.org/10.1109/FOCS.2011.40.

[Arc+19] David W. Archer et al. "RAMPARTS: A Programmer-Friendly System for Building Homomorphic Encryption Applications". In: *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2019, London, UK, November 11-15, 2019*. Ed. by Michael Brenner, Tancrède Lepoint, and Kurt Rohloff. ACM, 2019, pp. 57–68. DOI: 10.1145/3338469.3358945. URL: https://doi.org/10.1145/3338469.3358945.

[Ber+89] François Bergeron et al. "Addition Chains Using Continued Fractions". In: *J. Algorithms* 10.3 (1989), pp. 403–412. DOI: 10.1016/0196-6774(89)90036-9. URL: https://doi.org/10.1016/0196-6774(89)90036-9.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "Fully Homomorphic Encryption without Bootstrapping". In: *Electron. Colloquium Comput. Complex.* TR11-111 (2011). ECCC: TR11-111. URL: https://eccc.weizmann.ac.il/report/2011/111.

[BI20] Charlotte Bonte and Ilia Iliashenko. "Homomorphic String Search with Constant Multiplicative Depth". In: *CCSW'20, Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop, Virtual Event, USA, November 9, 2020*. Ed. by Yinqian Zhang and Radu Sion. ACM, 2020, pp. 105–117. DOI: 10.1145/3411495.3421361. URL: https://doi.org/10.1145/3411495.3421361.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_50.

[Car+16] Sergiu Carpov et al. "Practical Privacy-Preserving Medical Diagnosis Using Homomorphic Encryption". In: *9th IEEE International Conference on Cloud Computing, CLOUD 2016, San Francisco, CA, USA, June 27 - July 2, 2016*. IEEE Computer Society, 2016, pp. 593–599. DOI: 10.1109/CLOUD.2016.0084. URL: https://doi.org/10.1109/CLOUD.2016.0084.

[CAS17] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey. "A Multi-start Heuristic for Multiplicative Depth Minimization of Boolean Circuits". In: *Combinatorial Algorithms - 28th International Workshop, IWOCA 2017, Newcastle, NSW, Australia, July 17-21, 2017, Revised Selected Papers*. Ed. by Ljiljana Brankovic, Joe Ryan, and William F. Smyth. Vol. 10765. Lecture Notes in Computer Science. Springer, 2017, pp. 275–286. DOI: 10.1007/978-3-319-78825-8_23. URL: https://doi.org/10.1007/978-3-319-78825-8%5C_23.

[Chi+20] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *J. Cryptol.* 33.1 (2020), pp. 34–91. DOI: 10.1007/S00145-019-09319-X. URL: https://doi.org/10.1007/s00145-019-09319-x.

[Cho+21] Sangeeta Chowdhary et al. "EVA Improved: Compiler and Extension Library for CKKS". In: *WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021*. WAHC@ACM, 2021, pp. 43–55. DOI: 10.1145/3474366.3486929. URL: https://doi.org/10.1145/3474366.3486929.

[Cow+21] Meghan Cowan et al. "Porcupine: a synthesizing compiler for vectorized homomorphic encryption". In: *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*. Ed. by Stephen N. Freund and Eran Yahav. ACM, 2021, pp. 375–389. DOI: 10.1145/3453483.3454050. URL: https://doi.org/10.1145/3453483.3454050.

[Deg+24] Jean Paul Degabriele et al. “SoK: Efficient Design and Implementation of Polynomial Hash Functions over Prime Fields”. In: *45th IEEE Symposium on Security and Privacy (SP 2024)*. 2024.

[DLS81] Peter J. Downey, Benton L. Leong, and Ravi Sethi. “Computing Sequences with Addition Chains”. In: *SIAM J. Comput.* 10.3 (1981), pp. 638–646. DOI: 10.1137/0210047. URL: https://doi.org/10.1137/0210047.

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

[GMT23] Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. “SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks”. In: *Proceedings on Privacy Enhancing Technologies* 2023.3 (July 2023), pp. 154–172. DOI: 10.56553/popets-2023-0075.

[HS14] Shai Halevi and Victor Shoup. “Algorithms in HElib”. In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Springer, 2014, pp. 554–571. DOI: 10.1007/978-3-662-44371-2_31. URL: https://doi.org/10.1007/978-3-662-44371-2%5C_31.

[HVM04] Darrel Hankerson, Scott Vanstone, and Alfred Menezes. *Guide to Elliptic Curve Cryptography*. 1st ed. Springer Professional Computing. Springer Science+Business Media New York 2004. Springer New York, NY, 2004, pp. XX, 312. ISBN: 978-0-387-95273-4. DOI: 10.1007/b97644.

[IMM18] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. “PySAT: A Python Toolkit for Prototyping with SAT Oracles”. In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8%5C_26.

[INZ21] Ilia Iliashenko, Christophe Nègre, and Vincent Zucca. “Integer Functions Suitable for Homomorphic Encryption over Finite Fields”. In: *WAHC ’21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021*. WAHC@ACM, 2021, pp. 1–10. DOI: 10.1145/3474366.3486925. URL: https://doi.org/10.1145/3474366.3486925.

[IZ21] Ilia Iliashenko and Vincent Zucca. “Faster homomorphic comparison operations for BGV and BFV”. In: *Proc. Priv. Enhancing Technol.* 2021.3 (2021), pp. 246–264. DOI: 10.2478/POPETS-2021-0046. URL: https://doi.org/10.2478/popets-2021-0046.

[Lee+20] DongKwon Lee et al. “Optimizing homomorphic evaluation circuits by program synthesis and term rewriting”. In: *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*. Ed. by Alastair F. Donaldson and Emina Torlak. ACM, 2020, pp. 503–518. DOI: 10.1145/3385412.3385996. URL: https://doi.org/10.1145/3385412.3385996.

[McL21] Michael Ben McLoughlin. *addchain: Cryptographic Addition Chain Generation in Go*. Version 0.4.0. Oct. 2021. DOI: 10.5281/zenodo.5622943. URL: https://github.com/mmcloughlin/addchain.

[MDM14] António Morgado, Carmine Dodaro, and João Marques-Silva. “Core-Guided MaxSAT with Soft Cardinality Constraints”. In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by Barry O’Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 564–573. DOI: 10.1007/978-3-319-10428-7_41. URL: https://doi.org/10.1007/978-3-319-10428-7%5C_41.

[Mon+23] Johannes Mono et al. “Finding and Evaluating Parameters for BGV”. In: *Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings*. Ed. by Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne. Vol. 14064. Lecture Notes in Computer Science. Springer, 2023, pp. 370–394. DOI: 10.1007/978-3-031-37679-5_16. URL: https://doi.org/10.1007/978-3-031-37679-5%5C_16.

[PS73] Mike Paterson and Larry J. Stockmeyer. “On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials”. In: *SIAM J. Comput.* 2.1 (1973), pp. 60–66. DOI: 10.1137/0202007. URL: https://doi.org/10.1137/0202007.

[Sch75] Arnold Schönhage. “A Lower Bound for the Length of Addition Chains”. In: *Theor. Comput. Sci.* 1.1 (1975), pp. 1–12. DOI: 10.1016/0304-3975(75)90008-0. URL: https://doi.org/10.1016/0304-3975(75)90008-0.

[Sin05] Carsten Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*. Ed. by Peter van Beek. Vol. 3709. Lecture Notes in Computer Science. Springer, 2005, pp. 827–831. DOI: 10.1007/11564751_73. URL: https://doi.org/10.1007/11564751%5C_73.

[TC21] Edward G. Thurber and Neill Michael Clift. “Addition chains, vector chains, and efficient computation”. In: *Discret. Math.* 344.2 (2021), p. 112200. DOI: 10.1016/J.DISC.2020.112200. URL: https://doi.org/10.1016/j.disc.2020.112200.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. "SoK: Fully Homomorphic Encryption Compilers". In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1092–1108. DOI: 10.1109/SP40001.2021.00068. URL: https://doi.org/10.1109/SP40001.2021.00068.

[YM24] Mingfei Yu and Giovanni De Micheli. *Expediting Homomorphic Computation via Multiplicative Complexity-aware Multiplicative Depth Minimization*. Cryptology ePrint Archive, Paper 2024/1015. https://eprint.iacr.org/2024/1015. 2024. URL: https://eprint.iacr.org/2024/1015.

Chapter 6

Oraqle: A Depth-Aware Secure Computation Compiler

In the previous chapter, we introduced the concept of depth-aware arithmetization along with several algorithms for arithmetizing common primitives. While these algorithms solve impracticality 3: Arithmetization in theory, they do not allow non-experts to easily develop programs using homomorphic encryption in practice. To do so, one would still need to choose applicable parameters and generate code that can be executed using existing homomorphic encryption libraries that matches the generated arithmetic circuit.

In this chapter, we propose a compiler for this purpose. The compiler implements the depth-aware arithmetization, and automatically chooses parameters and generates code. Because the arithmetic circuits generated by the algorithm from Chapter 5 benefit from small plaintext spaces, we specifically use the HElib homomorphic encryption library as it supports small prime moduli. We use the compiler to show that the arithmetic circuits generated by the compiler can outperform Boolean circuits for circuit computational tasks, addressing impracticality 4: Compute-intensive.

This chapter is an adaptation of the work with the same title published in 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, authored by Jelle Vos, Mauro Conti, and Zekeriya Erkin.

6.1 Introduction

In the past decade, the field of somewhat and fully homomorphic encryption (FHE) has seen significant advancements, leading to the development of tens of homomorphic encryption compilers. These compilers are essential tools for enabling users who might otherwise not have the time or expertise to transition from plaintext computations to secure computation, and they do so with relative ease compared to other techniques like secret sharing.

Despite their utility, existing FHE compilers still have limitations. A critical shortcoming is their limited support for high-level primitives such as equality checks, comparisons, and AND and OR operations involving multiple operands. This is because FHE schemes compute circuits of additions and multiplications over some algebra. For the schemes that we consider in this work, this algebra is typically the commutative ring of integers modulo some q. In general, it is not straightforward to express high-level primitives as arithmetic circuits in this algebra. However, it is a misconception that generating these arithmetic circuits is

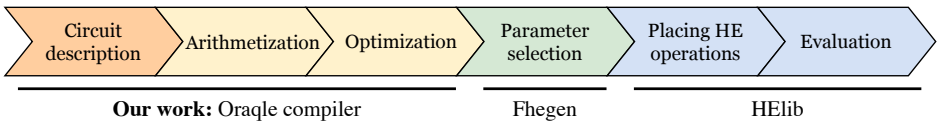


Figure 6.1: Typical pipeline in a homomorphic encryption compiler and the parts covered by our work.

impossible for every q. When q is a prime, the plaintext algebra is the finite field $\mathbb{F}_p$, in which any function can be expressed as an arithmetic circuit.

Expressing high-level operations as arithmetic circuits is a process called arithmetization. Some compilers do support the arithmetization of high-level primitives, but they only allow doing so for the plaintext algebra $\mathbb{F}_2$, thereby restricting the circuits to the Boolean circuits. This is a significant restriction that potentially ignores many more efficient circuits. For example, if we want to compute an AND operation between 16 operands, we require many more multiplications in Boolean circuits than in arithmetic circuits, where q can be larger. These multiplications are significantly more expensive to compute than additions. Concretely, we require 15 multiplications in $\mathbb{F}_2$, and only 4 in $\mathbb{F}_{17}$ (see Section 6.4).

There are also compilers that provide arithmetization of high-level primitives by relying on FHE schemes that support programmable bootstrapping. Instead of computing circuits consisting of additions and multiplications, these schemes compute circuits of additions and programmable bootstrapping operations, which are essentially lookup tables. A common example of such a scheme is TFHE [Chi+20]. While these schemes are typically computationally efficient, they require every evaluator to have large bootstrapping keys, which are in the order of tens to thousands of megabytes in size. Our work focuses on BFV/BGV-type schemes [Bra12; FV12; BGV12], which do not require the evaluator to have bootstrapping keys.

A second problem in arithmetization for FHE is that a circuit's efficiency is relies strongly on the multiplicative depth of an arithmetic circuit. This metric is defined as the highest number of multiplication on any path through the circuit. The reason is that FHE ciphertexts contain some noise as part of the underlying cryptographic hardness assumption. As FHE schemes perform more homomorphic operations on ciphertexts, this noise grows. At some point, the noise may become so large as to override the plaintext that the ciphertext originally encrypted. Since the noise grows most strongly during ciphertext multiplications, the multiplicative depth is a useful metric for measuring noise growth. Knowing the multiplicative depth of the circuit that will be computed allows one to choose parameters that are large enough to accommodate the expected amount of noise growth. However, larger parameters make multiplications more expensive to compute. As a result, we have two metrics that we want to minimize to increase the efficiency of arithmetic circuits: the number of multiplications, which are expensive to compute, and the multiplicative depth, which impacts the cost of individual multiplications.

In this work, we propose a new compiler called Oraqle that solves both the problem of arithmetizing high-level operations and the problem of multiplicative depth reduction simultaneously. The Oraqle compiler implements depth-aware

arithmetization, a concept recently introduced by Vos et al. [VCE24] (Chapter 5). By restricting the plaintext algebra to $\mathbb{F}_p$, where p is prime, we can arithmetize any high-level function. Unlike other compilers, the Oraqle compiler does not focus on reducing either the number of multiplications or the multiplicative depth, but it reduces both. In doing so, it returns multiple circuits that trade off these two metrics. To be precise, it generates a front of circuits that trade off the multiplicative depth and the multiplicative cost, which is a number that considers that squaring operations are cheaper to compute than arbitrary multiplications.

Our work is not the first to trade off the multiplicative depth with the number of multiplications. However, depth reduction has previously only been considered as an optimization stage that comes after arithmetization [CAS17; ACS20; Lee+20; YM24]. These works input any arithmetic circuit, so they cannot exploit the knowledge of the high-level operations that these circuits perform. Besides, it is possible to arithmetize a high-level operation in multiple ways, resulting in radically different arithmetic circuits. These techniques cannot recover all possible circuits generated using depth-aware arithmetization.

In Figure 6.1, we describe a typical pipeline for compiling high-level circuits into homomorphic encryption circuits. Notice that the Oraqle compiler considers depth reduction in the *arithmetization* stage, whereas other compilers consider depth reduction in the *optimization* stage. We note that our work only addresses the first parts of the pipeline, and we rely on other work for the later parts. We believe that this decoupling is a positive development. For example, a user (or compiler) could use our compiler to generate arithmetic circuits and another compiler to generate parameters and place homomorphic encryption operations. In the Oraqle compiler, we rely on fhegen [Mon+22] to select parameters and HElib [HS20] for the placement of relinearization and modswitch operations, as well as the final evaluation of the circuit using the BGV cryptosystem [BGV12].

In short, while there are already many homomorphic encryption compilers, the Oraqle compiler is unique in the sense that it:

- Arithmetizes any high-level operation in $\mathbb{F}_p$, supporting equality checks, comparisons, and AND and OR operations between many operands, among others.
- Minimizes both the number of multiplications and the multiplicative depth during arithmetization, generating multiple circuits that trade off these two metrics.
- Considers the fact that squaring is often cheaper to compute homomorphically than arbitrary ciphertext multiplications.

In this paper, we present the practical workings of the Oraqle compiler. We demonstrate that our compiler produces more efficient arithmetizations of comparison operations ($x < y$) than other work for circuits over $\mathbb{F}_p$ where p is prime. Next to that, we use our compiler to demonstrate that arithmetic circuits (i.e. where $p > 2$) can be more performant than Boolean circuits. We show this for doing an equality check ($x = y$) between two 64-bit inputs.

Our paper is structured as follows. We start by reviewing other general-purpose homomorphic encryption compilers for BFV/BGV-type schemes in Section 6.2.

After that, we proceed in the same order as the pipeline diagram in Figure 6.1. So, in Section 6.3 we explain how users can describe high-level circuits in Oraqle. In Section 6.4 we explain how we implement depth-aware arithmetization. We also explain some heuristics & approximations that can be used to speed up circuit generation time. Next, in Section 6.5, we describe how semantic common subexpression elimination can further optimize the generated arithmetic circuits. After that, we describe in Section 6.6 how we use *fhegen* and *HElib* to compile the circuits to an executable binary. We present some results in Section 6.7, and finish with an overview of limitations and a conclusion in Sections 6.8 & 6.9.

6.2 Homomorphic encryption compilers

We briefly discuss existing general-purpose homomorphic encryption compilers for $\mathbb{Z}_q$ plaintext spaces. We exclude works that are solely for TFHE, because these do not execute arithmetic circuits: instead, they are comprised of additions and programmable bootstrapping operations. We only consider works from 2020 and after. We refer the reader to the work by Viand et al. [VJH21] for prior works.

In Table 6.1, we provide an overview of the works described in this section. We specify several properties in the same order as the pipeline presented in Figure 6.1. Specifically, we consider each compiler's circuit description interface, by stating their input language and plaintext algebra. For arithmetization, we discuss whether they support high-level operations, and whether they consider the multiplicative depth during arithmetization. Moreover, we state whether the compilers implement common subexpression elimination (CSE) to reduce the multiplicative size, or depth reduction techniques. Finally, we state whether they automate parameter selection and the placement of relinearization and modswitch operations, as well as the library they use for evaluation.

A takeaway is that few of the previous compilers implement arithmetization for plaintext spaces with $p > 2$, so there is a large space of possible circuits they cannot generate. Moreover, none of the compilers generate circuits in a depth-aware manner for $p > 2$.

T2. The T2 cross-compiler [GMT23] provides a DSL for describing arithmetic circuits, allowing one to generate code for multiple libraries. If p is a prime, the compiler implements arithmetization for equality checks and comparisons, but it does not consider the depth-cost trade-off. The compiler chooses parameters from a predefined list, and places homomorphic encryption operations automatically.

HEIR. The HEIR compiler [Goo24] is based on the multi-level intermediate representation (MLIR) toolchain, which can be reused and extended by other compilers. For this reason, it supports multiple input formats and multiple FHE schemes. At the time of writing, the compiler only translates arithmetic operations on secrets to arithmetic operations on encrypted secrets, so it does not support high-level arithmetization but it can be extended as such. Due to its extensible nature, the compiler inherits common subexpression elimination and tree balancing from MLIR.

Porcupine. The Porcupine compiler [Cow+21] focuses on automatic vectorization of arithmetic circuits. It inputs circuits written in a new DSL. While it does not

Table 6.1: An overview of homomorphic encryption compilers since 2020 and their compilation stages (see Figure 6.1).

Compiler		**Circuit description**		**Arithmetization**		**Optimization**		**Parameter selection**	**Placing HE operations**	**Evaluation**
Name	Year	Input	Alg.	High-level	Depth-aware	CSE	Depth red.	Automatic	Automatic	Library
Porcupine [Cow+21]	2021	Quill	$\mathbb{Z}_q$	-	○	○	●	○	◐	SEAL
HEIR [Goo24]	2023	Multiple	$\mathbb{Z}_q$	-	○	●	◐	●	●	Multiple
HECO [Via+23]	2023	Python	$\mathbb{Z}_q$	-	○	●	●	◐	◐	SEAL
T2 [GMT23]	2023	C++	$\mathbb{Z}_q$	Eq. & comp.	○	○	○	◐	●	Multiple
HElium [Gün+23]	2023	HEDSL	-	○	○	○	◐	◐	◐	Multiple
Oraqle	**2024**	**Python**	$\mathbb{F}_p$	**Multiple**	●	●	◐	**fhegen** [Mon+22]	◐	**HElib**

arithmetize high-level operations, it performs depth reduction due to vectorization. Parameters must be selected manually and relinearization operations are placed naively.

HECO. Similar to the HEIR compiler, the HECO compiler [Via+23] relies on the MLIR toolchain. It supports a python front-end and a SEAL backend. It does not implement the arithmetization of high-level primitives, which is currently left to the user. It does perform simple parameter selection and naive placement of relinearization operations. Since it is based on MLIR, it can perform CSE, and it supports a vectorization pass that reduces the multiplicative depth of series of multiplications.

HElium. The HElium compiler is a compiler that focuses on proxy re-encryption, allowing computations on data stored under different keys. The main objective is reducing re-encryption operations. It implements depth reduction in the form of tree rebalancing. As an input language, it uses a new domain-specific language.

Our work: Oraqle. Our compiler implements depth-aware arithmetization and semantic CSE. The circuits can be described using Python functions. For parameter selection, we use fhegen, and for the placement of homomorphic encryption operations, we rely on the HElib library, which does so naively (i.e. it may scale down the modulus only to scale it up before the next operation).

6.3 Programming interface

The programming interface defines the way in which users supply input to the compiler. As shown in Table 6.1, several works provide a domain-specific language for the user to do so. While this allows one to tailor the language to the use

case, it introduces a learning curve. In the Oraqle compiler, we do not use a DSL, and instead allow the user to express circuits in pure Python, supporting a subset of Python functions by overloading operators. This means that we also do not perform introspection or analysis of the abstract syntax tree. While those approaches would allow one to be more expressive, they make it harder for users to change the behavior of the compiler to their needs. A downside is that in some cases, due to language restrictions, the user cannot use a built-in function and instead must resort to a function with a similar name. For example, instead of calling `sum`, the user must call `sum_`. In this section, we first describe how we go from the user's inputs in Python to a high-level circuit description. After that, we provide some examples of Python code and the circuits they describe.

The key way in which we construct high-level circuits that the compiler can arithmetize, is to symbolically execute the Python code by overriding the typical operators. For example, when the user calls `x - y`, this will result in a symbolic `Subtraction(x, y, gf)` node, rather than the interpreter trying to evaluate the expression. We provide several ways in which these symbolic nodes are combined to create different symbolic nodes. For example, additions are automatically flattened into one large `Sum` node. Moreover, if all the inputs to an operation are constants, then the constant is folded. In other words, the output is a constant too.

We capture the semantics of different high-level operations by specifying different types of operations, such as:

- **Fixed nodes**, which have a fixed number of operands.
  - *Commutative binary nodes:* E.g. addition and equality checks.
  - *Non-commutative binary nodes:* E.g. comparisons.
  - *Univariate nodes:* E.g. exponentiation by a constant.
  - *Leaf nodes:* E.g. inputs and constants.
- **Flexible nodes**, which have an arbitrary number of operands.
  - *Commutative & associative reducible nodes with a set of operands:* E.g. AND and OR operations.
  - *Commutative & associative reducible nodes with a multiset of operands:* E.g. sums and products.

In the compiler, we ensure that, for common operations, there is only one way to represent them. For example, we do not allow an AND operation with one operand, or an addition between two constants (this should simply be a constant). As a result, the only time that a `Constant` node exists, is when the entire circuit evaluates to a constant. Otherwise, the constant is part of an operation such as a `ConstantAddition`.

Next, we showcase several examples of the conversions from Python expressions to high-level circuits. These figures are generated by the compiler, which outputs DOT files. Note that these high-level circuits are not yet arithmetized; they describe the function that the user wants to perform, split into common primitives.

Describing high-level circuits. We start with a simple example of a program that a user might run. A user might wish to compute $[x < y]\texttt{AND}[y == z]$. The

Oraqle compiler requires the user to first specify the plaintext algebra, prior to the defining the input node x, y, and z. In Listing 1 (see Appendix 6.A), we use $\mathbb{F}_{31}$ as the plaintext algebra. As one can see, after defining the inputs, the operations are expressed in the same way as in regular Python functions. Finally, the user creates a `Circuit`, which contains an arbitrary number of outputs.
In Figure 6.2, we show the high-level circuit as generated by the compiler. For non-commutative nodes, the edges enter the node at the correct side, indicating the direction of the operation (from left to right). In commutative nodes, the edges enter the node anywhere.

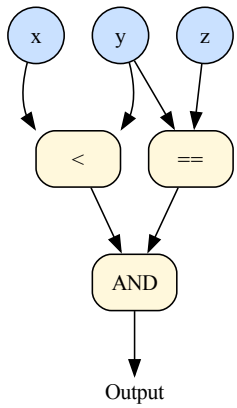


Figure 6.2: An example circuit with high-level operations.

Describing arithmetization in the compiler. While the Python interface is useful for users to express the functions they want to compute, it is also used within the compiler to implement transformations such as arithmetization. For example, if the scheme does not support subtractions, the compiler implements a way to arithmetize subtractions into an addition and constant multiplication as `x + -1 * y`. Subtractions can in turn be used to arithmetize if-else operations.

Extending arithmetization in the compiler. We also use the Python interface to implement arithmetization external to the compiler, making it easy to compare with other works, such as the comparison circuits as proposed by Gouert et al. [GMT23] for the T2 compiler. The code for this is almost as simple as the equation used to describe the arithmetization. We present this code in Listing 2 (see Appendix 6.A).

The high-level circuit that the compiler generates from this code can be seen in Figure 6.3. While it may seem that the compiler implements loop unrolling, this is not the case. The for-loop is executed as is. Since the compiler flattens sums, there is only one addition node at the end of the circuit. Operations like exponentiation by 6 will be arithmetized later, instead of turning them into multiplications at this stage. The reason is that exponentiation can be arithmetized in different ways, trading off the multiplicative cost and depth, as described in Chapter 5.

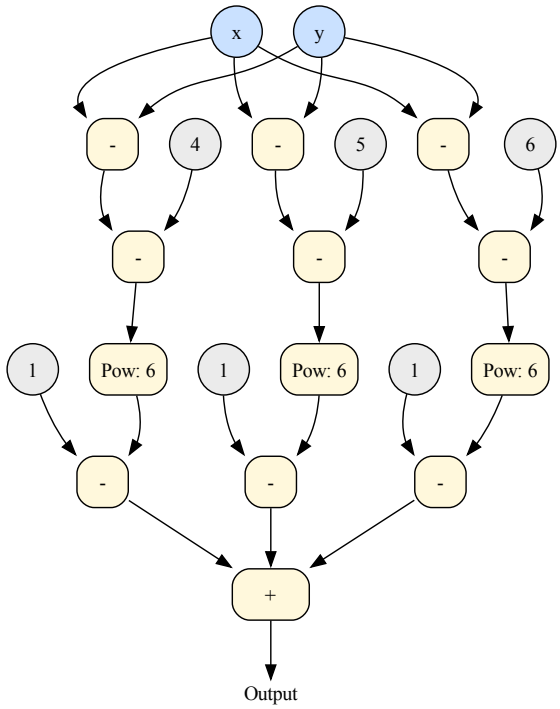


Figure 6.3: High-level circuit for $x < y$ as proposed by Gouert et al. [GMT23] when $p = 7$.

6.4 Depth-aware arithmetization

The Oraqle compiler implements the depth-aware arithmetization techniques described by Vos et al. [VCE24]. Specifically, they propose how to arithmetize distinct products, exponentiations, polynomial evaluations, and AND and OR operations between multiple operands in a way that trades off the multiplicative cost and the multiplicative depth. In this section, we do not discuss the theory behind the techniques, but we discuss our practical implementation. We begin by explaining our implementation and providing some examples, after which we describe several ways in which the time it takes to generate circuits can be reduced.

6.4.1 Arithmetization for $p \geq 2$

In the introduction, we gave an example of how AND operations between multiple operands can be performed with fewer multiplications in an arithmetic circuit over $\mathbb{F}_p$ with $p > 2$, than in a Boolean circuit where $p = 2$. The reason is that a Boolean circuit requires the operation $x_1 \wedge \ldots x_{16}$ to be arithmetized as a product $x_1 \times \cdots \times x_{16}$, whereas an arithmetic circuit over $\mathbb{F}_{17}$ allows for many different kinds

of circuits. The Oraqle compiler arithmetizes this operation as $(x_1 + \cdots + x_{16})^{16}$, in which the exponentiation only requires four multiplications.

While the Oraqle compiler implements depth-aware arithmetization, it also implements 'regular' arithmetization, in which the compiler only outputs a single circuits. In this mode the compiler seeks to minimize the multiplicative cost, and the multiplicative depth secondarily. The compiler is significantly faster at performing arithmetization in this way, because composition is straightforward: the output of arithmetization of high-level operations is a single arithmetic circuit rather than a Pareto front.

6.4.2 Depth-aware arithmetization for $p \geq 2$

When it comes to depth-aware arithmetization, the compiler outputs multiple arithmetic circuits for each high-level circuit if it can find a trade-off between the multiplicative cost and the multiplicative depth. We provide an example for performing equality checks in $\mathbb{F}_{31}$, which is the smallest plaintext modulus for which a trade-off occurs. Listing 3 (see Appendix 6.A) shows the Python input for this function. The Oraqle compiler allows the user to specify the cost of a squaring operation relative to a ciphertext multiplication, which we denote by σ. Calling `arithmetize_depth_aware()` defaults to $\sigma = 1.0$.

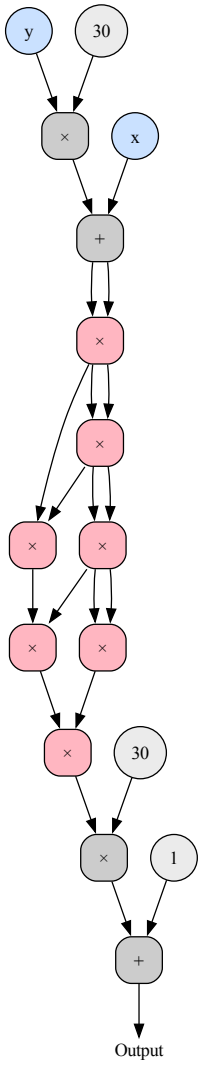
The compiler internally makes several calls to the MaxSAT compiler to generate multiple arithmetic circuits with different multiplicative depth. The results are in Figure 6.4. Here, red-colored multiplications denote non-constant multiplications, which are expensive to compute. The compiler here generates one circuit with a multiplicative depth of 5, and a multiplicative cost of 7, and another with depth 6 and cost 6. It is not clear which circuit is more efficient, especially under composition, until we evaluate them. One limitation is that, in the current version of the compiler, we only implement depth-aware arithmetization for circuits with a single output.

6.4.3 Practical optimizations

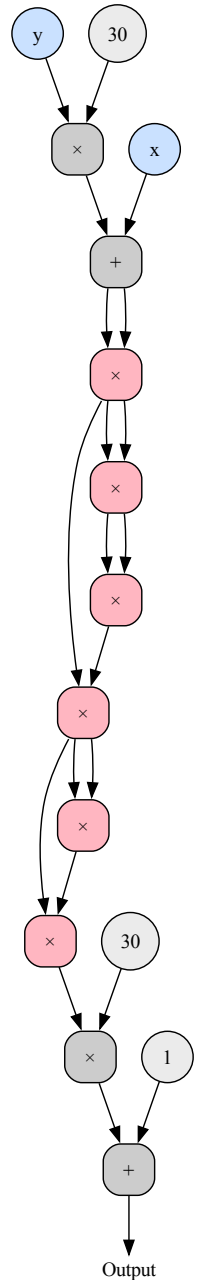
We discuss three optimizations that reduce the times it takes to generate these circuits without changing them.

The current bottleneck in our implementation is the MaxSAT solver that we use to arithmetize exponentiation circuits. Our implementation uses the RC2 solver [MDM14] implemented in PySAT [IMM18], and defaults to the Glucose 4.2.1 SAT solver [AS24]. While it is not always possible to reduce the number of exponentiations that we must arithmetize, we employ caching to significantly reduce the number of calls made to the MaxSAT solver.

Another optimization is that constant folding allows us to sometimes skip arithmetization altogether. For example, if the a subcircuit in a `Product` node evaluates to 0, we can output a constant. The same applies to other nodes with multiple operands.



(a) Depth-5 circuit



(b) Depth-6 circuit

Figure 6.4: Depth-aware arithmetization of $x = y$ in $\mathbb{F}_{31}$.

Finally, *commutative & associative reducible nodes with a set of operands* (see Section 6.3), the inputs are actually modeled as a set. This means that if during arithmetization, an AND operation receives the same operand twice (or one that is equivalent), it ignores the second. We discuss equivalence in the context of semantic subexpression elimination in Section 6.5.

6.4.4 Heuristics & approximations

We propose several ways in which the user may speed up circuit generation time at the cost of a potentially worse circuit. This is necessary for larger circuits and larger plaintext moduli, which increase the duration of arithmetization. The reason is that the depth-aware arithmetization techniques proposed by Vos et al. [VCE24] in some cases perform exhaustive searches.

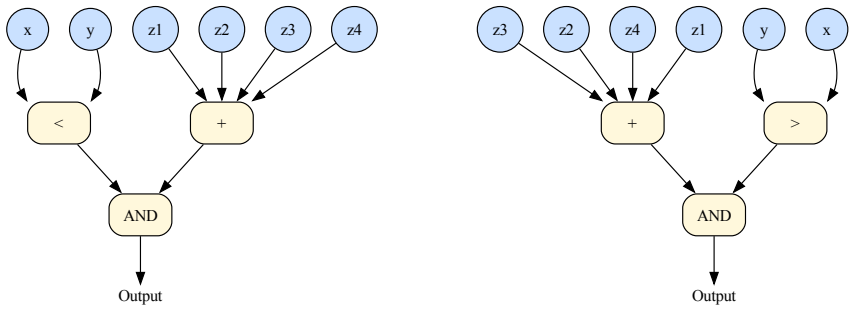
One context in which we can avoid exhaustive search, is in polynomial evaluation. All the polynomial evaluation methods described in [VCE24] require the compiler to choose a parameter k, which affects both the multiplicative depth and cost of the circuits that are generated. Vos et al. propose to try all values $1 \leq k < p$ but this requires a significant amount of computation and many calls to the MaxSAT solver. Paterson & Stockmeyer [PS73] instead analytically derive a single value for k, but Vos et al. show that this is not always optimal in practice. We propose a heuristic, where we only evaluate several values for k around the value derived by Paterson & Stockmeyer. In our experiments, we find that the optimal k is typically only 1 value away. The compiler only searches for values up to twice the size of the analytically-optimal k, which makes circuit generation approximately twice as fast in practice.

6.5 Optimization of arithmetic circuits

After generating arithmetic circuits, different compilers perform different forms of post-processing in an attempt to reduce the multiplicative cost or the multiplicative depth. These are transformations from arithmetic circuits to other arithmetic circuits. The Oraqle compiler currently implements one optimization in the form of semantic common subexpression elimination, which is similar to that implemented by the EVA compiler [Cho+21] for a different kind of homomorphic cryptosystem.

Common subexpression elimination is a technique that has been applied to many homomorphic encryption compilers and many regular compilers alike. The simple idea is to never compute the same thing twice. While this seems obvious, it is not uncommon for arithmetization (or compilation) to introduce common subexpressions. In our work, we implement semantic common subexpression elimination, meaning that the compiler can also recognize two subexpressions to be equivalent but not identical. We give an example in Figures 6.5a & 6.5b of two circuits that the compiler can tell to be equivalent.

The way we implement these equivalence checks efficiently, is to ensure that equivalent expressions have the same hash. This is not always possible, but it is easy to do for properties such as commutativity. The Oraqle compiler computes the hash for commutative & associative nodes by sorting the hashes of all the



(a) Original circuit

(b) Equivalent circuit

Figure 6.5: Two circuits that are not identical but equivalent.

operands before computing the hash, making the order of operands irrelevant. It can also be done for high-level operations that are non-commutative but each others inverses. For example, $x < y$ is equivalent to $y > x$.

6.6 Code generation

Since the Oraqle compiler currently focuses on arithmetization, it does not perform parameter selection, placement of homomorphic encryption operations, or evaluation. Instead, it relies on fhegen [Mon+22] and HElib [HS20]. In this section, we describe the steps from an arithmetic circuit to an executable binary chronologically. We note that one might also use other tools to finish compiling the arithmetic circuits generated by the Oraqle compiler.

Register allocation. To reduce the memory footprint of the final binary, we implement a register allocation step, which determines at any time throughout the computation how many ciphertexts must be stored. We note that these are logical registers containing FHE ciphertexts, and not actual hardware registers. We do so using a topological graph traversal of the arithmetic circuits. After this step, each node in the arithmetic circuit knows to which register it can assign the result of its computation.

Translation into instructions. At this point, we can compile the arithmetic circuit with register allocation to what is essentially assembly for FHE operations. That is, input nodes are translated to instructions that place a named input into a register, arithmetic nodes perform operations on several registers, placing the result in a (possibly overlapping) register, and output nodes instruct are translated to instructions that output a given register. Importantly, the compiler traverses the graph in the same way that it did before.

Translation to a program. Finally, we generate C++ code that can be compiled into an actual binary. This takes two steps, as the code includes both the parameters with which the FHE schemes will be instantiated, as well as the actual circuit evaluation.

We use the methods provided by fhegen [Mon+22] to generate parameters for the default settings in HElib using the OpenFHE cost model. To do so, we derive several metrics from the arithmetic circuit. We make one modification to support $p = 2$, which is to decrement the polynomial degree m by 1, as p may not divide m. After this step, code generation is a direct translation from the FHE instructions to the functions implemented in HElib for performing homomorphic operations. The resulting code can be compiled using any C++ compiler. HElib here performs two stages of the pipeline as presented in Figure 6.1: it places and performs relinearization and modswitch operations, and it evaluates the homomorphic operations.

6.7 Results

In this section, we provide two results using the Oraqle compiler.[1] We first show that choosing $p > 2$ does lead to circuits with better practical performance than fixing $p = 2$. Next, we show that an optimistic implementation of the arithmetization technique used in the T2 compiler produces circuits with worse practical performance than the Oraqle compiler does. We execute our experiments on a strong computer with a Threadripper 7970X CPU. The CPU has 64 threads, but we only use a single thread to compile and evaluate the circuits. When it comes to memory, it has 4x64GB DDR5 RAM.

Arithmetic versus Boolean circuits Since the Oraqle compiler allows compiling any function into an arithmetic circuit for any plaintext modulus p that is prime, we can evaluate the performance of different p for the same operation. We consider here the function that checks whether two 64-bit inputs are equal and provide the results in Table 6.2 and we set $\sigma = 0.75$. Choosing $p > 2$ allows circuits with lower multiplicative cost at the expense of a higher depth. Moreover, $p = 2$ does not allow the ring dimension m to be a power of two. This choice of m means that homomorphic operations are slower, even though m can be smaller. We see that choosing $p = 5$ leads to an arithmetic circuit that is almost twice as fast to compute as the Boolean circuit.

Table 6.2: Run time for a circuit checking whether two 64-bit integers are equal. We consider the front of solutions across all $2 \leq p \leq 257$ that are prime.

Circuits			**Parameters**				**Results**
Modulus p	Depth	Cost	Ring dimension m	r	Bits	c	Run time (s)
2	6	63	16385	1	142	1	3.28
5	7	58	32768	1	178	1	1.67
17	8	51	32768	1	217	1	1.96

[1] Our compiler is available at DOI `10.4121/e8332e69-994b-4ea7-aff9-2bb73fd2e5fe` or on GitHub

Arithmetization in other compilers While there are many homomorphic encryption compilers, they typically target the earlier or later stages of the compilation pipeline. To still facilitate a comparison, we compare the less-than circuits generated by our compiler with those generated by the T2 compiler as described in the paper by Gouert et al. [GMT23]. While the techniques described by Vos et al. [VCE24] and Gouert et al. only perform comparisons between half of the elements in $\mathbb{F}_p$, we propose a new arithmetization that performs three calls to the half-comparisons. We denote these half-comparisons by $\prec$. We precompute $x_{\text{small}} = [x \prec \frac{p-1}{2}]$ and $y_{\text{small}} = [y \prec \frac{p-1}{2}]$. Our arithmetization for both works is as follows, where $\overline{x}$ represents negation of a Boolean variable:

$$[x < y] = \overline{x_{\text{small}} \oplus y_{\text{small}}}[x \prec y] + \left(x_{\text{small}} \wedge y_{\text{small}}\right). \tag{6.1}$$

In Figure 6.6 we compare the actual run time of the T2 circuits with the circuits generated by the Oraqle compiler for a growing plaintext modulus. Our circuits consistently outperform the T2 circuits, often by an order of magnitude, even though these sometimes have a lower multiplicative depth. The reason is that these circuits have a significantly higher multiplicative cost.

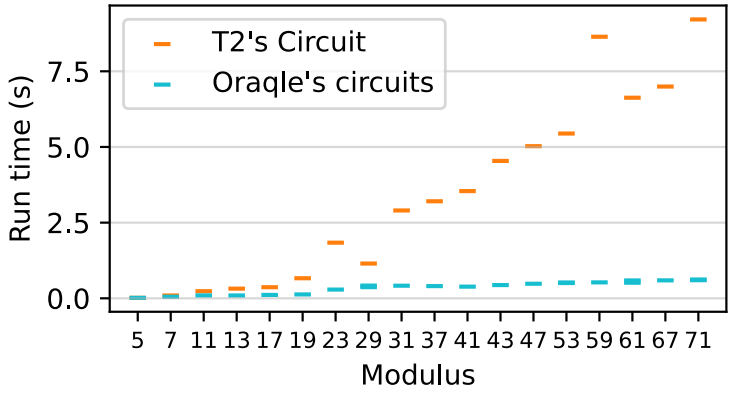


Figure 6.6: Arithmetization of a less-than operation in T2 and in Oraqle for $\sigma = 1.0$. In some cases the Oraqle compiler outputs multiple circuits but they perform similarly.

6.8 Limitations

Packing and rotations would allow for the number of multiplications to be reduced. Since we do not consider packing, we consider plaintext moduli that are not necessarily NTT-friendly, which would allow constant additions and multiplications with arbitrary vectors. This is also the reason why we currently only support code generation for HElib: other libraries do not support plaintext moduli that are not NTT-friendly.

The compiler does not yet take common inputs into account. E.g. when performing multiple polynomial evaluations, we could reuse the precomputations. The same applies to computing multiple products with the same operands.

The compiler also does not provide a layer of abstraction for integers (or e.g. real numbers) that exceed the plaintext space. In this stage, the user would have to implement this logic by hand.

We currently only perform depth-aware arithmetization on circuits with a single output. We argue that this is mostly a practical limitation and not a theoretical limitation.

As seen from our experiments, the cost of FHE cannot be completely described by the multiplicative depth and cost. These merely serve as metrics. There are other factors, such as the polynomial degree m being a power or two of not. Moreover, it matters how and when homomorphic encryption 'maintenance' operations are placed such as relinearizations and modswitches. Some circuits are more amenable to reducing the number of maintenance operations than others. Another example is that multiplications at the end of the circuit become slightly cheaper to compute as the ciphertext modulus shrinks.

6.9 Conclusion

In conclusion, while previous homomorphic encryption compilers play a crucial role in enabling users to transition from plaintext to secure computations, they typically only implement automatic arithmetization for Boolean circuits, or they require large bootstrapping keys. The Oraqle compiler addresses these issues by implementing depth-aware arithmetization for BFV/BGV-type cryptosystems with prime plaintext moduli, allowing it to express high-level primitives as arithmetic operations suitable for homomorphic encryption libraries. This allows one to find more efficient circuits than when considering depth reduction only as an afterthought.

Future work might focus on the following enhancements:

- Incorporating SIMD, which will require handling larger plaintext modulus for arbitrary plaintext vectors.
- Implementing multi-threading, accelerating compilation.
- Optimizing addition chain generation.
- Integrating sorting networks and other complex structures.
- Implementing early stopping, e.g. using a maximum depth.
- Extending the compiler's capability to handle nodes with an arbitrary number of outputs.

References

[ACS20] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. "Faster Homomorphic Encryption is not Enough: Improved Heuristic for Multiplicative Depth Minimization of Boolean Circuits". In: *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings*. Ed. by Stanislaw Jarecki. Vol. 12006. Lecture Notes in Computer Science. Springer, 2020, pp. 345–363. DOI: 10.1007/978-3-030-40186-3_15. URL: https://doi.org/10.1007/978-3-030-40186-3%5C_15.

[AS24] Gilles Audemard and Laurent Simon. *Glucose SAT Solver*. https://github.com/audemard/glucose. Accessed: 2024-07-27. 2024.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "(Leveled) fully homomorphic encryption without bootstrapping". In: *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*. Ed. by Shafi Goldwasser. ACM, 2012, pp. 309–325. DOI: 10.1145/2090236.2090262. URL: https://doi.org/10.1145/2090236.2090262.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_50.

[CAS17] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey. "A Multi-start Heuristic for Multiplicative Depth Minimization of Boolean Circuits". In: *Combinatorial Algorithms - 28th International Workshop, IWOCA 2017, Newcastle, NSW, Australia, July 17-21, 2017, Revised Selected Papers*. Ed. by Ljiljana Brankovic, Joe Ryan, and William F. Smyth. Vol. 10765. Lecture Notes in Computer Science. Springer, 2017, pp. 275–286. DOI: 10.1007/978-3-319-78825-8_23. URL: https://doi.org/10.1007/978-3-319-78825-8%5C_23.

[Chi+20] Ilaria Chillotti et al. "TFHE: Fast Fully Homomorphic Encryption Over the Torus". In: *J. Cryptol.* 33.1 (2020), pp. 34–91. DOI: 10.1007/S00145-019-09319-X. URL: https://doi.org/10.1007/s00145-019-09319-x.

[Cho+21] Sangeeta Chowdhary et al. "EVA Improved: Compiler and Extension Library for CKKS". In: *WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021*. WAHC@ACM, 2021, pp. 43–55. DOI: 10.1145/3474366.3486929. URL: https://doi.org/10.1145/3474366.3486929.

[Cow+21] Meghan Cowan et al. “Porcupine: a synthesizing compiler for vectorized homomorphic encryption”. In: *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*. Ed. by Stephen N. Freund and Eran Yahav. ACM, 2021, pp. 375–389. DOI: 10.1145/3453483.3454050. URL: https://doi.org/10.1145/3453483.3454050.

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

[GMT23] Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. “SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks”. In: *Proc. Priv. Enhancing Technol.* 2023.3 (2023), pp. 154–172. DOI: 10.56553/POPETS-2023-0075. URL: https://doi.org/10.56553/popets-2023-0075.

[Goo24] Google. *HEIR: A compiler for homomorphic encryption*. https://github.com/google/heir. Accessed: 2024-07-25. 2024.

[Gün+23] Mirko Günther et al. “HElium: A Language and Compiler for Fully Homomorphic Encryption with Support for Proxy Re-Encryption”. In: *CoRR* abs/2312.14250 (2023). DOI: 10.48550/ARXIV.2312.14250. arXiv: 2312.14250. URL: https://doi.org/10.48550/arXiv.2312.14250.

[HS20] Shai Halevi and Victor Shoup. “Design and implementation of HElib: a homomorphic encryption library”. In: *IACR Cryptol. ePrint Arch.* (2020), p. 1481. URL: https://eprint.iacr.org/2020/1481.

[IMM18] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. “PySAT: A Python Toolkit for Prototyping with SAT Oracles”. In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8%5C_26.

[Lee+20] DongKwon Lee et al. “Optimizing homomorphic evaluation circuits by program synthesis and term rewriting”. In: *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*. Ed. by Alastair F. Donaldson and Emina Torlak. ACM, 2020, pp. 503–518. DOI: 10.1145/3385412.3385996. URL: https://doi.org/10.1145/3385412.3385996.

[MDM14] António Morgado, Carmine Dodaro, and João Marques-Silva. “Core-Guided MaxSAT with Soft Cardinality Constraints”. In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by Barry O'Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 564–573. DOI: 10.1007/978-3-319-10428-7_41. URL: https://doi.org/10.1007/978-3-319-10428-7%5C_41.

[Mon+22] Johannes Mono et al. *Finding and Evaluating Parameters for BGV*. Cryptology ePrint Archive, Paper 2022/706. https://eprint.iacr.org/2022/706. 2022. URL: https://eprint.iacr.org/2022/706.

[PS73] Mike Paterson and Larry J. Stockmeyer. "On the Number of Nonscalar Multiplications Necessary to Evaluate Polynomials". In: *SIAM J. Comput.* 2.1 (1973), pp. 60–66. DOI: 10.1137/0202007. URL: https://doi.org/10.1137/0202007.

[VCE24] Jelle Vos, Mauro Conti, and Zekeriya Erkin. *Depth-Aware Arithmetization of Common Primitives in Prime Fields*. Cryptology ePrint Archive. 2024. URL: https://eprint.iacr.org/2024/1200.

[Via+23] Alexander Viand et al. "HECO: Fully Homomorphic Encryption Compiler". In: *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. Ed. by Joseph A. Calandrino and Carmela Troncoso. USENIX Association, 2023, pp. 4715–4732. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/viand.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. "SoK: Fully Homomorphic Encryption Compilers". In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1092–1108. DOI: 10.1109/SP40001.2021.00068. URL: https://doi.org/10.1109/SP40001.2021.00068.

[YM24] Mingfei Yu and Giovanni De Micheli. *Expediting Homomorphic Computation via Multiplicative Complexity-aware Multiplicative Depth Minimization*. Cryptology ePrint Archive, Paper 2024/1015. https://eprint.iacr.org/2024/1015. 2024. URL: https://eprint.iacr.org/2024/1015.

6.A Code samples

We present several code samples used to generate these circuits.

Listing 1 A simple example of a three-input function using high-level operations.

```
gf = GF(31)
x = Input("x", gf)
y = Input("y", gf)
z = Input("z", gf)

comparison = x < y
equality = y == z
both = comparison & equality

circuit = Circuit([both])
```

Listing 2 Implementation of a comparison operation as proposed by Gouert et al. [GMT23] in the Oraqle compiler.

```
gf = GF(p)
x = Input("x", gf)
y = Input("y", gf)

comparison = 0

for a in range((p + 1) // 2, p):
    comparison += 1 - (x - y - a) ** (p - 1)

circuit = Circuit([comparison])
```

Listing 3 Implementation of an equality operation over $\mathbb{F}_{31}$.

```
gf = GF(31)
x = Input("x", gf)
y = Input("y", gf)

equality = x == y

circuit = Circuit([equality])
arithmetic_circuits = circuit.arithmetize_depth_aware()
```

Chapter 7

Efficient Circuits for Permuting and Mapping Packed Values Across Leveled Homomorphic Ciphertexts

The oraqle compiler makes deploying homomorphic encryption easier and more efficient by implementing a full pipeline for generating efficient homomorphic encryption programs from high-level circuits. By integrating the arithmetization techniques described in Chapter 5, it allows exploiting the algebra of prime fields $\mathbb{F}_p$. Homomorphic encryption schemes like BFV and BGV then allow one to perform multiple evaluations of the homomorphic encryption circuit in parallel by packing multiple such elements as 'slots' of a polynomial in $\mathbb{F}_p[X]/(X^N + 1)$.

One limitation of the compiler is that the plaintext algebra $\mathbb{F}_p$ implies that the circuits generated by the compiler only focus on values within a single slot; there is no movement between slots. However, in some cases, one might want to move elements from one slot to another, to enable other computations. In this chapter, we propose algorithms for automatically generating efficient homomorphic encryption circuits capable of performing such permutations and maps, even across multiple BFV or BGV ciphertexts.

This chapter is an adaptation of the work with the same title published in European Symposium on Research in Computer Security 2022, authored by Jelle Vos, Daniël Vos, and Zekeriya Erkin. We would like to thank Neil Yorke-Smith for his great help with our optimization algorithm.

7.1 Introduction

Nowadays, organizations use cloud providers to outsource their data processing, easing deployment and allowing them to scale the architecture up and down when required [Arm+10]. While these organizations typically keep sensitive data in encrypted form at rest, they decrypt it when performing computations. Consequently, these organizations must fully trust the cloud providers, who can observe all sensitive data. To protect sensitive data while processing, researchers propose secure outsourced data processing solutions, which allow cloud providers to offer their services on data that they cannot see. In the settings of those proposals, organizations assume that the cloud provider performs the operations they ask them to, thus reducing privacy risks.

One possible approach that enables cloud providers to process sensitive data relies on fully homomorphic encryption (FHE) schemes. FHE allows anyone with

the correct public key to perform computations on encrypted data without seeing it. In current schemes, one typically encrypts integers or real numbers, which can be manipulated through addition and multiplication. A subset of FHE schemes (such as BFV [Bra12; FV12], BGV [BGV11], and CKKS [Che+17]) allows one to encrypt entire fixed-length vectors of integers or real numbers in one ciphertext through ciphertext packing. A limited number of additions and multiplications can be performed as element-wise operations between encrypted vectors, following the concept of single-instruction multiple-data (SIMD). As a result, operating on packed ciphertexts leads to significant speed-ups when there is a large set of data to be processed.

A problem arises when the data stored in two encrypted vectors do not align. For example, consider two ciphertexts that each hold a database relating to the incomes of a set of employees. One ciphertext holds their salary sorted by their first name, while another holds their yearly bonus sorted by their last name. An outsourced HR system might compute each employee's total income by adding the two together. However, directly adding the two ciphertexts together leads to a meaningless result. Instead, the HR system must align the data stored within one ciphertext with the other by permuting it.

FHE schemes that support ciphertext packing implement ciphertext rotations to allow one to align encrypted vectors. This primitive shifts the encrypted vector x places towards the end while cycling the last x encrypted numbers to the beginning. However, rotations alone are not enough to perform arbitrary permutations on encrypted vectors. Instead, it requires an intricate circuit that combines additions, multiplications, and rotations. We call these permutation circuits. Halevi & Shoup [HS14] conjecture that finding the optimal (i.e., fastest given a maximum multiplicative depth) is a hard problem.

Previous work has focused on generating permutation circuits that permute a single ciphertext. However, for applications in the real world, not all data may be stored in the same ciphertext due to size constraints or because the data has different origins. Therefore, with the current solutions, the problem of permuting across multiple ciphertexts requires splitting the entire permutation into multiple within-ciphertext permutations. We highlight this problem in Figure 7.1. Solving this problem may also lead to improvements in the circuits for other applications, such as circuits that perform AES encryptions homomorphically.

In this work, we propose a new primitive that performs arbitrary mappings on values in ciphertexts and does so significantly cheaper than previous work regarding the computational effort required. These mappings are arbitrary in the sense that they may span multiple ciphertexts. Unlike previous methods which generate circuits for a chosen maximum multiplicative depth, our method focuses on a specific class of permutation circuits with a constant multiplicative depth. Still, we argue that our circuits' depth is reasonable for the complexity of the operation required. Our new primitive takes the burden off the implementor to create manual mapping circuits when data spans multiple ciphertexts. Its high efficiency brings secure outsourced computation one step closer to practice.

We summarize our contributions as follows:

- We propose a new method for efficiently performing arbitrary mappings on encrypted values in packed, leveled-homomorphic ciphertexts.

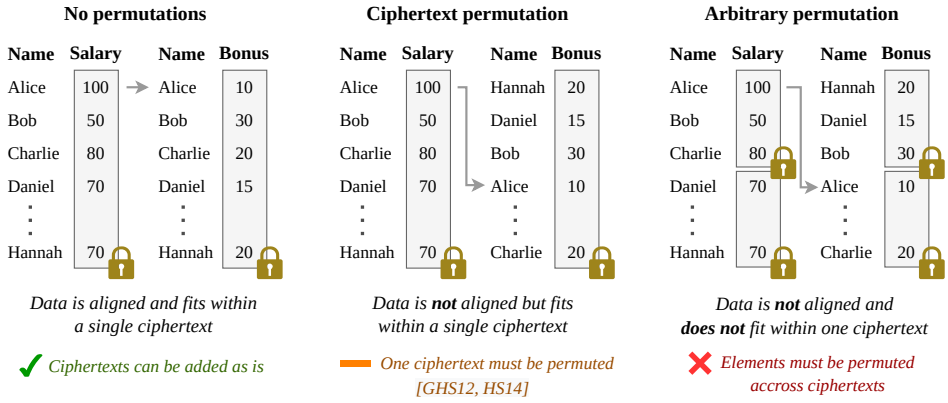


Figure 7.1: If data is not aligned between two ciphertexts, one of the ciphertexts must be permuted. The existing methods work when data fits within one ciphertext, but when data spans multiple ciphertexts they must be adapted and lose performance rapidly.

- We compare an open-source implementation of our method to HElib for performing permutations on single ciphertexts and show that it consistently outperforms HElib for circuits of similar multiplicative depth.

- We compare our implementation to an adjusted version of HElib to perform arbitrary permutations. We show that it outperforms HElib by more than an order of magnitude when the data is spread among five or more ciphertexts.

The remainder of this paper is structured as follows: In Section 7.2, we shortly explain operations in leveled homomorphic encryption, graph coloring, and the notation we use. In Section 7.3, we discuss related work. Next, in Section 7.4, we put forward our method for constructing mapping circuits, and in Section 7.5 we analyze its complexity. Finally, in Section 7.6 we compare our method against that implemented in HElib, after which we conclude in Section 7.7.

7.2 Preliminaries & Notation

In this section, we give a high-level explanation of the underlying techniques used in this paper. Table 7.1 contains a summary of the notation that we use.

7.2.1 Permutations & Mappings

We consider permutations and mappings of elements across vectors of length n. Here, we denote P as the set of indices to map, which is short for the preimage. We say that element $x \in P$ is permuted to position $\pi(x)$ when considering permutations, or mapped to position $\mu(x)$ in the case of a mapping. Note that permutations are a restriction of mappings.

Table 7.1: Summary of the symbols used in this work.

Symbol	Definition
ℓ	Number of slots in the ciphertext
n	Total number of elements to permute
$\pi(x)$	Target for index x after permuting
$\mu(x)$	Targets for index x after mapping
P	Set of indices to permute (preimage)
χ	Chromatic number (minimum number of colors)
$\phi(\_)$	Euler's totient function
m	Order of cyclotomic polynomial
p	Prime modulus defining the message space
Q	Ciphertext modulus defining the ciphertext space

7.2.2 Graph Coloring

Graph coloring is one of Karp's original 21 NP-complete problems [Kar72]. In this problem, we are given a loopless graph $G = (V, E)$ where we must assign a color to each vertex such that no two adjacent vertices share the same color. The minimum number of colors needed to be able to properly color G is the chromatic number χ. In this work, we translate the process of setting up an efficient homomorphic circuit for ciphertext mappings to the problem of graph coloring. While the problem is NP-complete in general, we can practically solve our instances here using algorithms such as DSATUR [Bré79].

7.2.3 Leveled Homomorphic Encryption Schemes

This work specifically considers leveled homomorphic encryption schemes that support packing multiple elements into one ciphertext. Here, leveled refers to the fact that we can only perform operations up to a certain level before decryption is likely to fail. The level is typically indicated as the multiplicative depth of the arithmetic circuit. The reason for this is that the ciphertexts incorporate a small noise term that grows with each homomorphic operation. This is why we speak of the *remaining noise budget* of a ciphertext, which we express as the number of bits of the ciphertext that the growing noise can still consume before the ciphertext is no longer decryptable. When there is a need to perform circuits of arbitrary depth, one can use bootstrapping techniques [Gen09]. In that case, we speak of *fully* homomorphic encryption. In our implementation, we only consider the BGV [BGV11] cryptosystem implemented in HElib, without bootstrapping operations.

One can add, multiply and rotate the values encrypted in a ciphertext. Element-wise additions are cheap operations between two ciphertexts with only small noise growth. In this work, we do not multiply ciphertexts together but only multiplications with constants, which is more efficient and incurs less noise growth. We use these plaintext multiplications to isolate values from the ciphertext by creating a mask that is zero everywhere except for the places with the elements we

Table 7.2: Comparison of permutation circuits generated by related work

Operation	Compute	Noise	Ciphertext permutation			Arbitrary	
			Naive	HElib	Ours	HElib*	Ours
Rotation	Expensive	Cheap	ℓ	$4\log(\ell)-2$	$\log^2(\ell)$	$O(n^2)$	$O(n)$
Plaintext mult.	Cheap	Moderate	ℓ	$4\log(\ell)-2$	$O(\log^3(\ell))$	$O(n^2)$	$O(n^2)$
Addition	Cheap	Cheap	ℓ	$2\log(\ell)-1$	$O(\log^3(\ell))$	$O(n^2)$	$O(n^2)$
Rotation keys	Severe	-	ℓ	$2\log(\ell)$	$\log(\ell)$	$2\log(\ell)$	$\log(\ell)$

need to isolate where it is 1. Rotations can be performed using automorphisms on the underlying ring. In this work, we only consider the case where those automorphisms cause one-dimensional rotations.

7.3 Related Work

To the best of our knowledge, the first work that studied permutations in leveled homomorphic ciphertexts was the work by Gentry et al. [GHS12]. In separate work, the same authors use it to implement an AES circuit homomorphically, which requires shuffling the elements within a ciphertext. Before that, Damgård et al. [DIK10] already used the underlying techniques within the context of secure multi-party computation to permute packed secret shares rather than ciphertexts. The underlying technique called Beneš networks [Ben64] originates in the study of efficient routing networks, which send packets from a range of senders to a range of receivers under constraints, effectively executing permutations.

In 2014, Halevi & Shoup [HS14] reduced the problem of constructing efficient permutation circuits for leveled homomorphic ciphertexts as a new problem named the cheapest-shift-network problem. Here, a shift-network is a series of shifts (permutations), which can be executed using additions, plaintext multiplications, and rotations. Each next shift considers only the shift before it. Halevi & Shoup put forward a method to efficiently optimize the computational cost of such a circuit given a maximum multiplicative depth, and implement it in the HElib library.[1] At the time of writing, we are not aware of other libraries that implement ciphertext permutations.

In this work, we consider a type of circuit that not only considers the layer before it but also any other layer before that. We also extend it beyond the range of a single ciphertext. In this sense, it is less restricted than the method proposed by Halevi & Shoup. However, it is an open question of how to optimize such a circuit efficiently, so we introduce other restrictions to turn the problem into one of graph coloring. For example, the multiplicative depth of our circuits scales logarithmically with the number of slots in a ciphertext. In the remainder of this section, we go into detail about the solutions of Gentry et al. [GHS12] and Halevi & Shoup [HS14] (summarized in Table 7.2) and explain how one can trivially but inefficiently extend them to perform arbitrary permutations and mappings.

[1] The HElib repository can be found at https://github.com/homenc/HElib

7.3.1 Naive Method for Permutations

A naive method for performing permutations within and across ciphertexts rotates each individual element to its target index and sums up the result. As mentioned before, elements can be isolated by multiplying them with a vector of zeroes and a 1 in the right index. This approach requires a plaintext multiplication, rotation, and addition for each of the ℓ slots in a ciphertext when performing a permutation within one ciphertext. Moreover, key generation will also be computationally expensive as one has to be able to perform each possible automorphism. Alternatively, one incurs an additional run time penalty for certain rotations by composing it from other rotations. Note that we can omit rotations of 0 and that there are scenarios where identical rotations can be rotated at the same time. Still, after these optimizations, the algorithm scales with $O(n)$ in the worst case.

7.3.2 'Collapsed' Beneš Networks for Permutations

Both the works by Gentry et al. [GHS12] and Halevi & Shoup [HS14] rely on Beneš networks. Such a network has a butterfly structure, which contains $2\log(\ell) - 1$ layers in the case of a ciphertext permutation. This structure makes it a shift-network that can be constructed efficiently in a recursive manner for all possible permutations. Elements are either rotated leftwards or rightwards in each layer by a given amount.

Gentry et al. use Beneš networks without any modifications, leading to a permutation circuit with a multiplicative depth that scales as $2\log(\ell) - 1$. Each layer only does a power-of-two rotation, meaning that one must generate $2\log(\ell)$ rotation keys.

Halevi & Shoup modify Beneš networks into other valid shift networks by collapsing layers to reduce the multiplicative depth of the resulting circuit. As mentioned before, they implement this in the HElib library. In Table 7.2 we consider the case where there is no bound to the multiplicative depth of the circuit. Since each layer of the network requires 2 plaintext multiplications and rotations, the total number is $4\log(\ell) - 2$ in the worst case.

7.3.3 Extending Permutation Circuits to Arbitrary Permutations

We remark that while previous works do not explicitly describe how to construct arbitrary permutations or mappings, they can be easily extended to do so. We shortly explain how the work Halevi & Shoup [HS14] can be extended as such by expressing the arbitrary permutation across multiple ciphertexts as a series of within-ciphertext permutations.

The key idea is that one can break a permutation across multiple ciphertexts into a set of permutations from each ciphertext to every other ciphertext. A similar trick can be used to perform mappings by first breaking it down into a set of arbitrary permutations. In the worst case, performing permutations in this way scales quadratically with the number of ciphertexts. When the elements are densely packed, we need a total of $\lceil \frac{n}{\ell} \rceil = O(n)$ ciphertexts. Here we consider ℓ to be constant. Consequently, the worst-case complexity for rotations, plaintext multiplications, and additions alike is $O(n^2)$.

7.4 Constructing Arbitrary Mapping Circuits

In this section, we propose our method for constructing circuits to perform arbitrary permutations and mappings. Since the construction only has to happen once for each permutation, it can be considered a one-time setup.

7.4.1 High-level Insight

The most time-consuming operation in a permutation circuit is a ciphertext rotation. Therefore, it stands to reason to minimize the number of rotations. Conversely, we want to maximize the number of elements we rotate at once. At the same time, since we have to generate special rotation keys for every possible rotation magnitude, we want to keep the number of different rotations as low as possible. In our method, we restrict all rotations to be powers of two. As we discuss later, this simplification allows us to optimize our permutation circuit efficiently. It is also possible to restrict rotations to powers of three (or any other base), but this requires certain rotations to be decomposed into a larger number of consecutive power of three rotations.

Given a permutation, we construct a circuit that realizes it by decomposing the number of places that each element must move into its binary representation. If there is a 1 in place x of the binary representation, we add the element to the set of elements that must be rotated by 2^x. For simplicity, let us fix the order of rotations in the final circuit as $2^0 = 1, 2^1 = 2, 2^2 = 4, \ldots$. One can imagine this idea as vertically-stacked conveyor belts that sequentially turn at increasing rates, as seen in Figure 7.2. In this figure, an element (pictured as a box) starts at index 1 and must end up at index 6. To do so, it must travel $5 = 101_2$ places rightwards, and therefore it enters the first and third conveyor belt, but not the second.

At first thought, the method described above seems to construct valid permutation circuits, but a problem arises when two elements must take the same place on the same conveyor belt. In an actual arithmetic circuit, this would add up the corresponding values of these elements, invalidating the permutation. In the right half of Figure 7.2, we visualize this. There are two simple solutions to this problem. Firstly, one might change the order of the conveyor belts. For example, one might bring the third conveyor belt to the start. Another approach is to add a second independent set of conveyor belts. In our method, we use both approaches: We try several different random orderings of conveyor belts and use a graph coloring algorithm to distribute elements over multiple sets of conveyor belts in a way that elements do not collide. We use the minimum number of conveyor belts given a certain order of conveyor belts.

7.4.2 Assigning Elements to Sets of Conveyor Belts

To assign the elements to multiple sets of conveyor belts, we construct a graph where the vertices represent elements of the encrypted vector. The edges between them represent that the elements cannot coexist in the same conveyor belts. After performing a graph coloring, the color of a vertex represents the set of conveyor

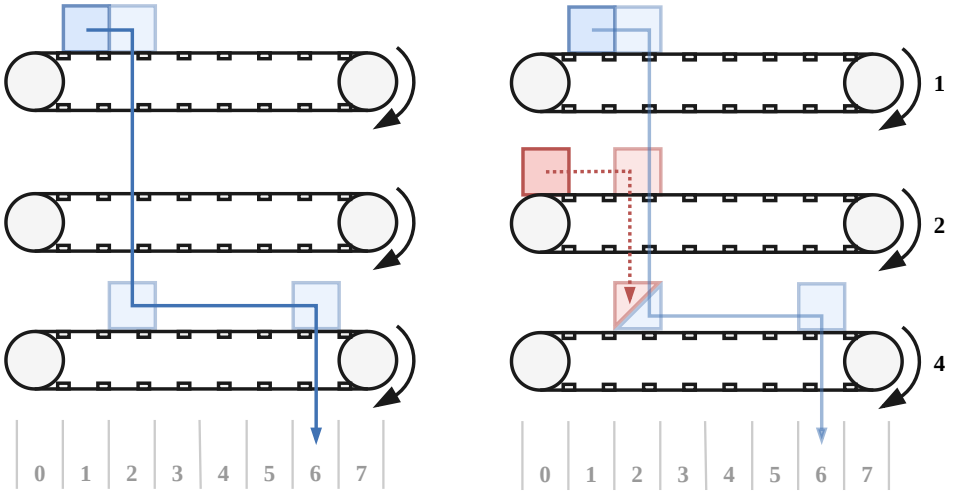


Figure 7.2: Elements can be mapped to other locations by applying a sequence of rotations on them, as if on a conveyor belt. Multiple elements can exist on the same set of conveyor belts so long as they do not enter the same conveyor belt at the same location.

belts to which it is assigned. In the remainder of this subsection, we refer to a single conveyor belt as a rotation.

For a permutation π with preimage P, we first create an undirected graph $G_\pi = (V, E)$, where $E = \emptyset$ and $V = P$. Then, for each element, we compute its position in the encrypted vector when it enters each rotation operation. If two elements $u, v \in P$ where $u \neq v$ enter the same rotation at the same position, we extend $E \leftarrow E \cup \{u, v\}$. This graph satisfies the property that any valid coloring represents a valid assignment. Figure 7.3 shows an example of such a graph and a possible coloring.

When we move beyond a permutation to a mapping μ, we must consider that elements in the preimage may map to multiple positions in the final encrypted vector (replication), or multiple elements in the preimage may map to the same position (overlapping). Notice that overlapping elements do not necessarily have to be assigned to different sets of rotations and that the graph G_μ constructed as described above already adequately handles such situations. The reason is that overlapping elements in the final encrypted vector do not necessarily overlap in the encrypted vectors to which rotations are applied. This graph also adequately handles replications, as all outputs relating to the same input element are assigned to the same set of rotations. This means that even in the extreme case where one element of the input ciphertext is mapped to all positions of the output ciphertext, we only require one set of rotations.

After generating the graph, we use a dedicated graph coloring algorithm to color the vertices with the minimum number of colors required. In our implementation, we use the DSATUR algorithm [Bré79], but any algorithm suffices.

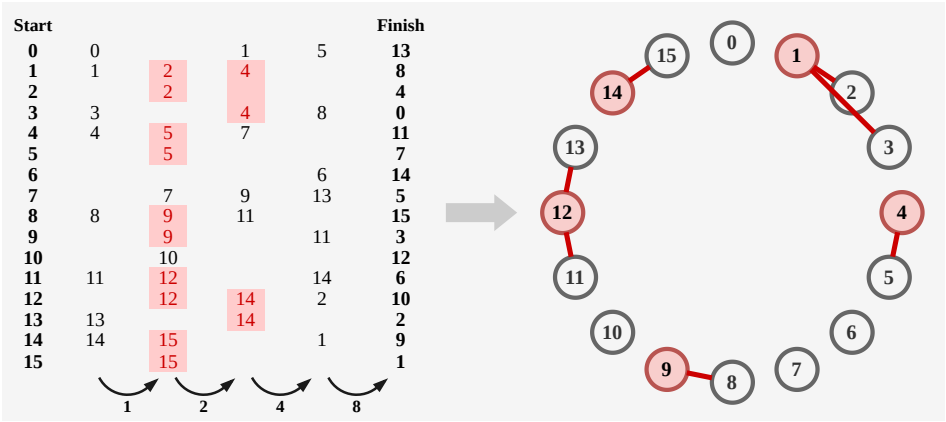


Figure 7.3: Example of the graph generated for a within-ciphertext permutation of 16 slots. The graph contains edges between the elements that would collide with each other at any of the rotations. This graph can be colored with two colors, but larger ciphertexts, across-ciphertext permutations, and mappings typically require more colors.

7.4.3 Determining the Order of Conveyor Belts

In the previous subsection, we did not explain how one should choose the order of the rotations. However, it follows that for the graph coloring to work, we require all sets of rotations to have the same order.

One approach is to fix the rotation order for every mapping. For example, $1, 2, 4, \ldots$. While this ordering performs well for random permutations and mappings, as we show in Section 7.6, one might try different orderings to avoid running into the worst-case behavior. In our implementation, we test multiple random orderings to find the one resulting in the graph that can be colored with the least colors. In our experiments, we compare the performance of trying only one random ordering against trying ten random orderings, which we refer to as a *long setup*.

It remains an open problem to integrate this step with the previous step to efficiently find an ordering that results in the minimum number of sets of rotations.

7.4.4 Generating Circuits for Conveyor Belts

Given an assignment that maps each element to a set of rotations, we construct a separate circuit for each set. Consequently, in a multi-threaded setup, one can execute these circuits in parallel. This subsection describes how to construct a circuit for one set of rotations, given a specific ordering of rotations and a set of elements that will not collide.

First, we create a set of masks for all the elements that must be included in a single rotation. In other words, we create one mask for each of the input ciphertexts and one mask for each of the ciphertexts resulting from all previous rotations. Such a mask contains ones in the positions of elements that must remain and zeroes in the positions of elements that must be dropped. We then perform a plaintext

multiplication between each ciphertext and the corresponding mask and sum up the results. The result is a ciphertext containing all the relevant encrypted values, which we subsequently rotate.

Note that there are several places where we can prune this circuit to prevent performing meaningless computations. For example, if we do not need to consider any values from a ciphertext, the corresponding mask would be empty (i.e., filled with zeroes). Moreover, we do not need to perform any summations if there is only one relevant ciphertext. We implement both of these optimizations, but we stress that more pruning is still possible. For example, by keeping track of which positions in each ciphertext actually contain values rather than zeroes, one can discard multiplications that mask all values in a ciphertext.

In the worst case, an element must be shifted $11\ldots11_2 = \ell - 1$ places in the encrypted vector. The resulting circuit then has a multiplicative depth of $1 + \log_2 \ell$ consecutive plaintext multiplications. When it comes to the asymptotic run time, each circuit only requires $\log_2 \ell$ rotations and, therefore, a total of $O(\log_2 \ell)$ plaintext multiplications and additions.

7.5 Performance of Special Mappings

In this section, we analyze the complexity of the circuits constructed by our method.

7.5.1 Permutations

In the case of permutations within a single ciphertext, the chromatic number χ of the graph that our method constructs to assign elements to sets of rotations is bound by $\log \ell$. We prove this in the following theorem:

Theorem 15. *It takes at most $\chi = K - 1$ colors to color graph G_π representing the collisions of permutation π with preimage P.*

Proof. It suffices to show that any element $x \in P$ can only collide with at most $\log_2(\ell) - 1$ other elements at one position. In that case, x and the other elements are all connected via an edge and must all be assigned a different color. For brevity, we denote $K = \log_2(\ell)$.

Let us express an upper bound for the maximum number of elements at a single position after r rotations as a function $M(r)$. At the first rotation, the maximum number of overlaps is $M(1) = 1$, because the encrypted vector has no overlaps. At every rotation after that, the maximum number of overlaps is that of the previous rotation, plus one element that was already in this position, so $M(i) = M(i-1) + 1$. This only holds for $i = 2, \ldots, K-1$, however, because at the Kth rotation, the result must not have any overlaps given that π is a permutation. So, $M(K) = 0$. Our function M is undefined for any other values.

We reach the maximum number of overlapping elements at any rotation at $M(K-1) = K-1$. In fact, this upper bound overestimates the number of overlapping elements, because, after r rotations, the overlapping elements can only move to 2^{K-r} remaining positions, so $K-1$ overlapping at $M(K-1)$ cannot satisfy a valid permutation. □

As a result, we require at most $\log(\ell)$ sets of $\log(\ell)$ rotations. Also notice that in the case of arbitrary rotations, the number of rotations required is $O(n)$, when ℓ is kept constant. This is because even in the worst case where each of the n elements to be permuted is assigned to a separate set of rotations, the relation is linear. However, this situation should be seen as an upper bound because when the number of elements grows, the sets of rotations become more densely packed in the average case. The number of plaintext multiplications and additions scale quadratically with the number of rotations because before the xth rotation there can be additions and multiplications with the prior $x - 1$ resulting ciphertexts.

7.5.2 Bounded Rotation Magnitude

The number of rotations that one element occupies is exactly the number of ones in the binary representation of the distance it must move. This number, which is called the Hamming weight, is $\frac{1}{2}\ell$ on average for random permutations. However, if the distance that elements move is bound or the Hamming weight of the distances is low, we expect to pack more elements within one set of rotations.

7.6 Results

In this section, we analyze the performance of our open-source implementation[2] and compare it against HElib. To facilitate a fair comparison, we execute our circuits with HElib's implementation of BGV. Note, however, that any FHE library can execute the resulting circuits with minimal engineering effort.

We perform three sets of experiments, which are increasingly generic. We start by comparing the performance of permutations within a single ciphertext to HElib. Then, we extend HElib to perform arbitrary permutations across multiple ciphertexts and compare the implementation against our work. Finally, we analyze the run time performance of our implementation when performing arbitrary mappings for increasing degrees of overlapping and replication.

Table 7.3 contains the parameters we used for our experiments. We choose the order of the cyclotomic polynomial $m = 2^x$ for some x, following the homomorphic encryption standard [Alb+18]. Since the number of slots $\ell = \frac{\phi(m)}{\text{ord}(p)}$, we want the plaintext modulus p to have a low order modulo m. On the other hand, when ℓ is large, the depth of our circuits might cause the noise in the ciphertexts to grow too large. So, we choose the highest ℓ for which the ciphertexts are still decryptable while selecting the lowest p that satisfies it. We provide the number of bits in the modulus chain $\log_2 Q$, which we maximized while satisfying 128 bits of security as specified by the homomorphic encryption standard [Alb+18].

[2]The repository is at DOI `10.4121/4b3dfb12-35e5-4d77-82ea-9758ac6dec18` and on GitHub

Table 7.3: BGV parameters used in the experiments

	Order m	Modulus p	$\log_2 Q$	Slots ℓ	HElib's depth
Small	$2^{13} = 8192$	31	111	$2^4 = 16$	4
Medium	$2^{14} = 16384$	127	213	$2^6 = 64$	7
Large	$2^{15} = 32768$	5119	<440	$2^6 = 64$	9

We executed all our experiments on a Unix machine with 16 virtual Intel® Xeon® Cascade Lake CPUs at 3100 MHz and 64 GB of memory. However, we only executed our experiments on a single thread. While our technique would work on any leveled homomorphic RLWE-based ciphertexts, we used the BGV cryptosystem in our experiments. Since the actual contents of the ciphertexts do not influence the performance in our experiments, we choose repeated encryptions of $0, \ldots, p-1$.

7.6.1 Within-ciphertext Permutations

Since HElib's permutation circuits aim to perform permutations on single ciphertexts, we compare its performance with that of our method. We test performance on the same 50 randomly-generated permutations. In Figure 7.4 we show the mean run time to perform such a permutation, not considering the setup time, which is considerably smaller. Notice that our method outperforms HElib in each scenario. Moreover, while we execute the separate sets of rotations consecutively in these experiments, one can execute them on separate threads for an even larger speed-up. On the other hand, unlike HElib, our method does not allow the user to specify a maximum circuit depth, so this is only a suitable alternative when the ciphertext's noise budget is large enough.

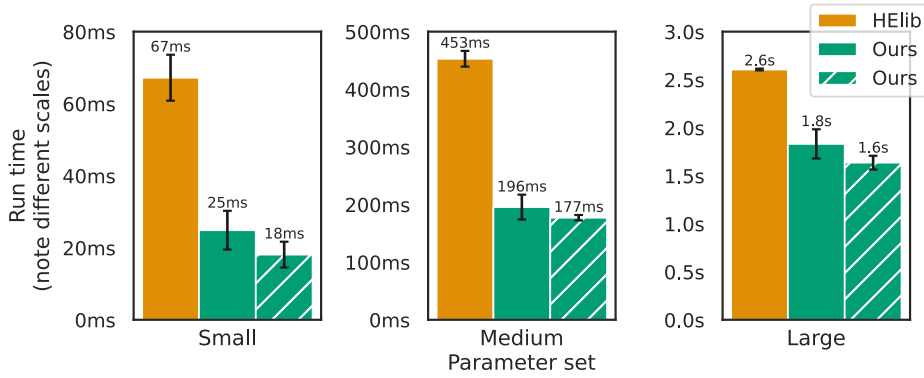


Figure 7.4: While our circuits are not specifically made for permutations within ciphertexts, they outperform HElib in execution time for a similar noise budget by a factor 1.4× for large parameters up to 2.7× for small parameters. The error bars denote the standard deviation.

In our experiments, we aimed for the remaining noise budgets between our method and HElib's method to be similar, as displayed in Table 7.4. To do so, we

set the depth bound for HElib's permutation circuit as displayed in the rightmost column of Table 7.3.

Table 7.4: Average remain noise budget of the resulting ciphertext expressed in bits. Here, higher is better, but we selected the parameters for both works to perform similarly.

	Small	Medium	Large
HElib	11.72	10.14	26.86
Ours	5.38	22.24	29.68
Ours (long setup)	5.46	22.34	29.76

7.6.2 Arbitrary Permutations

Next, we evaluate the performance when the number of ciphertexts we permute across grows. We measure the execution time for each number of ciphertexts over 20 random permutations, disregarding our long-setup method. We present the results in Figure 7.5. The experiment supports the worst-case complexities that predict HElib's method to scale quadratically and our method linearly regarding the number of ciphertext rotations, which make up the most expensive operation. The improvement in run time is significant, exceeding an order of magnitude starting from as little as five ciphertexts.

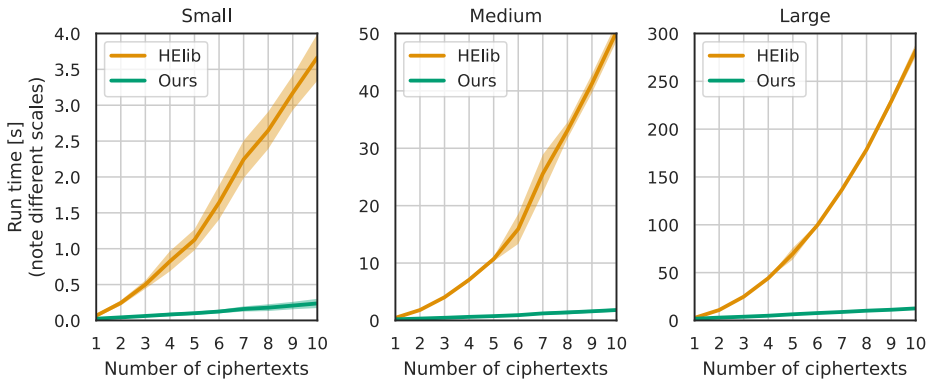


Figure 7.5: Execution time for random permutations among a growing number of ciphertexts. The experiment confirms that the execution time of HElib scales quadratically, while our approach scales linearly. The shaded area represents the 99% confidence interval.

7.6.3 Arbitrary Mappings

Finally, we evaluate the setup and execution time required for performing arbitrary mappings using our method. We do not consider HElib's method for these experiments, which is prohibitively expensive when the overlap or replication degree exceeds 1. Our experiment considers random mappings across eight

ciphertexts, which we generate by creating a set of possible targets and distributing them among the indices of each ciphertext, taking into account the overlap and replication constraints. We present the results in Figure 7.6. In this figure, the upper left corner is an arbitrary permutation, and the leftmost column represents injective mappings (replications). Notice that the small and medium parameters finish in the order of seconds, even when elements in the output are allowed to overlap with three other elements. Also, notice that both the setup time and execution time only significantly increase when *both* the overlap and replication degree.

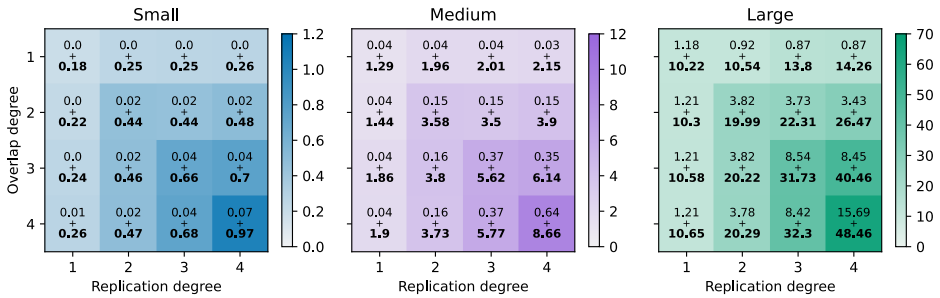


Figure 7.6: Total time in seconds of arbitrary mappings for increasing overlap and replication degrees. The bold number is the execution time, while the time above is the setup time. Notice that the times hardly increase when only one of the parameters grows and that the setup time becomes non-negligible for higher replication and overlap degrees.

7.7 Conclusion

To the best of our knowledge, this work proposes the first efficient method for constructing mapping circuits across multiple ciphertexts. We experimentally show that our method consistently outperforms the algorithm in HElib, given a ciphertext that supports a large enough multiplicative depth.

Still, open questions remain:

1. Future work can optimize the generated circuits by pruning parts of the circuit. For example, there is no need to isolate elements using a plaintext multiplication when the ciphertext already only contains those elements.

2. Future work might look for an optimization algorithm that separately optimizes the order of rotations.

3. In our current method, all sets of rotations contain all power-of-two rotations, but one might construct shallower circuits by considering using only a subset of those rotations. Such a method would require a different optimization algorithm, however.

With our new primitive, one can construct efficient permutation circuits for permuting elements within a single ciphertext and across multiple ciphertexts.

Where previous methods scale quadratically with the number of elements to permute, our method scales linearly regarding the total number of rotations to perform. Our method is concretely efficient when previous work becomes prohibitively expensive.

References

[Alb+18] Martin Albrecht et al. *Homomorphic Encryption Security Standard*. Tech. rep. Toronto, Canada: HomomorphicEncryption.org, Nov. 2018.

[Arm+10] Michael Armbrust et al. "A view of cloud computing". In: *Commun. ACM* 53.4 (2010), pp. 50–58. DOI: 10.1145/1721654.1721672. URL: http://doi.acm.org/10.1145/1721654.1721672.

[Ben64] V. E. Beneš. "Optimal rearrangeable multistage connecting networks". In: *The Bell System Technical Journal* 43.4 (1964), pp. 1641–1656. DOI: 10.1002/j.1538-7305.1964.tb04103.x.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "Fully Homomorphic Encryption without Bootstrapping". In: *IACR Cryptol. ePrint Arch.* (2011), p. 277. URL: http://eprint.iacr.org/2011/277.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_50.

[Bré79] Daniel Brélaz. "New Methods to Color the Vertices of a Graph". In: *Commun. ACM* 22.4 (Apr. 1979), pp. 251–256. ISSN: 0001-0782. DOI: 10.1145/359094.359101. URL: https://doi.org/10.1145/359094.359101.

[Che+17] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15. URL: https://doi.org/10.1007/978-3-319-70694-8%5C_15.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. "Perfectly Secure Multiparty Computation and the Computational Overhead of Cryptography". In: *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*. Ed. by Henri Gilbert. Vol. 6110. Lecture Notes in Computer Science.

Springer, 2010, pp. 445–465. DOI: 10.1007/978-3-642-13190-5_23. URL: https://doi.org/10.1007/978-3-642-13190-5%5C_23.

[FV12] Junfeng Fan and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption". In: *IACR Cryptol. ePrint Arch.* (2012), p. 144. URL: http://eprint.iacr.org/2012/144.

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440. URL: https://doi.org/10.1145/1536414.1536440.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Fully Homomorphic Encryption with Polylog Overhead". In: *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. Lecture Notes in Computer Science. Springer, 2012, pp. 465–482. DOI: 10.1007/978-3-642-29011-4_28. URL: https://doi.org/10.1007/978-3-642-29011-4%5C_28.

[HS14] Shai Halevi and Victor Shoup. "Algorithms in HElib". In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Springer, 2014, pp. 554–571. DOI: 10.1007/978-3-662-44371-2_31. URL: https://doi.org/10.1007/978-3-662-44371-2%5C_31.

[Kar72] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*. Ed. by Raymond E. Miller and James W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9. URL: https://doi.org/10.1007/978-1-4684-2001-2%5C_9.

The road ahead

Chapter 8

Oraqle Extended: Optimal Automated Protocol Design

for Homomorphic Encryption-based Multi-Party Private Set Intersections

While designing the protocols in Part A, we often wondered if it is possible to generate these protocols automatically. After all, Chapter 1 already defines high-level constructions for MPSI protocols, and abstractions like EMQFs as presented in Chapter 4 allow one to construct private set operation protocols from simple building blocks. Moreover, protocols for private set operations often use the same arithmetization of AND and OR operations. Automatic protocol design could perform these arithmetizations so that protocol designers do not have to reinvent them.

In this chapter, we extend the oraqle compiler to design homomorphic encryption-based protocols for MPSI protocols. As such, we show how to combine insights from Parts A and B. Unlike the oraqle compiler, which considers somewhat homomorphic encryption, we also consider partially homomorphic encryption, with the goal of having the compiler automatically generate the protocol from Chapter 3. Depending on the parameter choices, the extended oraqle compiler can also come up with slightly more efficient variants of this protocol by distributing computations among the clients, while remaining in the star topology.

This is an unpublished chapter detailing initial steps of combining the insights and techniques that were proposed in Parts A & B.

8.1 Introduction

Designing secure computation protocols is a task reserved for experts. It requires making design choices that optimize efficiency without sacrificing security. Moreover, since its inception, the research community has developed many new secure computation techniques. While this large increase in secure computation techniques allows designing more efficient protocols than before, it also means that there are now so many techniques to choose from that, even for experts, it is not possible to be familiar with all of them. As a result, it remains an open problem to design the *most* efficient secure computation protocol that computes a desired function given current secure computation techniques.

Since designing secure computation protocols is such a laborious task, it is typically hard to tailor previously-proposed protocols to a specific computing infrastructure. For example, the private set operation protocols proposed in

Chapter 3 centralize the majority of computations around a central party $\mathcal{P}_1$. It would take careful analysis to design a protocol that instead distributes most of the computations among the other parties. It raises the question whether a protocol that performs better than another is better because it corresponds closer to the computational infrastructure, or whether the underlying circuit or combination of secure computation techniques achieves better performance in general.

In this chapter, we offer an initial solution to these problems by proposing algorithms for automatically designing secure computation protocols from high-level descriptions. We limit the set of secure computation techniques to a subset of homomorphic encryption schemes (see Table 8.1). After all, homomorphic encryption is well-suited to the client-server model, as it does not require significant involvement of the clients during the computations, and unlike in secret sharing-based techniques, computations can be performed by a single party. Even within this smaller set of homomorphic encryption schemes there is a wide variety of computational properties and plaintext algebras, which allows one to design protocols with radically different performance characteristics. We note that, while the techniques in this chapter apply to all these schemes, we leave a practical implementation to future work.

We split the problem of automatically designing secure computation protocols from high-level descriptions into two sub-problems:

- **Expressing high-level circuits as extended arithmetic circuits:** Given a circuit containing high-level values and operations such as set representations and intersections, we consider how an algorithm can generate an extended arithmetic circuit that confidentially computes the high-level circuit. Next to additions and multiplications over some plaintext algebra, these extended arithmetic circuit enable selectively disclosing values and they incorporate randomness. These additional operations allow them to closely model the computational model of many secure computation techniques such as homomorphic encryption and secret sharing [BMY24].

- **Assigning & scheduling computation among the parties:** Deploying these extended arithmetic circuits requires deciding which party performs which computations. As such, we need to assign & schedule the computations in an extended arithmetic circuit among the parties involved in the secure computation protocol. In other words, we transform the circuit into a protocol. In this chapter, we show how to perform this transformation optimally with respect to two different optimization objectives.

In the remainder of this chapter we will refer to secure computation protocols as multi-party computation protocol interchangeably. We use n to denote the number of parties participating in a protocol, and t to denote the maximum number of colluding parties that the protocol can withstand. We note that our focus on MPC protocols does not exclude secure outsourced computation: one may model a secure outsourced computation task by defining one party who supplies the inputs but whose computations are infinitely expensive to compute locally.

Table 8.1: Non-exhaustive overview of homomorphic encryption schemes. The last column indicates whether they can be supported in our extension of the oraqle compiler.

Technique	Homomorphism		Plaintext algebra		Supported
	+	×	R	Structure	
Partially-homomorphic encryption					
Paillier [Pai99]	∞	0	$\mathbb{Z}_{(pq)^2}$	Ring	●
ElGamal [Gam84]	0	∞	$\mathbb{Z}_q^*$	Group	○
EC-ElGamal (see Ch. 3)	∞	0	$\mathbb{Z}_q$	Ring	●
Somewhat-homomorphic encryption					
BGN [BGN05]	∞	1	$\mathbb{Z}_q$ or $\mathbb{Z}_{q_T}$	Rings	○
BGV [BGV11]	Exp.	Poly.	$\mathbb{F}_p[X]/(X^N+1)$	Ring	●
w/ [SV14]	Exp.	Poly.	$(\mathbb{F}_p)^S$	Fields	●
BFV [Bra12; FV12]	Exp.	Poly.	$\mathbb{F}_p[X]/(X^N+1)$	Ring	●
w/ [SV14]	Exp.	Poly.	$(\mathbb{F}_p)^S$	Fields	●
CKKS [Che+17]	Exp.	Poly.	$\mathbb{C}^S$	Fields	○
Fully-homomorphic encryption					
BGV + bootstrapping	∞	∞	$\mathbb{F}_p[X]/(X^N+1)$	Ring	●
BFV + bootstrapping	∞	∞	$\mathbb{F}_p[X]/(X^N+1)$	Ring	●

8.1.1 Previous work

We briefly review other works that generate secure multi-party computation protocols from high-level specifications. We also discuss how they handle assignment & scheduling.

MPCircuits [Ria+19] is a pipeline for generating multi-party Boolean garbled circuits from high-level descriptions. Its objective is to generate circuits with the minimal number of non-XOR gates. In other words, it minimizes the multiplicative size. It uses the multi-party BMR protocol [BLO16] to evaluate these circuits, which implicitly handles assignment & scheduling: Because this work considers the dishonest majority setting with $t = n-1$, each party must perform all computations and each pair of parties must communicate with each other.

Hastings et al. [Has+19] discuss several other general-purpose MPC compilers. The tools they describe use different techniques to evaluate the circuits they generate: most tools use garbled circuits, Wysteria [RHH14] uses the GMW protocol [GMW87] (which relies on oblivious transfers, see Section 1.3.2), and others propose hybrid protocols, which switch between plaintext algebras or secure computation techniques throughout the computation. For example, ABY [DSZ15] is a compiler specifically for two-party secure computation that uses one of three techniques: GMW using arithmetic (so relying on oblivious linear evaluation, see Section 1.3.2), GMW using a Boolean circuit and garbled circuits. Apart from ABY, Hastings et al. discuss three other tools that support arithmetic protocols:

- SCALE-MAMBA [Aly+25] is a compiler that relies on secret sharing and

implements many sub-protocols using other cryptographic primitives, such as homomorphic encryption. It essentially hardcodes which sub-protocols to run for each operation, but it does implement multiple low-level optimizations.

- Sharemind [BLW08] is a proprietary tool that focuses on three-party computation but it also supports a larger number of parties. The tool uses secret sharing and is in the honest majority setting, so when there are three parties, all parties perform the same amount of computation.
- PICCO [ZSB13] is another secret sharing-based compiler that uses different kinds of sub-protocols for operations such as floating point operations or comparisons.

Another tool for generating secure multi-party computation protocols that was not discussed by Hastings et al. is MPyC. This tool takes a similar approach to the oraqle compiler, by implementing arithmetizations of several high-level primitives instead of supporting arbitrary programs in a high-level language. In fact, these arithmetizations also include randomness and reveal operations. MPyC considers the honest majority setting, evaluating the circuits it generates using Shamir's secret sharing scheme. They evenly distribute computations over all parties by assigning computations on the fly, keeping track which was the last party to receive a computation.

Other works that propose techniques for generating hybrid protocols These works go beyond arithmetic circuits with reveal and random operations because they perform operations such as scheme switching. Because hybrid protocols are more expressive than extended arithmetic circuits that are evaluated using a single secure computation technique, they can be more efficient. However, given the larger search space, it is hard to prove optimality. These hybrid protocols lead to particularly large efficiency gains when performing both arithmetic on and comparisons between large integers.

Another example of a work that generates hybrid protocols is HyCC [Büs+18], which was not discussed by Hastings et al. This multi-party compiler shares similarities with the ABY compiler. After splitting the high-level circuit into modules, it generates multiple possible circuits for each module. Specifically, it generates a Boolean circuit minimizing multiplicative size, a Boolean circuit minimizing multiplicative depth, and an arithmetic circuit. The arithmetic circuit is a straightforward mapping from arithmetic expressions to arithmetic circuits, so it does not generate an arithmetic circuit for operations such as equality checks and comparisons. HyCC also performs constant folding and no-op removal on all circuits. While HyCC can generate hybrid protocols, it does not consider Boolean circuits that trade off multiplicative size and depth (only the two extremes), and it does not consider arithmetizations of high-level operations beyond Boolean circuits. When it comes to assignment & scheduling, HyCC employs the simple approach of scheduling operations at the same time if they were also computed in parallel in the high-level program. The authors explain how this typically leads to a lower round complexity but that this method is indeed not optimal.

Finally, MP-SPDZ [Kel20] is a compiler similar to SCALE-MAMBA, but with a many more possible protocols that can be used to evaluate the circuits. MP-SPDZ

approach to scheduling computations is that it schedules them as soon as possible, but in a way that minimizes the number of rounds. By bundling computations in rounds, one can minimize the amount of intermediate communication. In general, MP-SPDZ does not take into account that one party may be faster at computing some operations than others.

In short, previous work focuses on generating Boolean circuits that are typically evaluated using secret sharing-based techniques or garbled circuits. Works that do use arithmetic, perform a binary decomposition protocol to switch to Boolean circuits anyways when performing complex computations such as equality checks or comparisons, which is not always the most efficient (see Table 6.2). In this work, we consider arithmetic circuits, and we evaluate these circuits using homomorphic encryption. The use of homomorphic encryption allows one to assign computations to the party that is most efficient at performing them. By considering arithmetic circuits beyond Boolean circuits, we can make full use of the plaintext algebra provided by these schemes.

8.1.2 Our solution

We solve the problem of expressing high-level circuits as extended arithmetic circuits by extending the oraqle compiler (see Chapter 6) beyond arithmetic circuits. We extend the compiler in three ways, allowing us to arithmetize private set intersections and to support a wide variety of homomorphic encryption schemes as listed in Table 8.1:

- The oraqle compiler assumes the plaintext algebra to be a prime field $\mathbb{F}_p$. We implement another plaintext algebra in the form of commutative rings, which allows the usee of homomorphic encryption schemes such as elliptic curve-based ElGamal. Note that not all arithmetization techniques described in Chapter 5 translate over to this algebra.

- We add extended arithmetic circuits, which besides arithmetic operations may contain reveal and random nodes (see Section 8.2). This extension is based on the computational model proposed by Blanton et al. [BMY24].

- We introduce new encodings for Booleans, which permit convenient conjunction operations. We use these Booleans to implement bitsets (see Chapter 1), which, in turn, permit convenient intersections.

As such, the supported homomorphic encryption schemes listed in Table 8.1 are schemes with a plaintext algebra satisfying the properties of a ring. Moreover, these schemes must have a finite characteristic and the plaintext algebra must remain the same throughout the entire computation. As a result, this chapter does not pertain to the CKKS homomorphic encryption scheme because its plaintext algebra has an infinite characteristic, meaning that the arithmetizations described in this chapter do not apply. It also does not support the BGN cryptosystem because its plaintext algebra changes throughout the computation: fresh messages are in the ring $\mathbb{Z}_q$, whereas the results of multiplications are in another ring $\mathbb{Z}_{q_T}$. By EC-ElGamal, we mean the cryptosystem resulting by instantiating ElGamal using

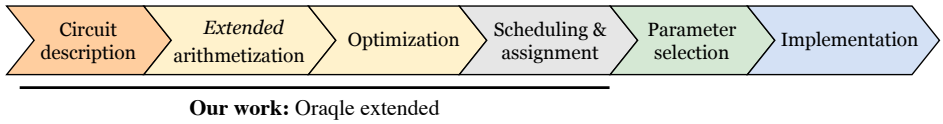


Figure 8.1: Our pipeline, which separates *arithmetization* and *scheduling & assignment*.

an elliptic curve, and encoding a message $m \in \mathbb{Z}_q$ as mG where G is a generator point. We provide more details in Chapter 3.

We provide two solutions to the second problem of assigning & scheduling computations that differ in their objective. Our first solution optimizes for efficiency by minimizing the total cost of the protocol. The total cost is the sum of all the tunable costs of individual operations, such as the cost of party $\mathcal{P}_i$ computing a multiplication, or for $\mathcal{P}_i$ to send a value to $\mathcal{P}_j$. Our second solution instead minimizes the total run time of a protocol, from the first party contributing their inputs to the last party concluding the protocol with their outputs.

One intricacy that comes up during the assigning & scheduling phase is that one must decide how to generate the random values required in the protocol. While it is possible for these values to be generated by a trusted third party, this is often undesirable. In Section 8.4 we discuss other approaches for realizing randomness such that the value remains unknown to all possible colluding subsets of parties.

We summarize the phases of compiling a high-level circuit into a secure computation protocol in the pipeline diagram in Figure 8.1. Here, the optimization phase is made up of the existing optimization methods implemented in the oraqle compiler such as common subexpression elimination. This phase also includes the extended arithmetic circuit-specific optimizations described in Section 8.2.3.

8.1.3 Contributions

In Section 8.5, we use a bitset-based private set intersection circuit to show that the extended oraqle compiler can generate the handcrafted MPSI protocol from Chapter 3 automatically. In doing so, we make the following scientific contributions:

- We provide a definition of secure encodings and propose a realization of a Boolean encoding that permits an efficient conjunction operation.
- We use this encoding to generate an extended arithmetic circuit from a high-level circuit for performing set intersections.
- We propose two formulations for optimally assigning & scheduling extended arithmetic circuits with respect to two different optimization objectives.

We note that we do not parameterize and implement the resulting protocol in this chapter. However, we can use the objective functions to quantify these protocol's theoretical performance.

8.1.4 Outline

This chapter is structured as follows. In the next section, Section 8.2, we discuss the concept of extended arithmetic circuit and what it means for such a circuit to be secure. After that, in Section 8.3, we define secure encodings for Booleans and sets that allow one to conveniently compute conjunctions and intersections, respectively. In Section 8.4, we propose methods for (optimally) transforming extended arithmetic circuits into MPC protocols. Finally, in Section 8.5, we present initial results, and in Section 8.6, we discuss the main findings in this chapter and conclude with limitations and future work.

8.2 Beyond arithmetic circuits

The oraqle compiler implements arithmetization of high-level circuits to pure arithmetic circuits: the resulting circuits only contain additions and multiplications. Arithmetic circuits describe the circuits that can be non-interactively and deterministically computed using homomorphic encryption (ignoring automorphism operations). However, when secure computation may involve interaction, it is possible for parties to reveal intermediate values. Next to that, there is no theoretical requirement for these circuits to be deterministic. These two notions allow for significantly more efficient secure computation protocols than when using pure arithmetic circuits [BMY24]. In this section, we discuss how to extend the oraqle compiler to *extended* arithmetic circuits. This formalization has been used implicitly in SCALE-MAMBA [Aly+25] and PICCO [ZSB13], and a variant was proposed explicitly in the work by Blanton et al. [BMY24]. We present our own version of this formalization that excludes redundant operations, and we discuss what it means for such a circuit to be secure. Finally, we present several simple optimizations.

8.2.1 Extended arithmetic circuits

Blanton et al.'s computational model [BMY24] of extended arithmetic circuits contains five additional functions. The authors use the notation $[x]$ to denote a secret share of a value x. We use this notation to also mean any type of encryption of x. The additional functions are:

- $x \leftarrow \mathsf{Open}([x])$ reveals the value to all parties.
- $[x] \leftarrow \mathsf{RandFld}()$ generates an encryption of a random field element.
- $[x] \leftarrow \mathsf{RandInt}(k)$ generates an encryption of a random k-bit unsigned integer.
- $z \leftarrow \mathsf{MulPub}([x], [y])$ reveals $z = x \cdot y$ to all parties.
- $[z] \leftarrow \mathsf{DotProd}(([x_1], \ldots, [x_\ell]), ([y_1], \ldots, [y_\ell]))$ generates $[\sum_{i=1}^{\ell} x_i \cdot y_i]$.

We define a variant of these extended arithmetic circuits that is more suited to secure computation beyond secret sharing because it does not require revealing values to all parties. Next to that, we reduce the set of additional nodes to two:

- $x \leftarrow \mathsf{Reveal}_S([x])$ reveals x to each party $\mathcal{P}_i$ where $i \in S$.
- $[x] \leftarrow \mathsf{Random}_P()$ generates an encryption of a random sample from probability distribution P.

As mentioned before, the key differences are that Reveal only reveals to a (sub)set of parties, whereas Open always reveals to all parties, and that our model provides fewer functions. Specifically, randomness is now captured by a single function parameterized by a probability distribution. Finally, we remove functions MulPub and DotProd altogether: MulPub can be realized by a multiplication followed by Reveal, and DotProd can be expressed using additions and multiplications.

We note that this model still restricts the enivisioned computational model. Specifically, in several use cases it may be required to perform a re-encryption by masking the value $[x]$, revealing it, and encrypting it again. For example, this would allow a BGN ciphertext that was already used in a multiplication to be multiplied again. Another example is that of a BGV or BFV ciphertext that is close to its noise limit. By re-encrypting the value, the we obtain a fresh ciphertext with low noise. The designer of such a protocol may not want to specify which parties must perform this re-encryption, but only that it must occur. Ideally, the designer specifies that the randomness can only be known to a large colluding subset of parties, and that the party that obtains the revealed value is not part of this colluding subset. Blanton's model cannot express this because $\mathsf{Open}([x])$ reveals the value to all parties. Our model can also not express this for the case where $t < n-1$ because $\mathsf{Random}_P()$ does not allow specifying t and $\mathsf{Reveal}_S([x])$ requires a set S of party indices: it is not possible to specify that $i \notin \mathsf{Known}(\mathsf{Random}_P()) : \forall i \in S$, which states that the parties that observe the masked value may not know the mask. Here, we use $\mathsf{Known}([x])$ to denote the set of parties that know plaintext value x.

8.2.2 Security of compiled circuits

Let us use the notation of Chapter 5 to denote circuits: a circuit is a directed acyclic graph C. We use $C(x_1, \ldots, x_k)$ to denote the evaluation of the circuit on inputs $x_1, \ldots, x_k$, which outputs one or more values for each party (i.e. it outputs an n-tuple). We use the notation $\mathsf{Outputs}_i(C(x_1, \ldots, x_k))$ to specifically denote the outputs of party $\mathcal{P}_i$. We say that the inputs are in some algebraic structure R that satisfies the properties of a commutative ring or prime field. The circuit may consist of arbitrary nodes (i.e. high-level nodes) or extended arithmetic nodes. In this chapter, we define what it means for such a circuit to be secure in the context of homomorphic encryption.

To define security regardless of the homomorphic encryption scheme, we focus on the behavior of circuits in the plaintext space. We consider security of a circuit with respect to another envisioned high-level circuit (i.e. the ideal functionality). We first define what it means for a circuit to be perfectly correct:

Definition 11 (Perfect correctness)**.** Let C denote the ideal high-level circuit, and C' another circuit for computing C. We say that circuit C' perfectly correctly evaluates C if and only if: $C(x_1, \ldots, x_k) = C'(x_1, \ldots, x_k) : \forall (x_1, \ldots, x_k) \in R^k$.

In other words, circuit C' is only perfectly correct with respect to C if it is equivalent. Next, we define confidentiality. The intuition behind this definition is that, if you can simulate a circuits outputs from only the intended outputs and the intended revealed intermediate values, then it reveals nothing more about the inputs than the intended high-level circuit already would.

Definition 12 (Confidentiality)**.** Let C denote the ideal high-level circuit, and C' another circuit for computing C. We say that circuit C' confidentially computes C if and only if the following pairs of outputs are indistinguishable:

$$\begin{gathered} S_i(K_i, \mathsf{Outputs}_i(C(x_1, \ldots, x_k))) \\ \text{and} \\ \mathsf{Outputs}_i(C'(x_1, \ldots, x_k)), \end{gathered} \qquad \forall i \in [1, n], \forall (x_1, \ldots, x_k) \in R^k \ ,$$

where S is a simulator, which can be any efficiently computable function, and K_i is the set of intermediate values $z \in C$ where $\mathsf{Known}(z) = \mathcal{P}_i$.

We note that when a circuit is perfectly correct with respect to the envisioned high-level circuit, it also confidentially computes that circuit. In this case, the simulator is the identity function. A second important insight is that perfectly correct sub-circuits can be composed into larger perfectly correct circuits. The oraqle compiler from Chapter 6 exploits these two facts, expanding high-level sub-circuits into perfectly correct arithmetic sub-circuits that are in turn composed to achieve the envisioned high-level circuit. As such, the circuits generated by the oraqle compiler achieve confidentiality.

So far, we have only considered operations in the plaintext space, but in practice we require a protocol that computes these plaintext operations obliviously. In this chapter, we use homomorphic encryption for this task. For IND-CPA-secure noiseless homomorphic encryption schemes such as the Paillier cryptosystem [Pai99], one can prove the security of such a protocol in the semi-honest model by noticing that the following holds. for a homomorphic ciphertext operation $\otimes$ that results in plaintext operation $\odot$, the following are indistinguishable:

$$\mathsf{ReRand}_k(\mathsf{Enc}_k(a) \otimes \mathsf{Enc}_k(b)) \text{ and } \mathsf{Enc}_k(a \oplus b) \ .$$

Here, $\mathsf{Enc}_k(m)$ is the encryption of message m under key k, and ReRand rerandomizes the ciphertext, for example by adding a fresh encryption of 0 in the context of Paillier encryption. For noisy homomorphic encryption schemes such as BFV and BGV, proving security is more complicated as such a rerandomization function must also ensure that the noise does not reveal information about the secret key and inputs or intermediate values. This has recently received a great deal of attention from the scientific community, specifically in the context of achieving IND-CPA-D security, which is an extension of the IND-CPA security game with a restricted decryption oracle [LM21].

In the next section, we use randomness to design *extended* arithmetic circuits that are not perfectly correct, but the outputs are still simulatable. But first, we briefly discuss some extended arithmetic circuit-specific optimizations.

8.2.3 Optimizations

We briefly review three optimizations for extended arithmetic circuits that do not apply to arithmetic circuits.

Firstly, one can reintroduce separate operations for the operations defined in Blanton's computational model [BMY24] that we omitted in our computational model. So, one may implement a specific public-multiplication operation that replaces a multiplication that is immediately revealed.

Another optimization is that we can fold randomness similarly to how we can perform constant folding. For example, an addition of two random values can be replaced by one random value, so long as the random values do not have other dependents. When a random value has multiple dependents, replacing it would imply a new random sample while the circuit expected the same random sample. Because of this limitation, we cannot fold randomness in a Beaver triple [Bea91] (a, b, ab) with $a, b \in R$, as this would yield (a, b, c) where $c \neq ab$ with high probability.

A third optimization is that one can use the property that a revealed value may allow one to deduce the value of other intermediate values. We can then reveal these other intermediate values as well, without leaking more information. For example, if $C(a, b) = (a + b, \bot)$, $\mathsf{Known}(a) = \mathcal{P}_1$, and $\mathsf{Known}(b) = \mathcal{P}_2$, then one can choose to reveal value b to $\mathcal{P}_1$ anyways, because the value of b is revealed by $a + b$ through the knowledge of a. Let $C'(a, b) = ((b, a + b), \bot)$ represent the outputs when we choose to reveal value b. To show that this still achieves confidentiality (see Definition 12), we can define simulators $S_1(a, a + b) = (a + b - a, a + b)$ and $S_2(\bot) = \bot$. Choudhary et al. [Cho+23] use this concept in the context of defending against speculative execution attacks by deciding what values do not need protecting. They refer to these values as the 'knowledge frontier'. We argue that a similar pass can be implemented to identify revealable intermediate values.

8.3 Encoding Booleans & sets

In the oraqle compiler, all plaintexts are elements of $\mathbb{F}_p$, and Booleans are represented as the additive identity 0 or the multiplicative identity 1. In this section, we extend the compiler with other secure encodings for Booleans that use the full set of possible elements. For the sake of generality, we also extend the compiler to support different algebras R. Specifically, in this section, we require that R at least satisfies the properties of a commutative ring. This allows us to use EC-ElGamal in Section 8.5 to evaluate resulting circuits.

8.3.1 Definition of secure encodings

In the previous section, we defined the security of compiled circuits. Specifically, we gave a simulation-based definition for when one circuit is said to compute another envisioned circuit confidentially. We now use this definition to define security for encodings. We focus on Boolean encodings but we claim that the definitions can be extended to other types as well.

A Boolean encoding is defined by an invertible map $\phi : \{0,1\} \mapsto R$ and at least one function f' over encoded elements that applies f on non-encoded elements. The idea is that such an encoding makes this computation f more convenient than when using a naive encoding. At the same time, they should not reveal information about the inputs of f' that could not be derived from the output(s) of f. In Chapter 1 we studied such encodings in the context of private set intersections under the name *private homomorphic set representations*. We define correctness as follows:

Definition 13 (Encoding correctness)**.** An operation f' inputting elements encoded by ϕ_{in} and outputting elements encoded by ϕ_{out} correctly realizes function f if there is no polynomial-time algorithm $\mathcal{A}$ that can distinguish the following distributions with non-negligible advantage:

$$\phi_{\text{out}}^{-1}(f'(\phi_{\text{in}}(x_1), \ldots, \phi_{\text{in}}(x_n))) \text{ and } f(x_1, \ldots, x_n) \ .$$

Similarly to the previous section, we use a simulation-based notion of confidentiality. In this definition, we do not require the inverse map to be applied after computing the homomorphism f'; the idea being that, if f is the last operation in the circuit, this decoding operation can be performed in plaintext after performing the rest of the circuit using homomorphic encryption. We do not consider Reveal.

Definition 14 (Encoding confidentiality)**.** An operation f' inputting elements encoded by ϕ is confidential w.r.t. function f if there is no polynomial-time algorithm $\mathcal{A}$ that can distinguish the following distributions with non-negligible advantage:

$$S(f(x_1, \ldots, x_n)) \text{ and } f'(\phi(x_1), \ldots, \phi(x_n)) \ .$$

When an encoding is correct, applying the decoding operation during the circuit (e.g., by arithmetizing it and using homomorphic encryption) or after decryption leads to the correct output(s) of the circuit. If the encoding also satisfies confidentiality, then the resulting circuit also confidentially computes the envisioned circuit as defined in Definition 12.

8.3.2 The negated reduced & unreduced Boolean encodings

We now formalize the negated Boolean encodings that have been implicitly used in many private set intersection protocols (see Chapters 1, 3, and 4) to conveniently compute the set intersection. These encodings exploit the fact that the sum of $k < \text{char}(R)$ Boolean variables $\{0,1\} \in R$ is 0 if and only if all variables were 0. Specifically, we define two encodings. The first encoding is the negated reduced Boolean encoding, which defines the following map that negates a Boolean and lifts it to R:

$$\phi_{\neg\text{red}}(x) = \begin{cases} 1 & \text{If } x = 0 \\ 0 & \text{If } x = 1 \end{cases} \in R \ .$$

The second encoding is the negated *unreduced* Boolean encoding, which defines the following inverse map:

$$\phi_{\neg\text{unr}}^{-1}(y) = \begin{cases} 1 & \text{If } y = 0 \\ 0 & \text{Otherwise} \end{cases} \in \{0,1\} \ .$$

We define the following homomorphic operation f' on $k < \mathrm{char}(R)$ negated reduced Booleans, which outputs a negated unreduced Boolean. Function f' achieves an encoding representing the conjunction of the input Boolean:

$$f'(y_1, \ldots, y_k) = \bar{r} \sum_{i=1}^{k} y_i \ ,$$
$$f(x_1, \ldots, x_k) = \bigwedge_{i=1}^{k} x_i = \phi_{\neg\mathrm{unr}}^{-1}\left(f'(\phi_{\neg\mathrm{red}}(x_1), \ldots, \phi_{\neg\mathrm{red}}(x_k))\right) \ .$$

Here, $\bar{r} \in R^*$ denotes a uniformly random unit of R that must remain unknown to any colluding subset of parties. In other words, $\bar{r}$ is represented by the node $\mathsf{Random}_{U[R^*]}$, where we use $U[R^*]$ to denote the uniform distribution over R^*. It is crucial here that $\bar{r}$ is a unit so that it does not send the sum $\sum_{i=1}^{k}$ to 0.

We provide a short proof of correctness and confidentiality for computing the conjunction using this encoding.

Lemma 16. *Function f' correctly computes the conjunction.*

Proof. If $x_i = 1$ for all $i = 1, \ldots, k$, then $y_i = \phi_{\neg\mathrm{red}}(x_i) = 0$ and $\bar{r} \sum_{i=1}^{k} y_i = 0$. We get that $\phi_{\neg\mathrm{unr}}^{-1}(0) = 1$, which is correct. In all other cases, we have that $\sum_{i=1}^{k} y_i \in R \setminus \{0\}$ because $y_i \in \{0, 1\}$ and $k < \mathrm{char}(R)$, so by the definition of the characteristic, the sum cannot 'wrap around the modulus'. Since $\bar{r} \in R^*$, it cannot send the output of the sum to zero. In other words, $\bar{r} \sum_{i=1}^{k} y_i \in R \setminus \{0\}$ and by the definition of $\phi_{\neg\mathrm{unr}}^{-1}(y)$ the output is 0, which is correct. □

Lemma 17. *Function f' confidentially computes the conjunction.*

Proof. We define the following simulator S and show how its output is identically distributed to the output of $f'(y_1, \ldots, y_k)$:

$$S\left(\bigwedge_{i=1}^{k} x_i\right) = \bar{r}' \cdot \neg \bigwedge_{i=1}^{k} x_i = \bar{r}' \bigvee_{i=1}^{k} y_i = \bar{r}' z \sum_{i=1}^{k} y_i \ ,$$

where $\bar{r}' \in R^*$, is a random unit. Now, there are two cases:

- $\sum_{i=1}^{k} y_i = 0$, so $\bigvee_{i=1}^{k} y_i = 0$ and z and $\bar{r}'$ can take any value for the result to be correct.
- $\sum_{i=1}^{k} y_i \neq 0$, so $\bigvee_{i=1}^{k} y_i = 1$ and $z = (\sum_{i=1}^{k} y_i)^{-1} \in R^*$. Since $\bar{r}'z$ is a random unit in R^*, it is statistically indistinguishable from $\bar{r}$. □

Note that by applying DeMorgan's law, so by negating the inputs and the output, we can define similar encodings that permit a convenient function for computing the disjunction.

8.3.3 Boolean-based set encodings

We briefly discuss Boolean-based set encodings, which may use the encodings described above to encode and intersect sets. Of these three encodings, we implement bitsets and present our initial results in Section 8.5. The bitset encoding allows us to compile a protocol that is essentially the same as the handcrafted bitset-based MPSI presented in Chapter 3.

Bitset

The bitset encoding maps a set to a vector of Booleans that has the length of the universal set $\mathcal{U}$. Specifically, each Boolean pertains to a specific element in the universe, and it is set to 1 if the element is contained in the set. Intersections are then straightforward to compute by performing an AND operation between each corresponding bit between multiple bitsets. In Section 1.4.1, we explain the operation in more detail and we explain how this intersection does not leak information about the elements outside of the intersection.

Bloom filter

In Chapter 2, we show that for a Bloom filter-based (M)PSI protocol to be secure, it must have a negligible false positive rate. A low false positive rate also implies a secure encoding according to the definitions in this chapter, because the Bloom filter representing the intersection equals the Bloom filter after aggregation with high probability. That said, when the false positive rate is set appropriately, this encoding can still be used to compute intersections by computing an AND operation between each corresponding bit between multiple Bloom filters. We describe the high-level structure of such an intersection in Section 1.4.2.

EMQF

The output of an EMQF, as defined in Chapter 4, is a negated unreduced Boolean. By following the high-level construction described in Section 1.4.3, one can also implement intersections on EMQFs using an AND operation.

8.4 Assignment and scheduling

The methods described in the previous section can be used to generate extended arithmetic circuits that compute a desired high-level circuit. These circuits dictate *how* to compute the high-level circuit, but not *who* performs the computations. In this section, we propose techniques for distributing these computations among the involved parties. In other words, we transform the circuit representation into a protocol. We first discuss how to realize the randomness introduced in extended arithmetic circuits. After that, we put forward two formulations for (optimally) assigning and scheduling computations among the parties. The first formulation is a MaxSAT formulation that minimizes the total cost of the protocols as measured by the individual cost of each operation. The second is a MILP formulation that minimizes the total run time.

8.4.1 Realizing randomness

Before we can assign computations to the involved parties, we must define how one can obtain the random samples required in an extended arithmetic circuit. The goal of this step is to reduce the extended arithmetic circuit to one that is realizable under the current security assumptions. Depending on those assumptions, this task can be trivial or not. For example, in the precomputation model with a trusted third party, all randomness can be pre-generated by the trusted third party and provided as encryptions. In fact, such a trusted third party can even pre-generate correlated randomness that may occur in extended arithmetic circuits such as Beaver triples [Bea91], or more generically, arithmetic tuples [Rei+22]. On the other hand, if no trusted third party and precomputations are allowed, realizing a complex random sample such as $\mathsf{Random}_{\mathcal{N}(0,\sigma)}()$, where $\mathcal{N}(0, \sigma^2)$ is the discretized Gaussian distribution with standard deviation σ, is a complicated task.

In this chapter, we do not rely on arbitrary precomputations. Instead, match the assumptions and security properties of the protocols proposed in Chapter 3:

- All computations happen during the protocol.
- There is no trusted third party.
- The collusion threshold is maximal: $t = n - 1$.
- Communication happens strictly in the star topology.
- We do not assume homomorphic multiplications.

Since our goal is to provide initial results, we do not consider realizing arbitrary probability distributions. In the context set intersections based on the encodings proposed in Section 8.3, we are only interested in how to generate uniform samples from R. We briefly discuss three methods of realizing such a sample $\mathsf{Random}_{U[R]}()$ and we consider how these samples may be multiplied with an encrypted value v:

1. If we can perform an arbitrary number of additions and multiplications, we can evaluate pseudo-random number generator prng that outputs elements in R under encryption. As such, one can realize a collusion-resistant random sample by having each of $t + 1$ parties contribute an encrypted random seed. If s_i is the seed of party $\mathcal{P}_i$, the resulting sample is the sum of $\mathsf{prng}(s_i)$. The seeds can be reused for many computations, as long as prng changes at every occasion. Since we can perform an arbitrary number of multiplications, we can homomorphically multiply the resulting sample with v.

2. If we can only perform few multiplications, we can instead have $t + 1$ parties contribute random samples in R. Let r_i denote the random sample of $\mathcal{P}_i$. Then, we can compute the resulting sample as the sum $\sum_{i \in T} r_i$, where T is the set of $t + 1$ parties. We can homomorphically multiply this sample with v.

3. If we cannot perform any multiplications, then we must change the second approach. Instead of computing $v \cdot \sum_{i \in T} r_i$, we can distribute the product such that the multiplications with v become scalar multiplications: $\sum_{i \in T} v \cdot r_i$. As such, the multiplications can be computed using repeated additions. This requires that $r_i \in \mathbb{Z}$.

Since we do not assume homomorphic multiplications, we implement the third method. This allows us to use EC-ElGamal to evaluate the protocol.

Note that the method described above realizes uniform samples on the entire set R, but in the encodings presented in Section 8.3, we required samples in R^*. However, notice that for some R, these distributions are almost exactly the same. Specifically, for some rings, such as $R = \mathbb{Z}_p$ where p is a large prime, we have that R^* is practically the entire set R. A sample $\mathsf{Random}_{U[\mathbb{Z}_p]}()$ from the entire set is only distinguishable from a sample $\mathsf{Random}_{U[\mathbb{Z}_p^*]}()$ of its units if the outcome is the zero element. This occurs with probability p^{-1}. As such, the above method is sufficient to realize the randomness required in the set encodings that we use in this chapter.

8.4.2 Optimizing for efficiency

One optimization objective in secure computation can be to minimize the total cost of all computation and communication. For example, one might consider the cost of running a cloud service, in which computation and communication translates to a monetary value. Another example is that of minimizing the power consumed by the protocol. In this case, we are optimizing for sustainability.

What is interesting about this objective is that if idling is considered free, then we can ignore the concept of time: it does not matter *when* an operation is computed or a message is sent or received. Instead, it matters *who* does *what*. As a result, we can formalize the problem using only Boolean variables, which leads to a natural MaxSAT formulation.

Our formalization of this optimization problem is as follows. Let Nodes denote the set of all operations (nodes) in the circuit. We use $(Z, \mathcal{P}_i) \in \mathsf{Inputs}$ and $(Z, \mathcal{P}_i) \in \mathsf{Outputs}$ to specify the set of pairs describing which party $\mathcal{P}_i$ should input or output Z, respectively. We define the following variables:

- $h_{i,Z}$ is a Boolean indicating whether $\mathcal{P}_i$ has an encryption of Z.
- $c_{i,Z}$ is a Boolean indicating whether $\mathcal{P}_i$ computes Z.
- $s_{i,Z,j}$ is a Boolean indicating whether $\mathcal{P}_i$ sends Z to $\mathcal{P}_j$.

We add the following constraints to this formulation:

C1. At the end of the protocol, $\mathcal{P}_i$ must have an encryption of each of its outputs.

C2. In order to have an encryption of the result of node Z, $\mathcal{P}_i$ must compute it or receive it from another party.

C3. In order to compute node Z, $\mathcal{P}_i$ must have an encryption of its operands.

C4. In order to send an encryption of Z to $\mathcal{P}_j$, $\mathcal{P}_i$ must have it.

C5. An encryption of Z can only be sent in one direction between two parties.

It may seem that this last constraint is merely a cut, but this is not true. Because this formulation does not consider the concept of time, it is perfectly possible for two parties to obtain an encryption of Z from each other 'at the same time', allowing them to send it to the other. The last constraint disallows such a paradox.

Our MaxSAT formulation is as follows:

Hard clauses:

$$\text{C1}\quad (h_{i,Z}),\ \forall (Z,\mathcal{P}_i) \in \text{Outputs}$$

$$\text{C2}\quad \left(\neg h_{i,Z} \vee c_{i,Z} \vee \bigvee_{j=1}^{i-1} s_{j,Z,i} \vee \bigvee_{j=i+1}^{n} s_{j,Z,i}\right),\ \forall Z \in \text{Nodes}, \forall i \in [1,n] : (Z,\mathcal{P}_i) \notin \text{Inputs}$$

$$\text{C3}\quad (\neg c_{i,Z} \vee h_{i,X}),\ \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = X \odot Y$$

$$\text{C3}\quad (\neg c_{i,Z} \vee h_{i,Y}),\ \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = X \odot Y$$

$$\text{C3}\quad (\neg c_{i,Z} \vee h_{i,X}),\ \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = c \odot X$$

$$\text{C4}\quad (\neg s_{i,Z,j} \vee h_{i,Z}),\ \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes}$$

$$\text{C5}\quad (\neg s_{i,Z,j} \vee \neg s_{j,Z,i}).\ \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes}$$

Here, we use $Z = X \odot Y$ to denote any addition or multiplication between X and Y, and $Z = c \odot X$ represents a scalar addition or multiplication.

Additionally, we add a cut to discard a large amount of suboptimal solutions in which a party $\mathcal{P}_i$ obtains an encryption of Z in multiple different ways, whereas one is sufficient. For example, when $s_{i,Z,j} = 1$ (for some $j \neq i$) and $c_{i,Z} = 1$. The cut is to enforce the following constraint:

$$\text{Cut}\quad c_{i,Z} + \sum_{j=1}^{i-1} s_{j,Z,i} + \sum_{j=i+1}^{n} s_{j,Z,i} \leq 1, \quad \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes}\ .$$

A simple way to encode such an at-most-1 constraint $\sum_i x_i \leq 1$ is as pairwise clauses containing all combinations of the sum $(\neg x_1, \neg x_2)$, $(\neg x_1, \neg x_3)$, $(\neg x_2, \neg x_3)$, and so forth. Other methods exist, such as the *ladder* approach by Gent & Nightingale [GN04], which may have more favorable optimization properties.

Next, we define the three functions to describe the individual costs of operations, which allows us to formulate the objective as a set of soft clauses. We use the function $\text{Compute}(i, Z)$ to denote the cost of $\mathcal{P}_i$ computing operation Z, and functions $\text{Send}(i, j)$ and $\text{Receive}(i, j)$ to describe the costs of party $\mathcal{P}_i$ sending or receiving a message to/from $\mathcal{P}_j$. The objective of the MaxSAT formulation is as follows:

Soft clauses:

$$\text{weight: Compute}(i,Z)\quad (\neg c_{i,Z}),\quad \forall i \in [1,n], \forall Z \in \text{Nodes}$$

$$\text{weight: Send}(i,j)\quad (\neg s_{i,Z,j}),\quad \forall i,j \in [1,n] \text{ s.t. } i \neq j, \forall Z \in \text{Nodes}$$

$$\text{weight: Receive}(i,j)\quad (\neg s_{j,Z,i}).\quad \forall i,j \in [1,n] \text{ s.t. } i \neq j, \forall Z \in \text{Nodes}$$

We note that some variables and clauses can be removed. For example, if a certain operation Z is uncomputable by $\mathcal{P}_i$, such as a homomorphic multiplication in the case of additively homomorphic encryption, one can set $c_{i,Z} = 0$ and change the clauses accordingly. The same holds for communication lines that cannot be realized in practice, such as in a star topology.

8.4.3 Optimizing for run time

Another common objective for secure computation protocols is to minimize their run time, from the moment the parties provide their inputs until the moment that the last party receives their output. Unfortunately, this objective cannot efficiently be captured by extending our MaxSAT formulation, because it requires additional variables that track time, which are inefficient to express using Booleans. Instead, we provide a MILP formulation that implements the same constraints as the MaxSAT formulation, while keeping track of time using continuous variables.

One challenge in designing a MILP formulation that minimizes run time, is that some of the required additional constraints are conditional with respect to some of the Boolean variables. For example, the time at which a party has an encryption of the result of operation Z depends on its origin. If the party computes it, it depends on the time at which it has encryptions of all required operands. Alternatively, if it obtains the result of Z by communicating with another party, it depends on the time at which the other party has such an encryption. The straightforward way of encoding these constraints would lead to a quadratically-constrained mixed-integer program, which cannot be optimized by a MILP solver. Instead, we encode these constraints using big-M constraints, allowing for linear constraints.

We note that the objective we use is still a simplified model of the real world. For example, while the objective captures that computations can only start when all their operands are available, it does not model that one party may not complete multiple computations at the same time. In other words, the objective assumes that each party has an infinite number of concurrent threads. When it comes to communication, the objective also does not provide a completely accurate model of the real world. Specifically, it only takes into account a fixed latency modeled by a function $\mathsf{Latency}(i, j)$, which is the time it takes for a message from party $\mathcal{P}_i$ to reach $\mathcal{P}_j$. It does not consider throughput or bandwidth limits, thereby ignoring the actual number of bytes that a ciphertext contains.

Our MILP formulation is essentially an extension of our MaxSAT formulation with more variables and constraints. Next to the Boolean variables defined in our MaxSAT formulation, our MILP formulation define a set of continuous variables $t_{Z,i} \geq 0$, which represent the time since the start of the protocol at which Z is known to party $\mathcal{P}_i$. Another continuous variable $\hat{t}$ represents the time for the entire protocol to finish. We add the following *additional* constraints to this formulation:

C7. The time it takes for $\mathcal{P}_i$ to compute Z is at least as long as the time it takes to obtain its operands plus $\mathsf{Compute}(i, Z)$.

C6. The time it takes for $\mathcal{P}_i$ to receive Z from $\mathcal{P}_j$ is at least as long as the time it takes for $\mathcal{P}_j$ to obtain Z plus $\mathsf{Latency}(j, i)$.

C8. The time it takes to finish the protocol $\hat{t}$ is at least as long as the time it takes to obtain its outputs $t_{i,Z}$ for $(Z, \mathcal{P}_i) \in \mathsf{Inputs}$.

Finally, to ensure that our MILP formulation can find the optimal solution, we must ensure that our big-M constraints are valid. The big-M constraints ensure that constraint C6 is only enforced when the value is actually computed, and that C7 is only enforced when a value has been sent. If a message is not computed or

sent, we subtract M from the time at which the value becomes available. As such, we want M to be as least as large as the run time of the optimal protocol. One way to choose this value is to compute the run time of any naive protocol. For example, in the case of fully homomorphic encryption, one may assign all computations to the party that performs computations the fastest. Our formulation is as follows:

Minimize $\hat{t}$ such that:

$$
\begin{aligned}
h_{i,Z}, c_{i,Z} &\in \{0,1\} \\
s_{i,Z,j} &\in \{0,1\} \\
t_{i,Z} &\in \mathbb{R}^+ \\
\hat{t} &\in \mathbb{R}^+
\end{aligned}
$$

$$
\begin{aligned}
&\text{C1} && h_{i,Z} = 1 && \forall (Z, \mathcal{P}_i) \in \text{Outputs} \\
&\text{C2} && c_{i,Z} + \textstyle\sum_{j=1}^{i-1} s_{j,Z,i} + \sum_{j=i+1}^{n} s_{i,Z,j} \geq h_{i,Z} && \forall Z \in \text{Nodes}, \forall i \in [1,n] : (Z, \mathcal{P}_i) \notin \text{Inputs} \\
&\text{C3} && c_{i,Z} \leq h_{i,X} && \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = X \odot Y \\
&\text{C3} && c_{i,Z} \leq h_{i,Y} && \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = X \odot Y \\
&\text{C3} && c_{i,Z} \leq h_{i,X} && \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = c \odot X \\
&\text{C4} && s_{i,Z,j} \leq h_{i,Z} && \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes} \\
&\text{C5} && s_{i,Z,j} + s_{j,Z,i} \leq 1 && \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes} \\
&\text{Cut} && c_{i,Z} + \textstyle\sum_{j=1}^{i-1} s_{j,Z,i} + \sum_{j=i+1}^{n} s_{i,Z,j} \leq 1 && \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes} \\
&\text{C6} && t_{i,X} + \text{Compute}(i,Z) - (\neg c_{i,Z}) \cdot M \leq t_{i,Z} && \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = X \odot Y \\
&\text{C6} && t_{i,Y} + \text{Compute}(i,Z) - (\neg c_{i,Z}) \cdot M \leq t_{i,Z} && \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = X \odot Y \\
&\text{C6} && t_{i,X} + \text{Compute}(i,Z) - (\neg c_{i,Z}) \cdot M \leq t_{i,Z} && \forall i \in [1,n], \forall Z \in \text{Nodes} : Z = c \odot X \\
&\text{C7} && t_{j,Z} + \text{Latency}(j,i) - (\neg s_{j,Z,i}) \cdot M \leq t_{i,Z} && \forall i,j \in [1,n] : i \neq j, \forall Z \in \text{Nodes} \\
&\text{C8} && \hat{t} \geq t_{i,Z} && \forall (Z, \mathcal{P}_i) \in \text{Outputs}
\end{aligned}
$$

8.5 Initial results

We present initial results that demonstrate that the extended oraqle compiler can automatically design a protocol that is essentially the bitset-based MPSI presented in Chapter 3. To obtain these results, we implemented the Boolean encodings and the bitset representation as explained in Section 8.3. We also implemented the method for realizing randomness that does not require homomorphic multiplications and the MaxSAT formulation for scheduling & assigning operations in the circuit, as presented in the previous section. In this section, we first explain how the compiler transforms a high-level set intersection circuit into an extended arithmetic circuit. After that, we present two protocols designed by the compiler. The first protocol is essentially Protocol 3 from Chapter 3. The second protocol is a variant designed for the scenario in which the leader's computations are an order of magnitude more expensive than those of other parties.

8.5.1 From a bitset intersection to an extended arithmetic circuit

We specify the high-level circuit of an MPSI protocol by its definition $f(X_1, \ldots, X_n) = X_1 \cap \cdots \cap X_n$ with $\mathcal{U} = \{1, \ldots, 10\}$. We must also define knowledge: $\text{Known}(X_i) = \{i\}$ implies that input sets are only known by their respective party, and $\text{Known}(X_1 \cap \cdots \cap X_n) = \{1\}$ marks that only the leader, party $\mathcal{P}_1$, obtains the output. Finally,

we must choose the plaintext algebra. Since we want to use elliptic curve-based ElGamal (EC-ElGamal, in Table 8.1), we specify it to be the ring of integers modulo q, where q is the order of the prime-order elliptic curve that we also used in Chapter 3. Compilation then happens in the following steps:

1. The only set encoding we implemented is the bitset encoding, so we encode the inputs as such: $X_1 \cap \cdots \cap X_n = \mathsf{Bitset}(X_1) \cap \cdots \cap \mathsf{Bitset}(X_n)$.
2. Extended arithmetization of the bitset intersection is simply the bit-wise conjunction of the bit vectors: $\mathsf{Bitset}(X_1) \cap \cdots \cap \mathsf{Bitset}(X_n) = \mathsf{Bitset}(X_1) \wedge \cdots \wedge \mathsf{Bitset}(X_n)$.
3. The compiler performs extended arithmetization of the conjunctions. It tries all the possible Boolean encodings and chooses the circuit with the lowest number of multiplications. Currently, it has two Boolean encodings that support conjunctions:
   - The compiler encodes the Booleans as reduced Booleans and applies the arithmetization of AND operations as described in Chapter 5. This leads to a circuit with multiplications, as these pure arithmetic circuits do not contain randomness.
   - The compiler then encodes the Booleans as reduced negated Booleans and applies the extended arithmetization of the conjunction as described in Section 8.3. This leads to a circuit without multiplications.

   It chooses the second encoding of negated reduced Booleans.

The result of this compilation stage is an extended arithmetic circuit with $|\mathcal{U}|$ outputs; one bit for each element in the universal set. Each bit is computed in the same way, using the conjunction operation as defiend in Section 8.3.

8.5.2 From the extended arithmetic circuit to an MPSI protocol

In the next stage, assignment & scheduling, the user must specify some high-level properties of the homomorphic encryption scheme. Specifically, it must indicate how many homomorphic multiplications the cryptosystem supports. The user must also specify the computational and communication characteristics of the entire set of parties, so as to parameterize the objective function. In doing so, the user also specifies the number of parties n, and the collusion threshold t. In our implementation, we only support $n = t - 1$.

We specify the following parameters to be in line with the protocols presented in Chapter 3: Since we want to use EC-ElGamal, we indicate that the cryptosystem does not support any multiplications. When it comes to communication, we assume a star topology, so we specify that the cost of $\mathcal{P}_i$ sending $\mathcal{P}_j$ a message for $i, j \in [2, n]$ to infinity. We set the other parameters as follows: An addition costs 1 unit, a scalar multiplications costs 100 units, and sending a message costs 1,000 units. To match Chapter 3, which centralizes computations around central server $\mathcal{P}_1$, we make the computational costs of the other parties 10x higher. As a minimal example, we set the number of parties to $n = 3$. The compiler then proceeds with the following steps:

4. The compiler determines how to realize the random samples in $\mathbb{Z}_q^*$. It asserts that q^{-1} is negligible (it does not exceed 2^{-128}), so it can realize the random samples as uniform samples in $\mathbb{Z}_q$. It uses the technique for the setting with no homomorphic multiplications as described in Section 8.4.1. This introduces three random inputs, say r_i for $i \in [1,3]$ where $\mathsf{Known}(r_i) = \{i\}$.

5. At this point, the comiler must assign & schedule the computations among the involved parties. It uses the MaxSAT formulation described in Section 8.4.2 to generate the most efficient protocol.

We present the resulting protocol for the first bit of the bitset in Figure 8.2 (the protocol is the same for each other bit). In this circuit, blue nodes indicate inputs, purple nodes indicate random inputs, and gray nodes represent homomorphic operations that do not require non-scalar multiplications to compute. Notice that the inputs are negated because they use the negated reduced Boolean encoding. We indicate the set of computations performed by each party by clustering these nodes together in a dotted rectangle. This protocol is almost identical to Protocol 3 from Chapter 3. The only differences are that Protocol 3 also considers point compression and that the inputs are encoded slightly differently; it uses the negated unreduced encoding multiplied by randomness. The two protocols have the same cost when measured using the objective of the MaxSAT formulation. It took less than ten seconds for the MaxSAT solver to construct the optimal protocol on a laptop with an M1 chip. We used the same MaxSAT solver as for performing arithmetization (see Chapter 5).

8.5.3 A variant of the MPSI protocol for another scenario

One benefit of separating the extended arithmetization phase from the assignment & scheduling phase is that one can distribute the computations in a protocol differently depending on the parties' properties. To demonstrate this, we consider the setting in which the leader's computations are ten times more expensive than the operations of the other parties. The compiler now outputs a different optimal protocol that lets the other parties perform more work. We present the resulting protocol in Figure 8.3. Notice that the protocol is still in the star topology, but that there is communication between parties $\mathcal{P}_2$ and $\mathcal{P}_3$ that is routed through the leader $\mathcal{P}_1$. While this would extend the run time of the protocol, it reduces its computational cost. It also took the compiler less than ten seconds to construct this optimal protocol.

8.6 Conclusion

In this chapter, we provided proof of concept extension of the oraqle compiler that can generate MPSI protocols from a high-level description. While these protocols are conceptually simple enough to be designed by hand, an algorithmic approach does not require a user to have the same expertise. Next to that, the extended oraqle compiler provides optimality guarantees that we cannot easily work out by hand. What is more, we can generate a scenario-specific protocols automatically based

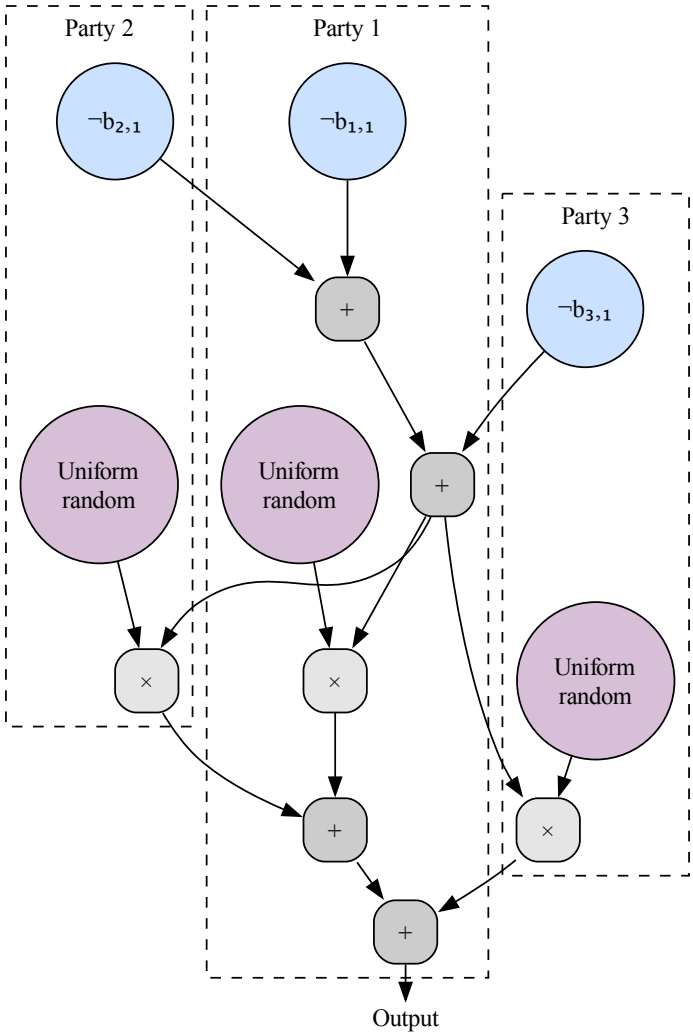


Figure 8.2: The optimal protocol as computed by our MaxSAT formulation when computing an addition costs 1 unit, a scalar multiplication costs 100 units, sending a message costs 1,000 units, and the computations of parties 2 and 3 are 10x more expensive. This protocol achieves a cost of 14,104 units. This protocol is almost identical to Protocol 1 as proposed in Chapter 3, except that its inputs are the identity 0 or the unit 1, whereas Protocol 1 encrypts 0 or randomness.

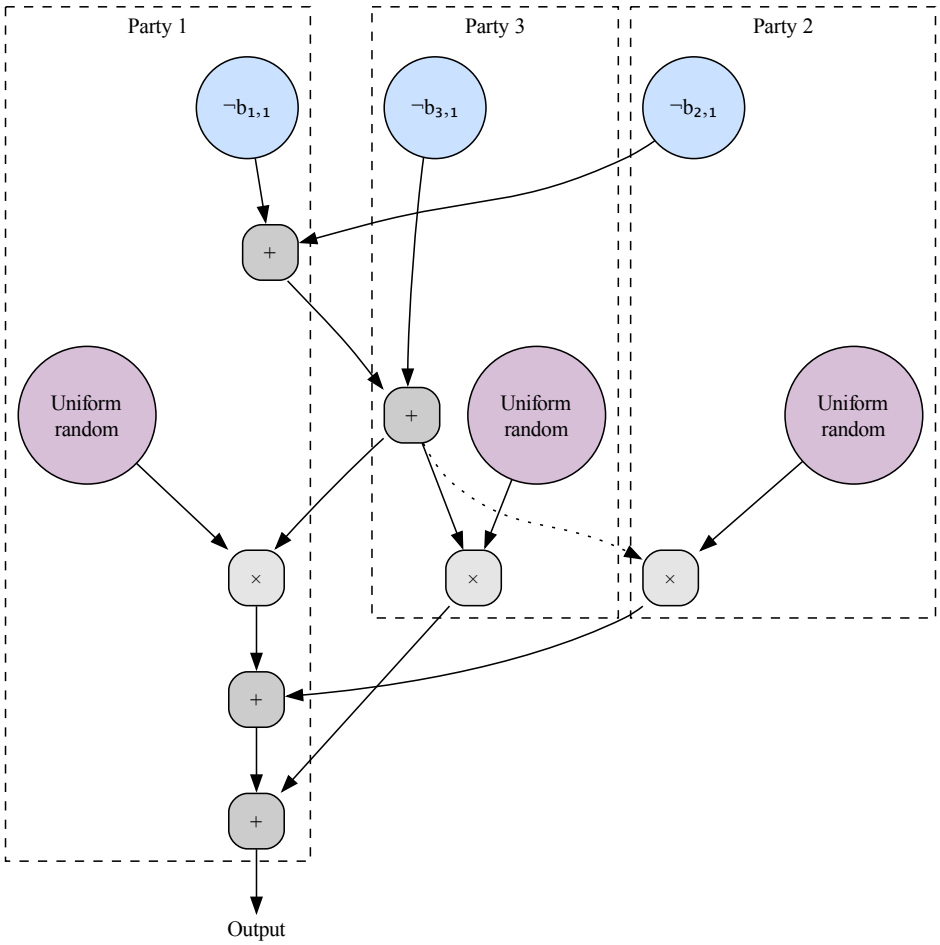


Figure 8.3: The optimal protocol as computed by our MaxSAT formulation when computing an addition costs 1 unit, a scalar multiplication costs 100 units, sending a message costs 1,000 units, and the computations of parties 2 and 3 are 10x **cheaper**. This protocol achieves a cost of 12,123.1 units. The dotted arrow between party 3 and party 2 indicates that the communication is actually routed through party 1. Notice that this is not wasteful, as party 1 also uses the result of the addition to perform its multiplication. Moreover, the objective is to minimize cost rather than run time, so routing the message through party 1 does not oppose the objective.

on the computational and communication costs. We discuss several limitations to our approach and future work, describing the steps required for the compiler to generate more complex protocols and to do so more efficiently. We conclude that, while this proof of concept provides promising results towards a generic secure computation compiler, there are still limitations that may require different approaches to address and there is a significant amount of implementation work left to support other homomorphic encryption schemes and computational tasks.

Limitations

Firstly, the computational model of extended arithmetic circuits does not allow one to express hybrid (or mixed) protocols that use multiple secure computation techniques. Hybrid protocols can achieve better performance than protocols relying on one secure computation technique. The computational model also does not cover more complex techniques such as some homomorphic secret sharing techniques [Boy+18] and function secret sharing [BGI15].

Another limitation that limits the efficiency of protocols is that the order of operations in an extended arithmetic circuit can have a drastic impact on the performance of the optimal protocol. For example, the sum $(a + b) + c$ may require fewer communication than the sum $a + (b + c)$, even though these are equivalent. Since extended arithmetization and assignment & scheduling happen in isolation, we cannot alleviate this problem in our current setup.

One limitation that makes the compiler significantly less practical is that assignment & scheduling may take very long for large circuits and many parties. In these cases, it may be sufficient to find a protocol that is 'good enough', rather than one that is optimal. As such, one may consider using an incremental weighted MaxSAT solver and terminating its search when it obtains a good (but not necessarily optimal) solution or when the search duration exceeds a predetermined limit. Note that, in some cases, the solution at termination is also the optimal solution, but it cannot be proven to be so.

Finally, as mentioned before, our model for describing the cost of a protocol does not always accurately describe the real world. For example, real computers have a specified number of concurrent threads and networks have a limited bandwidth and throughput. Moreover, the computational cost of homomorphic encryption operations is not always constant: in schemes such as BGV, the cost of operations becomes slightly cheaper after a multiplication. In the BGN cryptosystem, homomorphic additions also change after a multiplication.

Future work

An obvious next step for the extended compiler is to implement code generation, such that the protocols do not have to be implemented by hand. Ideally, non-experts could provide the compiler with a list of secure extended arithmetic circuit evaluation techniques and the cost of their operations so that the compiler can automatically choose which secure computation technique leads to the most performant protocol. One starting point is to implement the homomorphic encryption schemes listed in Table 8.1.

Another direction is to extend the compiler to support secret sharing. The key difference between the resulting protocols is that operations in homomorphic encryption-based protocols do not have to be performed by more than one party, while in secret sharing-based protocols, it is common for many parties to be involved in each computation. This would require taking into account that computations must be done by multiple parties, and that multiple shares must not be sent to the same party as that reduces the collusion threshold.

In the context of multi-party private set intersections, one could implement the other constructions presented in Chapter 1. Next to that, for the compiler to be useful in contexts other than set operations, it must support more high-level types and encodings. For example, in many programming languages it is common to work with 64-bit values, not with elements in $\mathbb{Z}_q$ where q is a small or large prime.

Finally, one can extend the compiler to realize other probability distributions. This may prove useful in distributed key generation protocols, among others.

References

[Aly+25] Abdelrahaman Aly et al. *SCALE-MAMBA: A framework for practical secure computation*. https://github.com/KULeuven-COSIC/SCALE-MAMBA. Accessed: 2025-01-28. 2025.

[Bea91] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 420–432. DOI: 10.1007/3-540-46766-1_34. URL: https://doi.org/10.1007/3-540-46766-1%5C_34.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. "Function Secret Sharing". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Springer, 2015, pp. 337–367. DOI: 10.1007/978-3-662-46803-6_12. URL: https://doi.org/10.1007/978-3-662-46803-6%5C_12.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. Ed. by Joe Kilian. Vol. 3378. Lecture Notes in Computer Science. Springer, 2005, pp. 325–341. DOI: 10.1007/978-3-540-30576-7_18. URL: https://doi.org/10.1007/978-3-540-30576-7%5C_18.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. "Fully Homomorphic Encryption without Bootstrapping". In: *Electron. Colloquium Comput. Complex.* TR11-111 (2011). ECCC: TR11-111. URL: https://eccc.weizmann.ac.il/report/2011/111.

[BLO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. "Optimizing Semi-Honest Secure Multiparty Computation for the Internet". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 578–590. DOI: 10.1145/2976749.2978347. URL: https://doi.org/10.1145/2976749.2978347.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. "Sharemind: A Framework for Fast Privacy-Preserving Computations". In: *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*. Ed. by Sushil Jajodia and Javier López. Vol. 5283. Lecture Notes in Computer Science. Springer, 2008, pp. 192–206. DOI: 10.1007/978-3-540-88313-5_13. URL: https://doi.org/10.1007/978-3-540-88313-5%5C_13.

[BMY24] Marina Blanton, Dennis Murphy, and Chen Yuan. "Efficiently Compiling Secure Computation Protocols From Passive to Active Security: Beyond Arithmetic Circuits". In: *Proc. Priv. Enhancing Technol.* 2024.1 (2024), pp. 74–97. DOI: 10.56553/POPETS-2024-0006. URL: https://doi.org/10.56553/popets-2024-0006.

[Boy+18] Elette Boyle et al. "Foundations of Homomorphic Secret Sharing". In: *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*. Ed. by Anna R. Karlin. Vol. 94. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 21:1–21:21. DOI: 10.4230/LIPICS.ITCS.2018.21. URL: https://doi.org/10.4230/LIPIcs.ITCS.2018.21.

[Bra12] Zvika Brakerski. "Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP". In: *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, 2012, pp. 868–886. DOI: 10.1007/978-3-642-32009-5_50. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_50.

[Büs+18] Niklas Büscher et al. "HyCC: Compilation of Hybrid Protocols for Practical Secure Computation". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie et al. ACM, 2018, pp. 847–861. DOI: 10.1145/3243734.3243786. URL: https://doi.org/10.1145/3243734.3243786.

[Che+17] Jung Hee Cheon et al. "Homomorphic Encryption for Arithmetic of Approximate Numbers". In: *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December*

3-7, 2017, Proceedings, Part I. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. Lecture Notes in Computer Science. Springer, 2017, pp. 409–437. DOI: 10.1007/978-3-319-70694-8_15. URL: https://doi.org/10.1007/978-3-319-70694-8%5C_15.

[Cho+23] Rutvik Choudhary et al. "Declassiflow: A Static Analysis for Modeling Non-Speculative Knowledge to Relax Speculative Execution Security Measures". In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*. Ed. by Weizhi Meng et al. ACM, 2023, pp. 2053–2067. DOI: 10.1145/3576915.3623065. URL: https://doi.org/10.1145/3576915.3623065.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation". In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015. URL: https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation.

[FV12] Junfeng Fan and Frederik Vercauteren. "Somewhat Practical Fully Homomorphic Encryption". In: *IACR Cryptol. ePrint Arch.* (2012), p. 144. URL: http://eprint.iacr.org/2012/144.

[Gam84] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2. URL: https://doi.org/10.1007/3-540-39568-7%5C_2.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 218–229. DOI: 10.1145/28395.28420. URL: https://doi.org/10.1145/28395.28420.

[GN04] Ian P Gent and Peter Nightingale. "A new encoding of alldifferent into SAT". In: *International Workshop on Modelling and Reformulating Constraint Satisfaction*. Vol. 3. 2004, pp. 95–110.

[Has+19] Marcella Hastings et al. "SoK: General Purpose Compilers for Secure Multi-Party Computation". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1220–1237. DOI: 10.1109/SP.2019.00028. URL: https://doi.org/10.1109/SP.2019.00028.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by Jay Ligatti et al. ACM, 2020, pp. 1575–1590. DOI: 10.1145/3372297.3417872. URL: https://doi.org/10.1145/3372297.3417872.

[LM21] Baiyu Li and Daniele Micciancio. "On the Security of Homomorphic Encryption on Approximate Numbers". In: *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12696. Lecture Notes in Computer Science. Springer, 2021, pp. 648–677. DOI: 10.1007/978-3-030-77870-5_23. URL: https://doi.org/10.1007/978-3-030-77870-5%5C_23.

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16. URL: https://doi.org/10.1007/3-540-48910-X%5C_16.

[Rei+22] Pascal Reisert et al. "Arithmetic Tuples for MPC". In: *IACR Cryptol. ePrint Arch.* (2022), p. 667. URL: https://eprint.iacr.org/2022/667.

[RHH14] Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. "Wysteria: A Programming Language for Generic, Mixed-Mode Multiparty Computations". In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 655–670. DOI: 10.1109/SP.2014.48. URL: https://doi.org/10.1109/SP.2014.48.

[Ria+19] M. Sadegh Riazi et al. "MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation". In: *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2019, McLean, VA, USA, May 5-10, 2019*. IEEE, 2019, pp. 198–207. DOI: 10.1109/HST.2019.8740831. URL: https://doi.org/10.1109/HST.2019.8740831.

[SV14] Nigel P. Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations". In: *Des. Codes Cryptogr.* 71.1 (2014), pp. 57–81. DOI: 10.1007/S10623-012-9720-4. URL: https://doi.org/10.1007/s10623-012-9720-4.

[ZSB13] Yihua Zhang, Aaron Steele, and Marina Blanton. "PICCO: a general-purpose compiler for private distributed computation". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM, 2013, pp. 813–826. DOI: 10.1145/2508859.2516752. URL: https://doi.org/10.1145/2508859.2516752.

Discussion

The title of this thesis being 'Practical Secure Computation in the Client-Server Model' implies that there are also impracticalities related to secure computation. Indeed, we formulated four impracticalities in the introduction, with the ultimate goal of identifying and implementing ways to address them. Specifically, we formulated the following impracticalities:

- Impracticality 1: High interactivity, secure computation protocols can require many communication rounds.
- Impracticality 2: Full-mesh topology, secure computation protocols can require all parties to communicate with each other.
- Impracticality 3: Arithmetization, secure computation protocols can require programmers to express high-level circuits as arithmetic circuits by hand.
- Impracticality 4: Compute-intensive, secure computation protocols can require significant computational effort.

We addressed impracticalities 1 & 2 in the context of multi-party computation, and 3 & 4 in the context of fully-homomorphic encryption. In Chapter 8 we provide initial steps to combine these approaches. Looking back at the entire thesis, we discuss our main findings, its societal impact, as well as its limitations and future directions.

Takeaways

In addressing the four major impracticalities as defined in the introduction of this thesis, we have learned several new facts. We do not reiterate all separate conclusions from the chapters in this thesis. Instead, we discuss the most prominent takeaways in the order in which these insights appear in the thesis.

Part A: Efficient protocols for MPSO in the star topology

In Part A, we focused on the design and analysis of multi-party private set operations that address impracticalities 1: High interactivity and 2: Full-mesh topology. We provided a new systematization for MPSI protocols and analyzed the security of Bloom filters in realizing secure MPSI protocols. We also proposed multiple low-round protocols in the star topology for private set operations, including membership queries, intersections, and unions. Our research led to the following insights.

There are three high-level constructions that all MPSI protocols follow MPSI protocol roughly consist of three stages: some form of *aggregation*, membership *queries*, and a phase in which values that were cryptographically hidden are *revealed*. In Chapter 1, we found that MPSI protocols can be classified into three categories that each has a fixed order in which these phases occur:

- Protocols based on private homomorphic set representations first aggregate the representations, after which they reveal the result and query it in plain text.
- Protocols based on leaky homomorphic set representations also first aggregate the representations, but they query under encryption before revealing the results because the representation leaks information.
- Protocols based on aggregatable membership queries first query the separate sets, aggregate these results by computing an AND operation, and revealing the results.

It stands to reason that revealing cannot occur before aggregation, because that would leak information about the individual sets. As a result, these are the only three categories that occur.

Approximate Bloom filters cannot be used to realize secure (M)PSI protocols In Chapter 2, we showed that the approximate nature of Bloom filters leads not only to a non-negligible probability of incorrectness, it leads to both theoretical and practical attacks with non-negligible attack success rates. That said, Bloom filters are still convenient to design private set operation protocols around, but the Bloom filters must be large enough to make the probability of false positives occurring negligibly small.

We achieve efficient low-round private set operation protocols in the star topology In Chapter 3, we proposed multiple private set operation protocols. Specifically, we proposed multi-party private set intersection and union protocols for small and large universes. In Chapter 4, we proposed protocols for performing multiple sequential privacy-preserving membership queries on the same set. Our multi-party private set union protocol for large universes allows trading off the number of rounds and the computational effort required. The other protocols have a constant round complexity. All our protocols are strictly in the star topology and they achieve computational efficiency by using elliptic curve-based homomorphic encryption.

Part B: Automatic generation of HE circuits

In Part B of this thesis, we addressed impracticalities 3: Arithmetization and 4: Compute-intensive by proposing several methods for generating computationally-efficient circuits that can be evaluated using homomorphic encryption from high-level representations. The main product of this research is the oraqle compiler, which arithmetizes circuits composed of several common operations in seconds

or minutes. Since the oraqle compiler produces arithmetic circuits, which differ from the Boolean or LUT circuits generated by other compilers, we demonstrate speedups for several circuits. Next to that, in Chapter 7, we considered circuits beyond arithmetic circuits, that use the polynomial ring structure of ring learning with errors-based cryptosystems to permute and map values across packed ciphertexts. We briefly summarize some of the learnings that emerged from this research.

Depth-aware arithmetization produces better circuits than single-objective methods In Chapter 5, we proposed depth-aware arithmetization: a multi-objective version of arithmetization that minimizes for both the multiplicative depth and size (or cost) of an arithmetic circuit. By considering multiplicative depth during arithmetization, instead of as an additional optimization afterwards, we obtain circuits that could not by generated by focusing on either the multiplicative depth or size. Table 5.1 showcases that depth-aware arithmetization can generate comparison circuits that are smaller in size than the work by Iliashenko & Zucca that focuses on minimizing the multiplicative size, and as shallow as the work by Gouert et al. that focuses on minimizing the multiplicative depth with a significantly lower multiplicative size.

The amortized computational cost of BGV can be lower than TFHE The TFHE homomorphic encryption scheme essentially computes LUT circuits: circuits comprised of additions and look-up tables. These circuits are well-suited to non-linear operations, because the cost of evaluating a look-up table does not depend on the actual operation. One may expect that this makes TFHE more suitable for evaluating complex circuits as compared to schemes like BGV, which evaluate arithmetic circuits that contains additions and multiplications. In Chapter 5, we show that the computational cost of evaluating a complex high-level circuit consisting of comparisons and AND operations using BGV is comparable to that of TFHE when accounting for the fact that the BGV scheme permits ciphertext packing. A key aspect is that the circuit is shallow enough to be computed without any bootstrapping operations.

Arithmetic circuits for complex operations can outperform Boolean circuits Previous homomorphic encryption compilers for HE schemes like BGV typically generate Boolean circuits, even though these schemes are capable of evaluating arithmetic circuits beyond $\mathbb{Z}_2$. In Chapter 6, we demonstrate that the arithmetic circuits generated by the oraqle compiler lead to a decrease in computational effort when compared to Boolean circuits, even for complex non-arithmetic operations. Specifically, Table 6.2 shows that a circuit over $\mathbb{Z}_5$ is almost twice as efficient as a Boolean circuit for computing an equality check between two 64-bit sequences.

Rotation blocks outperform Beneš' shift-networks on random permutations In Chapter 7, we propose a method based on so called 'conveyor belts' to automatically generate shift-networks for permuting packed HE ciphertexts. These shift-networks, consisting of rotations, masking operations, and additions, have

been studied and implemented before in the HElib library. While our method was designed with the goal of optimizing shift-networks for permutations across ciphetexts, we show that our circuits can outperform the collapsed Beneš networks implemented in HElib, even for within-ciphertext permutations, see Figure 7.4.

The road ahead

In Parts A & B we addressed the four major impracticalities in isolation, but in the final chapter of this thesis, Chapter 8, we aimed at addressing these impracticalities collectively. The main product of this research is an extension of the oraqle compiler that is general enough to include the generation of MPC protocols. In other words, the compiler has a concept of parties, and may decide which parties to assign certain computations: the compiler generates protocols rather than circuits. Moreover, the compiler considers operations beyond arithmetic: it enables the use of randomness and it selectively allows for parties to reveal plaintexts in the middle of a circuit. We discuss our three main takeaways.

Given an extended arithmetic circuit, we can generate an optimal protocol In Chapter 8, we propose a MaxSAT and a mixed-integer linear programming formulation for assigning and scheduling the computations of an extended arithmetic circuit among multiple parties. These formulations can be solved *to optimality* by existing MaxSAT and MILP solvers. The two formulations optimize for different objectives: the MaxSAT formulation minimizes the total cost of the protocol, e.g. in terms of the energy required or the amount of cloud credits that it consumes. The MILP formulation minimizes the total run time of the protocol, provided that it knows how long it takes for each party to perform each possible operation, as well as the latency on each communication line.

Existing MPSI protocols can be generated from a high-level description Chapter 8 connects parts A and B of this thesis by showing that the extended oraqle compiler can generate what is essentially the bitset-based MPSI protocol as proposed in Chapter 3. After specifying that the each party contributes a bitset, that the plaintext algebra is the group $\mathbb{Z}_p$, and that the protocol should output the intersection, the compiler automatically generates this MPSI protocol. Moreover, depending on the cost of each operation, the resulting protocol may differ from the protocol proposed in Chapter 3 if the MaxSAT formulation decides that it is cheaper to distribute the computations differently.

The extended oraqle compiler generalizes MPC in the star topology and SOC The extended oraqle compiler allows specifying arbitrary costs, so if the goal is to completely outsource computations in a two-party setting to a server, one may specify that the computations are infinitely expensive when performed by other parties. In a secure multi-party computation setting, one can use the communication costs to encode arbitrary communication structures. For example, one might encode a star topology by specifying communication between two clients to be infinitely expensive.

Societal impact

The aim of this thesis is to make secure computation more practical. We briefly review the societal impact we achieve in doing so.

Secure multi-party computation Secure multi-party computation has the potential to distill knowledge from information sources that could previously not be joined, whether for regulatory or ethical reasons. We mentioned several examples in the introduction of this thesis. In situations where users' private data are already combined but in a way that requires placing trust in the aggregator, the confidentiality achieved by MPC can improve the privacy of these users. In a way, our work on MPC has already had some societal impact: the United States and the United Kingdom collaborated to organize a challenge in privacy enhancing technologies (PETs), and our solution that we proposed in Chapter 4 was awarded the second prize. As a result, the challenge and the three top teams were briefly presented at the Summit for Democracy in 2024 for the president of the United States, with the aim of underlining the ability of PETs to reinforce democratic values.

Secure outsourced computation Secure outsourced computation allows parties to rely on cloud computing without the inherent trust assumption. This may allow medical analyses to be outsourced in a way that was not previously legally permitted, while protecting user's sensitive data under cryptographic hardness assumption. In other words, SOC may be used to limit the spread of private information to only those parties that need to see it. An important first step here is to democratize the technology that is homomorphic encryption, so that it may be deployed by the organizations that offer cloud computing and understood by non-experts. In this aspect, our work has already achieved a modest societal impact: at the time of writing, our work on ciphertext permutations (see Chapter 7) is being adapted in Google's HEIR compiler.[1]

The road ahead We believe that the extended oraqle compiler paves the road ahead to a future in which laypersons and experienced researchers alike can generate efficient and secure protocol in the client-server model to achieve better privacy or distill knowledge from information silos.

Limitations

We discuss six general limitations to the research presented in this thesis.

Private set operations do not necessarily preserve privacy By their definition, private set operations achieve confidentiality. Whether they provide privacy depends on the context in which they are deployed. Here we consider control to be a fundamental property of privacy: an individual should personally determine

[1]The implementation effort is tracked in: https://github.com/google/heir/issues/919

the extent in which their personal information is shared. To see that it is possible that these protocols do not preserve privacy, consider the use of an MPSI protocol between multiple organizations on private user data, without giving users the option to opt out of this computation. At the end of the protocol, if an intersection exists, the protocol may reveal private user data that those users did not agree to release.

Quantum algorithms break elliptic curve-based homomorphic encryption All the realizations of the protocols that we presented in Part A of this thesis rely on the security of elliptic curve-based homomorphic encryption. However, quantum algorithms, such as Shor's algorithm, are known to break the IND-CPA security of these schemes in polynomial time. It stands to reason that these protocols should be deployed with caution, and that it may be advisable to deploy them using post-quantum homomorphic encryption. For example, the lattice-based homomorphic encryption methods described in Part B of thesis are conjectured to withstand quantum attacks when suitably parameterized.

Our optimality guarantees only apply to simple operations In Chapter 5, we provide optimality guarantees for exponentiation and equality circuits, but we cannot guarantee optimality for more complex operations such as comparisons or AND and OR operations. Even circuits composed of optimal sub-circuits are not optimal: for example, checking whether the result of an exponentiation equals 0, e.g. $x^5 = 0$, in a finite field is equivalent to checking $x = 0$. In other words, the sub-circuit that computes x^5 is unnecessary, increasing the multiplicative size beyond optimality.

Optimality is only guaranteed with respect to simplified models We provide optimality guarantees in two ways in this thesis. First, in Chapter 5, we show how to generate circuits that optimally trade off multiplicative size and depth. However, the multiplicative size and depth do not capture all computational aspects in homomorphic encryption. For example, the noisier a ciphertext is, the smaller we can represent it, which in turn decreases computational effort. This means that multiplications at the start of a circuit are more expensive than later multiplication, which is not captured by the multiplicative size metric. Next to that, these metrics do not consider automorphism operations, which permit ciphertext permutations, among others. The multiplicative depth also does not fully describe the noise growth in homomorphic encryption circuits: a circuit multiplying three ciphertexts is less noisy than a circuit multiplying four ciphertexts, even though may have the same multiplicative depth. So, we cannot conclude that these are the least computationally-expensive circuits in practice. Second, in Chapter 8, we provide optimality guarantees for the assignment and scheduling of extended arithmetic circuits among parties. However, this model abstracts away several practical aspects. For example, the MILP formulation assumes that parties have an infinite number of threads and that communication lines have an infinite bandwidth. The MaxSAT formulation assumes that idling has no cost, but in practice, a waiting computer still carries a cost.

Extending our protocols to withstand malicious adversaries is non-trivial Whereas some protocols that withstand honest-but-curious adversaries can easily be transformed to withstand malicious adversaries by instantiating them with maliciously-secure primitives, this hard to achieve for the protocols proposed in this thesis. The reason is that our protocols, which are based on homomorphic encryption, rely on the honest-but-curious assumption to ensure that parties do not perform arbitrary computations on the ciphertexts. For elliptic curve-based ElGamal ciphertexts there exist constructions for zero-knowledge proofs, but for fully homomorphic encryption the overhead would be significant.

Centralized secure computation does not ensure availability The protocols that we propose in this thesis achieve confidentiality, but the do not necessarily ensure availability. Especially considering that our protocols revolve around a powerful centralized party, this party essentially controls whether a protocol execution succeeds. Note that even though the honest-but-curious model expects such a party to follow the protocol faithfully, they can in theory delay it indefinitely. In other words, while the client-server model is a practical computational model, an ill-intentioned central parties may at any point decide to selectively deny service.

Future work

In Chapter 8 we described the road ahead for unifying the research in Parts A and B with the idea that this would allow us to address all four major impracticalities that we identified earlier. Unfortunately, there are remaining challenges. We briefly discuss areas in which our current work falls short at addressing the impracticalities, and propose several future directions.

Two-round secure computation protocols in the star topology are already achievable through fully-homomorphic encryption, addressing impracticalities 1: High interactivity and 2: Full-mesh topology. Next to that, through the extended oraqle compiler presented in Chapter 8, non-experts can design parallelizable FHE circuits. However, our methods for addressing impracticality 3: Arithmetization still impose restrictions, and, depending on the evaluated function, the resulting FHE circuits may still require significant computational effort. We conclude that impracticality 4: Compute-intensive is still a dealbreaker for practical deployments of complex functions: users may not be willing to wait for seconds or minutes when a plaintext computation would finish in milliseconds, and companies may not be willing to pay significantly more for secure operations as compared to plaintext operations.

Develop MPSO protocol applications We are not aware of practical deployments of multi-party private set operations. As a result, it is unclear what the context-specific design requirements are of such a protocol. In other words, theree may be context-specific impracticalities that current protocols do not take into account. As shown in Chapter 1, protocols already provide a large set of trade-offs, but it is unclear in what ways these protocols still require changes. One may develop

practical use cases for existing functionalities or for new variants of private set operations.

Increase the efficiency of bootstrapping While homomorphic encryption schemes have seen significant improvements in their efficiency since their inception, bootstrapping remains their bottleneck. Schemes like TFHE do allow for the efficient bootstrapping of a single value, but it remains inefficient to bootstrap a large number of values. On the other hand, bootstrapping in schemes like BFV and BGV supports ciphertext packing, but remains computationally expensive. Alternative approaches such as re-encryption (where a value is masked, decrypted, then re-encrypted before removing the mask) require additional interactions. Future work could design new methods for bootstrapping multiple values efficiently.

More efficient and more intelligent arithmetization Arithmetization is a complex process that scales with the size of a circuit. As such, large circuits may be too slow to arithmetize in practice. A clear next step is to further optimize the arithmetization process. Next to that, the arithmetization process may also still be improved to generate more efficient circuits: the compiler may infer knowledge about plaintexts so that it can perform optimizations it could otherwise not perform. For example, if the compiler can assert that a set of if statements are mutually exclusive, it can significantly reduce the number of branches in the arithmetized circuit. The compiler may also consider automorphism operations to extend beyond pure arithmetic circuits.

Increase the flexibility of FHE libraries FHE libraries typically impose restrictions on the parameters that they permit. For example, we are not aware of any libraries other than HElib that allow the small plaintext moduli we use in Chapters 5 & 6. HElib, too, enforces some restrictions because it does not permit the user to manage the ciphertext modulus manually. These restrictions prevent further optimizations, such as the compiler deciding when and how to switch ciphertext moduli based on the circuit that it is compiling. By lifting these restrictions, a homomorphic encryption compiler may generate more computationally-efficient circuits.

Hybrid protocols As mentioned before, our work on automatic protocol generation may be extended to support hybrid protocols. These protocols, which combine multiple secure computation techniques, may achieve better performance than protocols that only use one technique. It would require significant changes for the extended oraqle compiler to generate such protocols as it would require support for multiple different plaintext algebras within one circuit. When implemented naively, the compiler would consider multiple algebras for each sub-circuit, which may significantly increase compilation time. Future work may research how to generate hybrid protocols efficiently, for example, through the use of heuristics.

Acknowledgments

There is one person I want to thank in particular for the enormous role he played, not only during my PhD, but also for the past six years, in my life. Wouter, you are the love of my life. Over the past four years, you have been thanklessly acting as the second member on my PhD team, often experiencing the consequences of my stress without any of the rewards. You have also carried the burden of being both your own and our collective memory, and you have been on the receiving end of so many sacrifices, such as when I moved away for months, when I was jetlagged, or when I was working late. I will never be able to thank you enough for all your support. I hope that, in a little over a year, we can move into our new house together, overlooking the university and recalling the beautiful memories we made there.

Of course, Wouter is not the only one who experienced the sacrifices I made for my PhD. The volunteer work I dropped, the drinks I didn't join after hockey training, and the family dinners I skipped are all small ways in which my decision to pursue this career turned out to be a somewhat selfish one. I want to thank everyone for giving me the space and the opportunities that allowed me to pursue this career and develop myself.

I also want to express my gratitude to my family, who have always been there for me. I could always count on Daniël and Evelien to cheekily play some Minecraft or to commit culinary crimes at Pavarotti: I'm super fortunate to call you my brother and sister. Also, who gets to write scientific papers with his twin?! In the same way, I could always count on my parents, both for creating fun memories and for asking for advice. I look back with great love and admiration for the constant support I received from my immediate family and I hope to give the same back. I am also grateful to the rest of my family. My love for research and teaching must have surely been influenced by my brilliant grandmother who has been a school teacher in her working life and my late grandfather with infinite curiosity, who went back to university to study some more, even after retiring. Together, they created this amazing family that I look to with great admiration.

Having a partner comes with the benefit of extra family! It has been amazing to get to know Wouter's (step)parents, (step)brothers, and (step)sisters. Going on holidays, seeing musicals, playing hockey tournaments; all these activities contributed to finding some relaxation between all the research. What's more, a partner also comes with extra friends! Thanks for including me on your gaming adventures and zombie movie nights, Mike and Joris. I can't say these nights helped me sleep better, but they certainly took my mind off research. Thank you to Roos for the fun activities and for coining the name 'Wossie'.

Next to my family, I have also been blessed to be supported by two amazing paranymphs: Daniël and Leon. Having you two next to me for my defense feels like a full circle moment. After all, we started our studies together at TU Delft, and now I get to finish it. Over the course of my studies, Leon has been a great best friend, collaborator, and roommate. If I can be even a fraction as ambitious and pragmatic as you, I would be super proud. Together with Daniël we worked hard on so many projects and hackatons, or we hardly worked when we got distracted. Now with Luuk at Leon's side, we get to 'hardly work' even more!

Of course, I want to thank my promotors, Zeki and Mauro, for the trust they placed in me over the past four years: I am super fortunate for the freedom they gave me to follow my curiosity wherever it led and for allowing me to help teach and supervise. Zeki's supervision has been a constant in my life: He has been my supervisor in three separate roles over a span of almost ten years! Zeki, you already challenged me during the context project in the bachelor's together with Daniël, Leon, Matthijs, and Max (and much-appreciated cake moments with Ozzy and Gamze!). Then came my master thesis, in which you taught me to 'hold my horses', which is advice I can still lean on. Finally, as my PhD supervisor, you have been putting up with four years of stubbornness (although I prefer to think of it as determination). Thank you for playing such a big part. At the same time, Mauro's guidance has also taught me a lot throughout my PhD. Mauro, I'm continuously impressed by your ability to reason about all the different security & privacy topics and manage so many projects in parallel. Thank you for teaching me to be more objective and transparent. You also made me feel at home when visiting Italy, and I still recall all the fun conversations we had there (and in the Netherlands). I hope to keep learning from you!

Zeki and Mauro were not the only colleagues contributing to my research and personal development. First and foremost, I want to thank the person at the helm of the cybersecurity group: Sandra! Just kidding, of course, but our group head George and Sandra make a great team together. You are both approachable, empathetic, and generous; you always try to make things right for someone and provide them with the best opportunities. I felt safe in your hands. Sandra, you have meant so much to me and the other students over the past years, often offering the support I and others longed for. George, similarly, you always (literally) have your door open for us. Thank you for your willingness and aptitude to level with students of any level.

One of the downsides of a PhD is that colleagues come and go quite rapidly. As such, the group of PhD students I worked besides when I was writing on this thesis were completely different from the group I started with. However, I admire each of them equally.

Tianyu and Florine, thank you for kicking off my PhD with me and for all the times I could call for your help. Tianyu, you are such a gentle person to be around and such a hard worker. It did not only show in the office but also while playing FIFA, where you would silently destroy us with your years of training. Florine, you are such a kind person and I respect your eye for detail. I cannot commend you for your FIFA skills but I am sure you would beat me in any virtual reality

game. Thank you for all the times I could rely on your advice. Back when I started, our group also included Sicco's students! I want to thank Clinton in particular for our fun conversations at the coffee machine. I don't only speak for myself when I say that our group misses you guys!

In the second half of my PhD, our team was fortunate enough to welcome Jorrit. Jorrit, thank you for all the collaboration and for making the office so 'gezellig'. You are curious, you are a fast learner, and you have a strong sense of righteousness: I think you are the definition of a scientist. After Jorrit came so many new, talented researchers: I am certain that the next generation of research in good hands. I was particularly lucky to be in the office with Tjitske, who I even got to collaborate with right away, along with her supervisor Lilika! Tjitske, you are a brilliant mathematician and now also a Rust programmer. I'm going to miss our banter. Lilika, thank you for all your paper-writing advice and hands-on supervision. I have learned so much from you in a short time. I hope to be able to supervise the way you do. I also want to thank Tjitske and Maarten together for being a jolly duo and for helping me out tremendously with the design of this thesis' cover. Lastly, I want to thank all the other PhD students, old and new, for making this group feel like a group of friends rather than strictly colleagues. I have been so fortunate with you all around.

Besides the PhD students, our group has also been the host of some amazing master students. There are too many fun conversations I have had with you all, and I can only highlight a fraction of them. If I have to choose one, it would be the many tea moments with Dāvis in the past year. Dāvis, thank you for keeping me and the other students sane and caffeinated on busy days! I would also like to thank the students that I got to supervise (for weeks or years!) and learn from at the same time. Armin, Aslı, Codrin, Dan, Dieuwer, Ioana, and Ivo, you are all so talented, making my role super easy. You will do well!

One of the turning points of my PhD was when I got to do an internship at Apple. I want to extend my sincerest gratitude to the team I joined for one summer, and in particular, to Yannick, who has provided me with countless opportunities since then. If I am ever in a leadership position, I aspire to manage as you do. I also want to thank Cathie, Chris, Frederic, George, Steve, and the other team members, who taught me so much in such a short time. Everyone on the team is ambitious, driven, talented, and generally just wonderful to be around. Thank you so much for making me feel at home and part of the team. I wish you all the best in the space donut and elsewhere!

The awesome people I met in Cupertino extended far beyond the team at Apple, though. First and foremost, I was blessed to share the appartment with Cameron, who was an amazing housemate and is an even more amazing researcher. Cameron, I still miss your endless supply of jokes and our late-night conversations about economics, cryptography, machine learning, and what not. I also had such a good time with Rutvik and Joachim, who I have both got to see again after our internships. Rutvik, thank you for all the laughs and for the Powerpoint inspiration! Joachim, I loved your British-Danish humor and thank you for being my Rust helpline. I am grateful for meeting all the other interns, too, over the

course of the summer! Then, in the last week or two of the summer, I met Tabby, who by a stroke of luck, also worked on fully homomorphic encryption. That said, Tabby was and still is way more knowledgeable on the topic than I am. In fact, I think she is probably one of the most knowledgeable FHE researchers I know. Tabby, thank you for all the biweekly meetings after we finished our internships, in which you came up with so many useful suggestions. In a way, you have played the role of my third supervisor. Also, I don't think I have laughed harder during work meetings than I have with you.

One place of constant joy for me has been my hockey team. Heren 5, you guys are such a fun, welcoming, and beautiful group of people. I hope you don't underestimate the positive impact you had on my PhD. Especially during the pandemic, when hockey was one of the few moments to disconnect from work, but also later, when life got more and more stressful. I don't think anyone could convince me to do another PhD, but I would consider it for another couple of years on my favorite hockey team.

Another source of joy and relaxation during my PhD were the group of friends I play video games with. Although I do not get to play as much as I used too (probably much to my parent's delight), I want to thank you for the minutes of joy and peace (and hysterics) you graced me with over the past years. Joep, Nemo, and Max, sorry for all the times I forgot to *glimmer cape* you. Daniël and Alexander, we should sail the salty seas again!

Much of my student life has been spent at Outsite & DWH, Delft's LGBT+ association. I have received so much support from our association, and I regret not helping out so much during the past years. I want to thank the association and all the friends I made there. I am so fortunate to still be in touch with so many of you! Our fake *'dispuut'* is filled with such warm and smart friends. And thank you, Daniel, my old fellow board member and good friend! It's always so refreshing when we call or meet up again. I feel so comfortable around everyone I met at our association; you are truly a pillar of strength.

Another group that I feel so comfortable around, and who have definitely left their mark on my student times and PhD, are my old rowing friends. Every time we get back together it feels like going back in time. I'm so *glad* you guys are still around. I should also thank the person who got me rowing in the first place: Rens. Thank you for bringing so much positive energy when my mind was stuck thinking about work.

In Dutch student life we have a nice word for the friends you made in high school and before: Vrienden van vroeger (friends from the past), or VVV for short. I have not always been the best *vvv'tje*, especially in the past few years where I was constantly occupied by the next project or some urgent task that needed finishing. I cannot thank Alice, Lah Tascha, Megan, Lilian, Veerle, and my other VVV'tjes enough for their patience with me. We may not see each other as often, but I am always so happy when I return from one of our little reunions.

Finally, I want to thank Célio, Martine, Sikha, Steven, and the rest of the PPML Huskies team from Washington Tacoma that I got to join for the PETS prize challenge. Everyone was so committed; I think we made a good team! And of course, I want to thank the thesis committee that took time from their busy schedules to see me dressed as a penguin.

All in all, I cannot help but feel privileged for all the support I got over the past years and how I have been provided with so many good opportunities essentially out of the blue. I hope I can mean the same to the people around me in the future.

Curriculum Vitae

Jelle Victor Vos

March 6, 1998	Born in Delft, The Netherlands
Aug 2009 - Jul 2015	Bilingual VWO, Gymnasium Grotius College Delft
Sep 2015 - Jan 2019	BSc Computer Science & Engineering Delft University of Technology Honours programme Thesis internship at ING Group, Amsterdam
Sep 2019 - Jun 2021	MSc Computer Science Delft University of Technology Cyber security specialisation Graduated cum laude
Jul 2021 - Jun 2025	PhD in Cryptography Delft University of Technology Cyber Security research group
May 2023 - Sep 2023	Cryptographic Engineering Intern Apple Inc., Cupertino, United States of America
Feb 2024 - Jun 2025	Security Engineer Apple Inc., remote

List of publications

Published

1. *Chapter 4:* **Jelle Vos**, Sikha Pentyala, Steven Golob, Ricardo Maia, Dean Kelley, Zekeriya Erkin, Martine De Cock, Anderson Nascimento. 'Privacy-Preserving Membership Queries for Federated Anomaly Detection'. In Proceedings on Privacy Enhancing Technologies, PETS 2024. 10.56553/POPETS-2024-0074

2. *Chapter 6:* **Jelle Vos**, Mauro Conti, Zekeriya Erkin. 'Oraqle: A Depth-Aware Secure Computation Compiler'. In Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC 2024. 10.1145/3689945.3694808

3. *Chapter 1:* **Jelle Vos**, Mauro Conti, Zekeriya Erkin. 'SoK: Collusion-resistant Multi-party Private Set Intersections in the Semi-honest Model', In 2024 IEEE Symposium on Security and Privacy, IEEE S&P 2024. 10.1109/SP54263.2024.00079

4. Armin Memar Zahedani, **Jelle Vos**, Zekeriya Erkin. 'Practical Verifiable & Privacy-Preserving Double Auctions'. In Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023. 10.1145/3600160.3600190

5. Tianyu Li, **Jelle Vos**, Zekeriya Erkin. 'Decentralized Private Freight Declaration & Tracking with Data Validation'. In 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events, BRAIN 2022. 10.1109/PERCOMWORKSHOPS53856.2022.9767444

6. *Chapter 7:* **Jelle Vos**, Daniël Vos, Zekeriya Erkin. 'Efficient Circuits for Permuting and Mapping Packed Values Across Leveled Homomorphic Ciphertexts'. In European Symposium on Research in Computer Security, ESORICS 2022. 10.1007/978-3-031-17140-6_20

7. Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, **Jelle Vos**. 'Practical Multi-Party Private Set Intersection Protocols'. In IEEE Transactions on Information Forensics and Security, IEEE TIFS 2021.10.1109/TIFS.2021.3118879

8. Aslı Bay, Zeki Erkin, Mina Alishahi, **Jelle Vos**. 'Multi-party private set intersection protocols for practical applications'. In 8th International Conference on Security and Cryptography, SECRYPT 2021. 10.5220/0010547605150522

9. **Jelle Vos**, Zekeriya Erkin, Christian Doerr. 'Compare Before You Buy: Privacy-Preserving Selection of Threat Intelligence Providers'. In IEEE International Workshop on Information Forensics and Security, IEEE WIFS 2021. 10.1109/WIFS53200.2021.9648381

Under review

1. *Chapter 2:* **Jelle Vos**, Jorrit van Assen, Tjitske Koster, Evangelia Anna Markatou, Zekeriya Erkin. 'On the Insecurity of Bloom Filter-Based Private Set Intersections'. Under review at the Theory of Cryptography Conference, IACR TCC 2025.

2. *Chapter 5:* **Jelle Vos**, Mauro Conti, Zekeriya Erkin. 'Depth-Aware Arithmetization of Common Primitives in Prime Fields'. Under review at the 34th USENIX Security Symposium, USENIX Security 2025.

Unpublished

1. Daniël Vos, **Jelle Vos**, Tianyu Li, Zekeriya Erkin, Sicco Verwer. 'Differentially-Private Decision Trees with Probabilistic Robustness to Data Poisoning'. ArXiv preprint: arXiv:2305.15394.

2. *Chapter 3:* **Jelle Vos**, Mauro Conti, Zekeriya Erkin. 'Fast Multi-party Private Set Operations in the Star Topology from Secure ANDs and ORs'. Cryptology ePrint Archive: 2022/721.

Encryption is like a lockable box that protects whatever message you store inside. **Homomorphic encryption** extends this functionality by allowing someone to usefully modify the message without seeing it. For example, by introducing a hole in the side of the box that allows one to **manipulate but not see what is inside**. Pictured on the front cover is a locked box with a wheel that encodes a number. Someone may add a small constant to this number by turning the wheel.