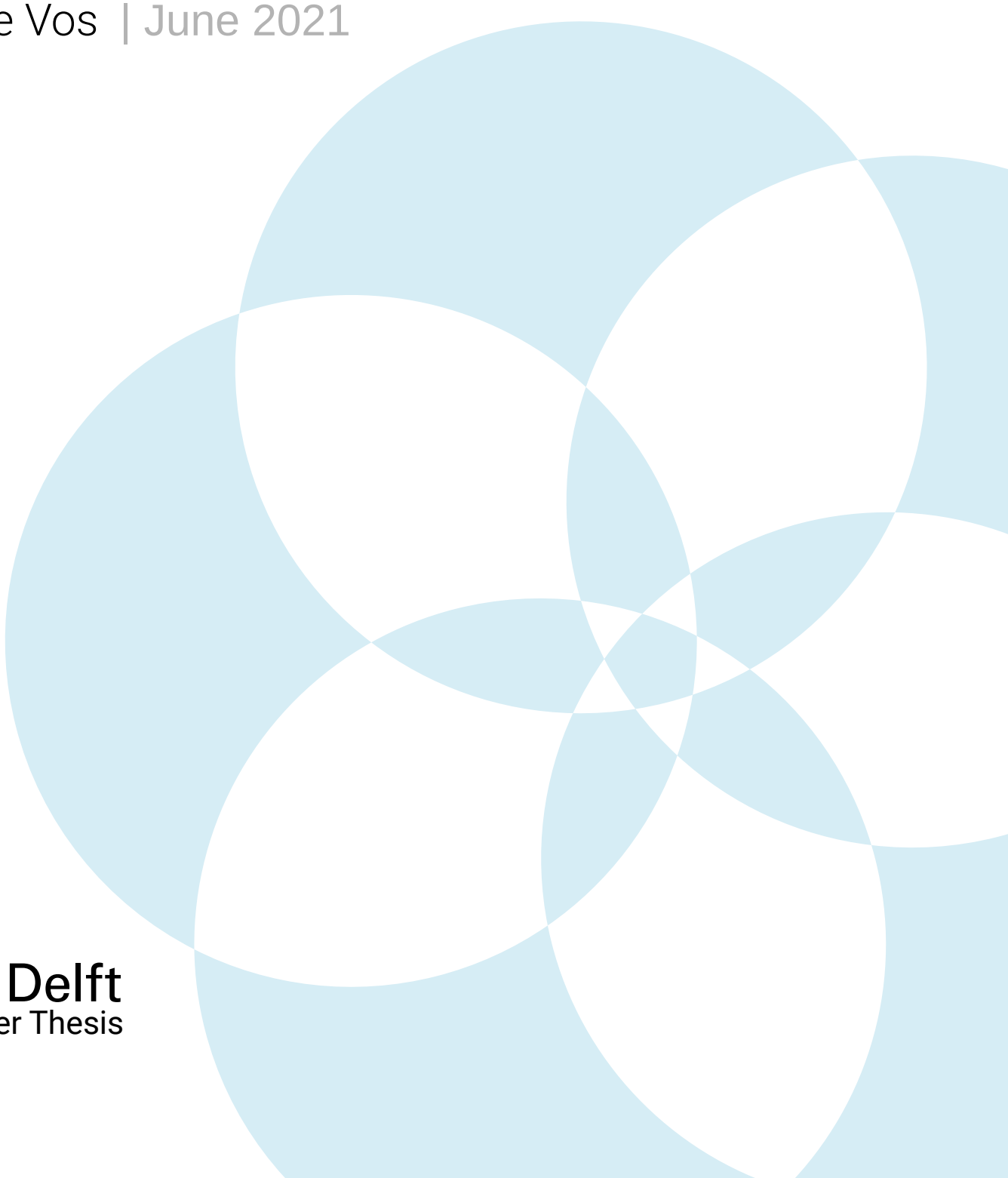
Minimally-Interactive Protocols for

Privacy-Preserving Set and Multiset Operations Between Multiple Parties

Jelle Vos | June 2021



TU Delft
Master Thesis

Minimally-Interactive Protocols for Privacy-Preserving Set and Multiset Operations Between Multiple Parties

Jelle Vos

to obtain the degree of Master of Science

at the *Delft University of Technology*,

to be defended publicly on Monday June 21, 2021 at 10:00AM.

Student number:	4454278	
Project duration:	November 16, 2020 – June 21, 2021	
Thesis committee:	Dr. Z. Erkin	TU Delft, supervisor
	Dr. S. Picek	TU Delft
	Dr. L. Chen	TU Delft

An electronic version of this thesis is available at https://repository.tudelft.nl.

Abstract

In our increasingly digital society, we are 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 and multiset operations. In such a scenario, multiple parties, each with their own input set or multiset, want to collectively find the result of an operation over their inputs, without revealing these original inputs. Such operations are the cornerstone of many complex privacy-preserving protocols. For example, a two-party private set intersection forms the key to several privacy-preserving contact tracing protocols.

While multi-party private set and multiset operations have been studied for almost two decades, these privacy-preserving alternatives are often impractical: one limitation is that, to the best of our knowledge, all known protocols require several interactions between the cooperating parties. This means that rather than simply submitting their input, each party must actively take part in the protocol. In this thesis, we propose the first non-interactive protocols for privately computing set and multiset operations between multiple parties, which rely on two constructions for non-interactive secret sharing. In addition, for operations that cannot be trivially performed using our non-interactive primitives, we propose minimally-interactive alternatives that instead rely on a homomorphic cryptosystem over elliptic curves. By using elliptic curves, this cryptosystem is faster and requires less bandwidth than the commonly used cryptosystems over integers, while retaining the same level of security. We provide proof-of-concept implementations of exact and more efficient approximate protocols that take on the order of seconds to minutes to compute, depending on the number of parties and possible inputs. Finally, we give formal proofs for the security of these protocols, so as to offer practical and provably privacy-preserving alternatives to otherwise sensitive operations.

Preface

Thank you for reading my thesis, which was written to fulfill the requirements of the computer science master with a specialization in cyber security at the Delft University of Technology. While the thesis is lengthy, I have split it up into short chapters, to provide a pleasant reading experience for those who are interested in a selection of topics or protocols.

For a reader who wants to gain a quick insight in this thesis, I would recommend reading:

- Chapter 1
- Chapter 2
- Chapter 5
- Chapter 20

Additionally, Chapters 22 & 25 present results for real-life scenarios.

1 On the cover

The image on the cover is a Venn diagram, every surface and combination of surfaces represents the result of a valid set operation. Actually, the blue surfaces represent the symmetric difference between five sets, which is an operation that can be expressed in terms of all primitive set operations: intersections, unions, and complements.

2 On this thesis format

The main author of this format is Federico Marotta, and it was based on the doctoral thesis by Ken Arroyo Ohori. The format itself is called `kaobook`. Although the format is perhaps unusual for a master's thesis, I have chosen for a format with a large margin to:

- Highlight some citations
- Include helpful sidenotes
- Provide a more pleasant reading experience for captions
- Put smaller figures and tables in the margin
- Save space for a reader's notes

3 On the implementations

We have implemented proof of concepts for most of the ideas we put forward in this thesis.* This includes Python, C++ and Rust code. We do not provide the code to our experiments.

* The implementations can be found at https://github.com/jellevos/thesis-MPSO-MPMO

4 Acknowledgements

During my thesis I was supervised by Zeki Erkin. I want to thank him for his personal help during my thesis process, and the late hours he spent reviewing my work. I have learned so much over the course of 9 months, and I truly believe no other supervisor could have achieved what he has. Moreover, in the hard time that is the COVID-19 pandemic, even with people working from home, Zeki managed to bring the research group together, to provide weekly supervision and to keep me motivated throughout, for which I am extremely grateful.

I also want to thank all members of the research group for their constant help and collaboration. I felt like I always had someone to consult if necessary. Also, I want to congratulate the research group on their sublime cooking skills.

I also want to express my gratitude for the technical discussions that I had with my twin Daniël and my friend Leon Overweel, which were very valuable to my thesis.

Finally, I want to thank friends and family who have really kept me going through this pandemic and through my thesis at the same time. Specifically, I want to thank my boyfriend Wouter for putting up with me through stressful times and always being there for me. I hope I can help out with your thesis as much as you have with mine.

Jelle Vos
Delft, June 2021

Contents

Abstract iii

Preface iv
1 On the cover . . . iv
2 On this thesis format . . . iv
3 On the implementations . . . iv
4 Acknowledgements . . . v

Contents vi

1 Introduction 1
1.1 Private contact tracing . . . 2
1.2 State of the art . . . 3
1.3 Research questions . . . 4
1.4 Contributions . . . 5
1.5 Outline . . . 6

Private set & multiset operations 8

2 Sets & set operations 9
2.1 Sets . . . 9
2.2 Set operations . . . 9
2.3 Applications for set operations . . . 11

3 Multisets & multiset operations 12
3.1 Multisets . . . 12
3.2 Multiset operations . . . 12
3.3 Multisets as sets . . . 13
3.4 Multiset applications . . . 14

4 Multi-party privacy-preserving set & multiset operations 15
4.1 Setup . . . 15
4.2 Correctness requirements . . . 16
4.3 Privacy requirements . . . 17
4.4 Terminology . . . 18

Set & multiset representations 19

5 Bitsets 20
5.1 Creating bitsets . . . 20
5.2 Operations on bitsets . . . 20
5.3 Count vectors . . . 21

6 Bloom filters 22
6.1 Creating Bloom filters . . . 22
6.2 Operations on Bloom filters . . . 24
6.3 Counting Bloom filters . . . 26

7 Polynomial roots **27**
7.1 Polynomial multiset encoding . . . 27
7.2 Operations on polynomials . . . 27

8 Other set representations **29**
8.1 Garbled Bloom filters . . . 29
8.2 Min-max sketches . . . 29

Cryptographic building blocks **30**

9 Hardness assumptions & elliptic curves **31**
9.1 Decisional Diffie-Hellman . . . 31
9.2 Elliptic curve groups . . . 31
9.3 Pairing-based cryptography . . . 32

10 Secret sharing methods **34**
10.1 (n, n)-secret sharing . . . 34
10.2 (t, n)-secret sharing . . . 35

11 Homomorphic cryptosystems **36**
11.1 ElGamal cryptosystem . . . 36
11.2 Paillier cryptosystem . . . 38
11.3 Decryption to zero . . . 38

12 Summary of preliminaries **39**
12.1 Notation . . . 39
12.2 Hardness assumptions . . . 40

Previous work **41**

13 Multi-party private set intersections **42**
13.1 Polynomial roots-based MPSI . . . 42
13.2 Bitset-based MPSI . . . 44
13.3 Bloom filter-based MPSI . . . 44
13.4 Other MPSI . . . 45
13.5 Summary . . . 46

14 Multi-party private threshold-set operations **48**
14.1 Polynomial roots-based T-MPSO . . . 48
14.2 Bloom filter-based T-MPSO . . . 49
14.3 Other T-MPSO . . . 50
14.4 Summary . . . 50

15 Multi-party private set unions **51**
15.1 Polynomial roots-based MPSU . . . 51
15.2 Summary . . . 52

16 Multi-party private set operation-cardinality protocols **53**
16.1 Polynomial roots-based MPSO-CA . . . 53
16.2 Bitset-based MPSO-CA . . . 54
16.3 Bloom filter-based MPSO-CA . . . 54

16.4 Other MPSO-CA . . . 54
16.5 Summary . . . 56

17 Multi-party private multiset operations **57**
17.1 Polynomial roots-based MPMO . . . 57
17.2 Bloom filter-based MPMO . . . 57
17.3 Other MPMO . . . 58
17.4 Summary . . . 58

Non-interactive protocols **59**

18 Non-interactive secret sharing **60**
18.1 Outline . . . 60
18.2 Symmetric construction . . . 61
18.3 Asymmetric construction . . . 62

19 Multi-party private `OR` & `AND` operations **66**
19.1 Protocol description . . . 66
19.2 Correctness . . . 67
19.3 Security . . . 67

20 Multi-party private set and multiset operations **71**
20.1 Set intersections . . . 71
20.2 Set unions . . . 72
20.3 Multiset intersection & union . . . 76
20.4 Multiset sums . . . 78

21 Multiset sums within one symmetric share **79**
21.1 Prime factors as multisets . . . 79
21.2 Multiset sum protocol . . . 82
21.3 Results . . . 84

22 Private collaborative detection of malicious IP addresses **86**
22.1 Dataset . . . 86
22.2 Results . . . 88

Interactive protocols **90**

23 Set intersection & union cardinality **91**
23.1 Shuffle-decrypt protocol . . . 91
23.2 MPSO-CA protocol . . . 94

24 Threshold set intersections **96**
24.1 Shuffle-decrypt-to-zero . . . 96
24.2 Threshold set intersections . . . 96

25 Privacy-preserving selection of threat intelligence providers **99**
25.1 Setup . . . 99
25.2 Results of scenario A . . . 100
25.3 Results of scenario B . . . 100
25.4 Use in practice . . . 101

Conclusion **102**

26 Conclusion & discussion **103**
26.1 Discussion 103
26.2 Future work 105

A Adaptations of previous works' complexities **106**
A.1 MPSI protocols 106
A.2 T-MPSO protocols 110
A.3 MPSU protocols 111
A.4 MPSO-CA protocols 112
A.5 MPMO protocols 114

Bibliography **116**

1 Introduction

1.1 Private contact tracing 2
Private set intersection 2
Private intersection-cardinality 2
Private set union-cardinality . 3
1.2 State of the art 3
1.3 Research questions 4
1.4 Contributions 5
Summary 6
1.5 Outline 6
Preliminaries 6
Related work 7
Our work & results 7

As we progress in the 21st century, we are witnessing digitalization all around: For example, while bank transactions in the past required a great deal of administration, they can now be done digitally from the comfort of your own home. Moreover, we can reach out to others at every moment of the day via chat services, social media and video calls. We can even search the web from the most distant places, in order to find important information whenever we need it.

It is undeniable that this digitalization has brought with it comfort and wealth, but there are downsides. In particular, we are producing and sharing data at a growing rate, and this is putting our privacy-sensitive data at a risk of being leaked. To address the issue of privacy while embracing the benefits of digitalization; that is the ideal of the field of multi-party computation. More specifically, multi-party computation [1] aims at developing protocols, which enable computation of a function with secret inputs.

Among those operations are multi-party private set and multiset operations, which form the cornerstone of several privacy-preserving protocols. In short, a multi-party private set operation is a collaboration between two or more parties, each with a set as input, who want to collectively find the result of some set operations over these sets, without revealing the inputs. Since sets are distinct collections of elements, some applications require computations over multisets, which allow elements to appear several times within the collection.

Since multi-party private set and multiset operations serve a role in countless privacy-preserving protocols [2, 3], they have already been studied for almost two decades. Still, there are factors that hinder practical use of such protocols. In particular, while protocols have become faster over the years, all proposed protocols, to the best of our knowledge, require several interactions between the collaborating parties. For example, where the privacy-less way of finding a common date between you and your colleagues only required them to send you their availability, the current privacy-preserving alternatives require all colleagues to communicate in multiple rounds of a protocol. As a result, these protocols typically require more bandwidth, and cooperating parties must be online at several stages.

In this work we propose the first non-interactive protocols for these purposes. Going by the previous example, these protocols only require your colleagues to send a cryptographically-secured version of their availability once, after which you can privately and locally compute the result of the set or multiset operation. Still, some set operations require a functionality that we cannot capture with our non-interactive techniques.[1] For those operations, we provide protocols that require the minimum number of required interactions.

While our proposed protocols generally incur a larger cost in computation than other recent works, we pose that our protocols are fit to be deployed

1: That is, using the same techniques leads to contradictory requirements for certain operations.

in situations where others are not, such as when a network is lossy or has a high latency, or in situations where messages must be sent manually, such as when a set operation must be performed only incidentally. In addition, by using elliptic curves, we bring this extra cost down significantly; in some settings outperforming state of the art protocols in terms of run time. Ultimately, we deem that practicality does not only entail a short run time, but also a low degree of interactivity.

1.1 Private contact tracing

Multi-party private set and multiset operations serve countless applications.[2] For instance, those applications where parties work with sets containing private data of other individuals, or applications where parties hold data in their sets that could damage them if it was leaked or linked to them. As a motivating example, we examine the application of contact tracing in the context of a pandemic. Due to the COVID-19 pandemic, there are several recent works that propose protocols based on multi-party private set operations for this application [2, 4]. We provide three examples of use cases in the context of private contact tracing that rely on private set operations between two or more parties.

2: See Chapters 2 & 3 for more examples.

Private set intersection

Consider a contact tracing mobile phone application. The mobile phone will constantly search for other phones in its vicinity to register that it has been in close proximity, and thereby identifying a moment of disease transmission for the individual carrying it. The registration happens by exchanging a pseudonymous identifier that is personal to that mobile phone, but it does not contain otherwise identifying information.

When an individual receives a positive test result for infection, the application should notify the phones of those who have been in close proximity to the infected individual. Let us say that this is realized by a health agency, which registers the pseudonyms of those who are infected. The application can then occasionally check in with the health agency to see if any of its registered pseudonyms turn out to be infected. However, in doing so, the health agency can form a complete graph of all phones and who they have been in contact with.

Ideally, the health agency does not learn anything about the registered pseudonyms on each phone, but only about the infected pseudonyms registered on it. At the same time, the individual with a mobile phone should not learn the entire list of pseudonyms of infected individuals. Fortunately, this is realized by a private set intersection: The health agency and the phone only learn about the pseudonyms that they have in common.

Private intersection-cardinality

In the previous example, the health agency and the phone of someone checking for risky contacts both receive the pseudonyms of infected

individuals. One might argue that this still enables the health agency to perform detailed statistical analysis, and the individual carrying the phone might compare with close friends to identify the persons belonging to the received pseudonyms. Instead, it should be sufficient for both the health agency and the individual carrying a mobile phone to learn only about the size of the intersection, rather than the pseudonyms contained in it.

In the study of private set operations, the size of a set is called its cardinality, so this operation is called a private set intersection-cardinality. By performing such a two-party private set intersection-cardinality protocol, both parties can still fulfill their purposes, but it is much harder for them to learn any additional information, guaranteeing privacy of the users of this application as well as the privacy of infected individuals.

Private set union-cardinality

In the previous examples, we have only considered intersection-based operations between two parties. However, other set types of private set operations can also play an important role in contact tracing. For example, consider a feature that allows you to estimate the chance of exposure in a group of people, by identifying the size of the 'bubble' that these people are contained in. In other words, finding out how many different people have been in contact with any member of the group. An operation realizing this must extend to more than two parties, and it must return the total set of contacted individuals rather than those that each party has in common.

Fortunately, this operation can be achieved by a multi-party private set union-cardinality protocol. Again, the group only strictly needs to learn about the size of the 'bubble' rather than all its related pseudonyms. Note that such a protocol also hides how many members of the group have met each pseudonym.

In short, private set operations form the basis of multiple privacy-preserving operations within the topic of contact tracing. Nonetheless, private set operations serve many more applications, such as collaborative malicious host detection and no-fly list creation. We have also not yet touched upon multiset operations, which extend the range of possible applications even further.

1.2 State of the art

One of the most recent works on multi-party private set intersections is the work by Kolesnikov et al. [5]. While this work is fast in terms of run time for large sets, each party must communicate with every other party in two stages of communication. Additionally, their protocol requires significant bandwidth. In Chapter 20 we provide a non-interactive multi-party private set intersection that requires less communication and which is almost an order of magnitude faster than Kolesnikov et al. [5] for a growing number of involved parties and a small collection of possible encoded set elements.

When it comes to threshold set operations, where elements are only included in the resulting set when they appear in the sets of at least some threshold number of parties, most works require four or more stages of communication. The work with the least interactions is by Miyaji and Nishida [6], which takes two stages of communication, but the protocol reveals how often each element in the resulting appeared in the input sets. Moreover, the protocol uses the ElGamal cryptosystem over the integers, which requires large keys and expensive computations for the necessary level of security. We propose a protocol with two stages of communication that does not reveal how often each element appeared in the input sets, and we ElGamal over elliptic curves to achieve smaller keys, less bandwidth and faster computation.

Similarly, for multi-party private set operation-cardinality protocols, to the best of our knowledge, all works require at least three stages of communication. We provide a protocol that runs in just two stages of communication, and we show experimentally that an approximate version of the protocol runs in the order of seconds on a single thread, even when the final set contains $20,000$ elements, and with a standard deviation of 251.

For multi-party private multiset sums, the work by Hong et al. [7] provides experimental results for an interactive protocol. We provide two exact protocols that out-compete this work in terms of run time on the same experiments, and both of these protocols are non-interactive.

We provide an extensive overview of previous works, as well as their complexities, their required number of stages of communication and other properties in Chapters 13 – 17.

1.3 Research questions

In this thesis we set out to create provably privacy-preserving set and multiset protocols that are non-interactive, and otherwise require as few interactions as possible. After all, a non-interactive protocol would closely resemble one's workflow when performing these operations without preserving privacy. Specifically, we answer the following research question:

RQ How can multiple parties collaboratively, but with minimal interactions, compute set and multiset operations —such as set intersections —without revealing their inputs?

In those cases where we cannot provide a non-interactive protocol, we provide minimally-interactive alternatives. As such, we make a distinction between non-interactive and minimally-interactive protocols in the last two parts of this thesis, where we answer the corresponding sub-questions, respectively:

- **SQ1** What operations can be performed non-interactively and how can we do so?
- **SQ2** What operations must be performed interactively, and how can we do so with the minimum number of interactions?

1.4 Contributions

The work in this thesis has lead to four publications, of which three are still in the peer-reviewing process. Additionally, some parts contain new work that is not yet part of any publications. Specifically, this concerns the extension of our non-interactive protocols to the symmetric construction described in Section 18.2, the reversible hash functions described in Subsection 20.2 and the cardinality & threshold set operations in Chapters 23 & 24. We evaluate the set cardinality protocols in Chapter 25 for the context of 'shopping' for threat intelligence, which is a yet unpublished idea by Christian Doerr and Zekeriya Erkin.

Specifically, at the time of writing, these are our publications and the stage of the peer-reviewing process they are in.

Multi-party Private Set Intersection Protocols for Practical Applications

Accepted at SECRYPT 2021
Aslı Bay, Zeki Erkin, Mina Alishahi, and Jelle Vos

In this work, we describe and implement exact multi-party private set operations using bitsets, which is a fast encoding for small universes. We demonstrate how run time of the protocol scales linearly with the number of parties, which is more efficient than the state of the art, which scales quadratically.

Practical Multi-party Private Set Intersection Protocols

Under review at IEEE Trans. on Information Forensics and Security
Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos

Using the threshold Paillier cryptosystem and a secure comparison protocol, we describe and implement protocols for approximate multi-party private set intersections and threshold set intersections. This protocol, too, scales linearly with the number of parties. Additionally, since we use Bloom filters to encode the sets, the protocols are fast, regardless of the size of the universe.

Non-interactive Multi-party Private Set and Multiset Operations for Collaborative Malicious Host Detection

Second round at IEEE Security & Privacy 2022
Jelle Vos and Zekeriya Erkin

This work contains many findings that we present in this thesis. Specifically, we describe and implement exact non-interactive protocols for set intersections, set unions, multiset intersections, multiset unions and multiset sums that scale with the size of the universe. In addition, we propose approximate non-interactive protocols for set intersections and multiset intersections, and we provide a straw man proposal for extending this to perform approximate unions and multiset sums. The protocols rely on secure `AND` and `OR` protocols that may be of independent interest.

Practical Multi-party Private Multiset Sums using Single-Share Multiset Encodings

Under review at ESORICS 2021
Jelle Vos, Martin Koster, and Zekeriya Erkin

In this work, we propose and implement an exact protocol for multiset sums, where each party's set encoding fits entirely in a single secret share. As such, the protocol requires minimal bandwidth. The protocol relies on the fact that each integer has a unique prime factorization, which we exploit to encode sets as a single integer. We decode this integer using the elliptic curve factorization method.

Summary

Our contributions can be summarized as follows:

- The first non-interactive protocols for set intersections, set unions, multiset intersections, multiset unions, and multiset sums, based on non-interactive `AND` and `OR` protocols using multiplicative secret sharing that may be of independent interest.
- A comprehensive analysis of previously proposed schemes for set and multiset operations, and a comparison of their asymptotic complexities and security among others.
- Minimally-interactive protocols for threshold set operations and set operation-cardinalities based on the threshold ElGamal cryptosystem over elliptic curves using a simultaneous shuffle-decrypt protocol that may also be of independent interest.
- Both exact and more efficient approximate versions of all proposed protocols that enable operations over large domains such as the IPv4 address space.
- An implementation of all proposed protocols, which we evaluate in practical scenarios within the cyber security domain and depending on the parameters demonstrate to be faster than implementations of previous multi-party set intersection and multiset sum protocols.

1.5 Outline

The rest of this thesis is split into several parts that each contain a number of short chapters. The parts 'Non-interactive protocols' and 'Interactive protocols' discuss our own contributions. Table 1.1 shows in what chapter we discuss which protocol. In the final part, we discuss our findings and conclude. The parts before that can be divided into the following categories.

Preliminaries

We split the preliminaries into three parts. In 'Private set & multiset operations' we introduce sets and multisets, as well as their respective operations, and we put it in the context of multi-party private operations. In the next part 'Set & multiset representations' we discuss the underlying representations that are commonly used to encode sets, of which our work

specifically considers bitsets and Bloom filters. The part on 'Cryptographic building blocks' puts forward the cryptographic primitives and protocols that form the basis of many previous works including ours, as well as the security assumptions that multi-party private set and multiset protocols often rely on.

Related work

Our related work contains one part: In 'Previous work' we give a comprehensive overview of the prevalent works in multi-party private set and multiset protocols. This part contains the work related to the new protocols that we introduce.

Our work & results

We have two parts that discuss our own contributions. In 'Non-interactive protocols' we put forward all our non-interactive protocols, and in 'Interactive protocols' we put forward minimally-interactive protocols that are based on the same techniques.

Table 1.1: Overview of common multi-party private set & multiset operations and in which chapters they are discussed.

Short	Operation	Previous work	Our work
`MPSI`	Set intersection	Chp. 13	Chp. 20
`MPSU`	Set union	Chp. 15	Chp. 20
`MPMI`	Multiset intersection	Chp. 17	Chp. 20
`MPMU`	Multiset union	Chp. 17	Chp. 20
`MPMS`	Multiset sum	Chp. 17	Chp. 20 & 21
`MPSI-CA`	Set intersection-cardinality	Chp. 16	Chp. 23
`MPSU-CA`	Set union-cardinality	Chp. 16	Chp. 23
`T-MPSI`	Threshold set intersection	Chp. 14	Chp. 24
`T-MPSU`	Threshold set union	Chp. 14	Chp. 24

Private set & multiset operations

Sets & set operations 2

A set is an unordered collection of distinct elements. These structures arise in many problems. For example, when you want to find a suitable meeting date with multiple colleagues, each colleague has a collection of different available dates in no particular order. The suitable meeting dates are the set of those dates that are in each colleague's set, which is an example of a set operation. Specifically, a set intersection.

2.1 Sets 9
2.2 Set operations 9
Basic operations 10
Algebra of sets 10
Compound operations 11
2.3 Applications for set operations 11

2.1 Sets

The notation for a set containing elements Tuesday, Wednesday and Friday is $\{\text{Tuesday}, \text{Wednesday}, \text{Friday}\}$. We say that the universe $\mathcal{U}$ is the set of all possible elements; in this case that is the set containing every day of the week. In this work, though, we are only concerned about sets of integers. This is without loss of generality, given that one can devise a mapping from every element in $\mathcal{U}$ to an integer. For example, we can devise a mapping where Monday $\rightarrow 0, \ldots,$ Sunday $\rightarrow 6$, so that our original set becomes $\{1, 2, 4\}$.

While in theory a set can contain infinitely many elements, for practical applications, we typically consider finite sets. The number of elements contained in a set X is referred to as its *cardinality*, and it is denoted by $|X|$. Since a set can only contain as many elements as the universe $\mathcal{U}$, the cardinality of any set is bound by $0 \leq |X| \leq |\mathcal{U}|$. For this reason we also refer to $|\mathcal{U}|$ as the domain size. In practical applications we can often provide a bound for the maximum number of elements in a set that is tighter than the domain size. We denote this bound by k, and we refer to it as the maximum set size. In our previous example, the domain size $|\mathcal{U}|$ is 7 as it contains every day of the week. The maximum set size k could be 5 if we know that every colleague has their weekends planned full already.

2.2 Set operations

Sets by themselves are simple structures, but when combined with other sets using set operations, they can capture complex behavior. In our example of finding a suitable meeting date between several colleagues we have already hinted about one such operation: the set intersection. In this thesis, we are only concerned about set operations that are extensible to an arbitrary number of input sets, and we formulate them as such. Note, however, that there exist different formulations for some of these operations, and that there exist other operations that only apply to a specific number of sets.

Basic operations

The result of a set intersection is the set of elements that were contained in each of the original sets. In the context of our example, given the sets of available dates of n colleagues, $X_1, \ldots, X_n$, the set intersection, and thereby the set of suitable dates, is formally defined as:

Definition 2.1 (Set intersection) *Given n sets $X_1, \ldots, X_n$, their intersection Z is defined as:*

$$Z = \{x \mid \forall_{i=1}^{n} x \in X_i\} = \bigcap_{i=1}^{n} X_i$$

Note that the cardinality of the resulting set Z is at most $|Z| \leq k$, since this set cannot contain more elements than any of the original sets $X_1, \ldots, X_n$. Suppose now that we wish to find the set of elements that were submitted in any of the original sets, this is equivalent to the set union operation, which is formally defined as:

Definition 2.2 (Set union) *Given n sets $X_1, \ldots, X_n$, their union Z is defined as:*

$$Z = \{x \mid \exists_{i=1}^{n} x \in X_i\} = \bigcup_{i=1}^{n} X_i$$

This time, the resulting set can contain at most nk elements.

Algebra of sets

While the purpose of intersections and unions may be immediately apparent, this may not be the case for the complement operation, which is an important operation in the algebra of sets. The complement of a set X is the set $\overline{X}$ that contains all elements from the universe $\mathcal{U}$ that are not in set X. In that sense, it is the exact opposite of set X. The operation is formally defined as:

Definition 2.3 (Set complement) *Given a set X, its complement $\overline{X}$ is defined as:*

$$\overline{X} = \{x \notin X \mid x \in \mathcal{U}\}$$

The set intersection and set union are actually related to each other through the set complement under DeMorgan's law. Through this property, it is possible to express a set intersection in terms of a set union and vice versa. In fact, this law holds both in set theory and in Boolean algebra, where it is possible to express an `AND` operation in terms of an `OR` operation. DeMorgan's law is formally defined as:

Definition 2.4 (DeMorgan's law) *Given n sets $X_1, \ldots, X_n$, the following identities hold:*

$$\overline{\bigcap_{i=1}^{n} X_i} = \bigcup_{i=1}^{n} \overline{X_i} \quad \text{and} \quad \overline{\bigcup_{i=1}^{n} X_i} = \bigcap_{i=1}^{n} \overline{X_i}$$

Compound operations

We can describe richer set operations by combining the basic operations defined before. Threshold unions, for instance, contain those elements which appear in at least a certain number of the original sets. This number is given by a threshold τ. The threshold union is a compound operation because we can write it in terms of intersections and unions.

Example 2.1 Given $n = 3$ sets X_1, X_2 and X_3, and a threshold $\tau = 2$, the threshold union Z can be written as:

$$Z = (X_1 \cap X_2) \cup (X_2 \cap X_3) \cup (X_1 \cap X_3)$$

Indeed, there is also a threshold intersection operation. This operation is defined as the intersection between one of the original sets and the result of the threshold union. This operation has favourable properties in practical applications, because the resulting set is guaranteed to contain at most k elements. In this thesis, we focus on operations where the number of elements *exceeds* a threshold.

Of course, any composition of the aforementioned set operations makes for a valid compound operation. Some of them have special names. For example, the set difference of sets A and B defined by $A - B$ is equivalent to the intersection with the complement of B: $A \cap \overline{B}$. A similar operation, called the symmetric difference $A \triangle B$, returns the elements that are in one of the sets, but not in their intersection. In other words, it can be written as $(A \cup B) - (A \cap B)$. In this thesis we do not explicitly discuss these operations, but note that it is possible to perform these operations by expressing them in terms of intersections, unions and complements.

2.3 Applications for set operations

Set operations form the solution to many real-life problems. We provide three examples:

- **Set intersection:** In this chapter we introduced set operations based on the example of finding a suitable date between colleagues. In this example, the sets contain suitable dates and the result is computed using the set intersection.
- **Set unions:** One use case for set unions is the creation of no-fly lists. Several agencies can prevent people from flying, but it would leak information if an agency knew which individuals the other agencies were investigating. Considering that all agencies have a set of personal identifiers, the corresponding set union will contain all identifiers, without reference to what sets they originated from or how often they occurred.
- **Identifying profiteers:** Consider multiple charities that financially support individuals, then these charities can identify profiteers among them who are benefiting from multiple charities at once using a threshold intersection. So, each charity has a set of personal identifiers of those individuals they support, and the resulting set will contain the identifiers of those who are benefiting from at least τ charities at once.

Multisets & multiset operations

3

3.1 Multisets 12
3.2 Multiset operations 12
3.3 Multisets as sets 13
3.4 Multiset applications 14

Multisets are an extension of sets in which element can appear multiple times. So, a multiset is an unordered collection of elements that are not necessarily distinct. For this reason, a multiset is also referred to as a 'bag' in natural language.

3.1 Multisets

Other than sets, multisets capture the concept of multiplicity. That is to say, for a multiset X there is a formal multiplicity function $\mathcal{M}_X(x)$ that returns how often element x is contained in X. The notation for a multiset containing three 1s and two 2s is $[1, 1, 1, 2, 2]$. For this multiset, $\mathcal{M}_X(1) = 3$ and $\mathcal{M}_X(2) = 2$, and it is zero for all other elements in the universe $\mathcal{U}$.

In practical applications, we typically consider finite multisets, so the multiplicity function has some upper bound M that we refer to as the maximum multiplicity. In addition, it is useful to define an upper bound for the maximum number of distinct elements in a multiset. To prevent confusion with the maximum set size k we denote this number by K.

3.2 Multiset operations

Like for sets, multisets can be combined using intersections and unions. However, there is some nuance when it comes to these operations. For simplicity we define the operations in terms of the multiplicity function. The multiset intersection is as follows:

Definition 3.1 (Multiset intersection) *Given n multisets $X_1, \ldots, X_n$, their intersection Z is defined as:*

$$Z = \bigcap_{i=1}^{n} X_i \quad \textit{where} \quad \mathcal{M}_Z(z) = \min_{i=1,\ldots,n} \mathcal{M}_{X_i}(z) \quad \textit{for} \quad z \in \mathcal{U}$$

In other words, the multiset intersection contains the elements that appear in every input multiset, and exactly as often as these elements appeared in the multiset that contained the fewest of that specific element.

Example 3.1 Given multisets $X_1 = [1, 2, 3, 3]$, $X_2 = [2, 2, 3, 3]$ and $X_3 = [1, 2, 3, 3, 3]$, the intersection is:

$$Z = X_1 \cap X_2 \cap X_3 = [2, 3, 3]$$

The union is the opposite of the intersection in that the multiplicity of each element in the union is the maximum multiplicity of the element in

the original multisets, rather than the minimum. Formally it is defined as follows:

Definition 3.2 (Multiset union) *Given n multisets $X_1, \ldots, X_n$, their union Z is defined as:*

$$Z = \bigcup_{i=1}^{n} X_i \quad \textit{where} \quad \mathcal{M}_Z(z) = \max_{i=1,\ldots,n} \mathcal{M}_{X_i}(z) \quad \textit{for} \quad z \in \mathcal{U}$$

Using the same multisets as in Example 3.1, the union is as follows:

Example 3.2 Given multisets $X_1 = [1,2,3,3]$, $X_2 = [2,2,3,3]$ and $X_3 = [1,2,3,3,3]$, the union is:

$$Z = X_1 \cup X_2 \cup X_3 = [1,2,2,3,3,3]$$

Apart from the intersection and union, another multiset operation is the multiset sum. In literature [7, 8] , this operation has been mistaken as the multiset union, but there is a strict difference. The multiset sum is the total collection of all elements contained in the original multisets, so each element's multiplicity is summed:

[8]: Huang et al. (2012), 'Privacy-preserving multi-set operations'
[7]: Hong et al. (2013), 'Constant-Round Privacy Preserving Multiset Union'

Definition 3.3 (Multiset sum) *Given n multisets $X_1, \ldots, X_n$, their sum Z is defined as:*

$$Z = \biguplus_{i=1}^{n} X_i \quad \textit{where} \quad \mathcal{M}_Z(z) = \sum_{i=1,\ldots,n} \mathcal{M}_{X_i}(z) \quad \textit{for} \quad z \in \mathcal{U}$$

The multiset sum for the same multisets as in Examples 3.1 and 3.2 is:

Example 3.3 Given multisets $X_1 = [1,2,3,3]$, $X_2 = [2,2,3,3]$ and $X_3 = [1,2,3,3,3]$, their sum is:

$$Z = X_1 \uplus X_2 \uplus X_3 = [1,1,2,2,2,2,3,3,3,3,3,3,3]$$

Interestingly, the natural numbers are also examples of multisets, since each has a unique prime factorization.[1] As a result, multiplication between natural numbers is equivalent to the multiset sum of their respective prime factorizations. Moreover, the greatest common divisor is equivalent to the multiset intersection, and the least common multiple is equivalent to the multiset union.

1: We exploit this property to encode multisets as natural numbers in Chapter 21.

Finally, the threshold operations as described in the previous chapter work similarly for multisets.

3.3 Multisets as sets

Although multisets are fundamentally different from sets in that they capture the concept of multiplicity, there is a simple transformation from multisets to sets in the case where the universe and the multiplicity domain are finite [9] . Consider a multiset $[1,2,2,3]$. For simplicity, let us

[9]: Blanton et al. (2016), 'Private and oblivious set and multiset operations'

write it as $[1_1, 2_2, 3_1]$, where x_y denotes that element x is contained in the multiset with multiplicity y. Then we can transform this multiset to the following equivalent set $\{1_1, 2_1, 2_2, 3_1\}$. The idea behind the transform is that since 2 is contained twice in the multiset, it is also contained once. Formally, the transform is as follows:

$$\{x_i \mid x \in \mathcal{U}, 1 \leq i \leq \mathcal{M}(x)\}$$

Due to the existence of this transformation, the majority of this thesis discusses operations on sets rather than multisets, as these are applicable to multisets as well after the transformation. One exception is our protocol for multiset sums as described in Chapter 21, which strictly applies to multisets.

3.4 Multiset applications

Like sets, multisets serve several real-life applications, including:

- **Web traffic analysis:** Let us say that a group of web users track the websites they visit by keeping a multiset containing their URLs. Then, we can perform web traffic analysis using multiset intersections, unions and sums to identify how often certain websites were visited by all users, by any user and by all users collectively, respectively.
- **Counting rare diseases:** For scientific or medical purposes it is interesting to know how many cases of rare diseases exist in a country, but it might risk patient privacy if hospitals would publish this highly sparse information. Instead, the multiset sum provides more anonymization, as it is unknown which hospitals treat which and how many patients.

Multi-party privacy-preserving set & multiset operations

4

The field of private multi-party computation offers privacy-preserving alternatives to standard computational operations performed by multiple parties. In the previous two chapters, we explained several operations that can be performed over arbitrary numbers of sets and multisets. In this chapter, we discuss the requirements for protocols to perform such operations in a privacy-preserving manner, where every cooperating party submits a set or multiset.

4.1 Setup 15
Security model 15
Network topology 15
4.2 Correctness requirements . . 16
Exact operations 16
Approximate operations . . . 16
4.3 Privacy requirements 17
Size-hiding 17
Count-hiding 18
Collusion resistance 18
4.4 Terminology 18

4.1 Setup

In this thesis, we are concerned about a group of n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Each party $\mathcal{P}_i$ has a set or multiset X_i, and the goal is to collaboratively compute $Z = X_1 \odot \cdots \odot X_n$, where $\odot$ denotes some operation that is defined over sets or multisets.

In our setup we differentiate between a leader and assistants. The leader is the party that receives the result Z, while the other parties are assistants that only take part in the protocol to facilitate this result. Without loss of generality, we assign the first party $\mathcal{P}_1$ to be the leader, and the remaining parties $\mathcal{P}_i$ for $i = 2, \ldots, n$ are assistants.

Security model

All parties in the protocols are considered to act according to the semi-honest model. That is to say, they follow the protocol at all times, but whenever they can they will try to deduce as much as possible with the information that is available to them. We choose this model over the malicious mode, which considers parties that do not honestly follow the protocol and try to disrupt it. The semi-honest model can be seen as a starting point towards creating protocols that are secure in the malicious model.

Other works might be interested in a setup where every party receives the final result Z, rather than only the leader. In the semi-honest model, this is easy to realize: the leader can simply provide the assistants with the final result. In some cases, and assuming a broadcast medium exists, parties might broadcast the messages that were otherwise sent to the leader to put all assistants in the position to produce the final result for themselves.

Network topology

As mentioned before, a privacy-preserving protocol would limit communication as much as possible. To that end, we characterize a protocol's communication setup by examining which parties must communicate

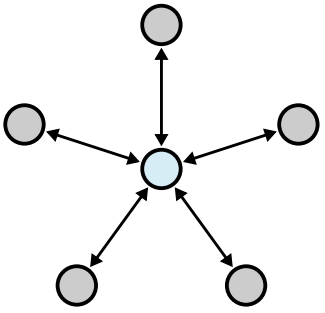


Figure 4.1: Star topology with six parties, the leader is the center node.

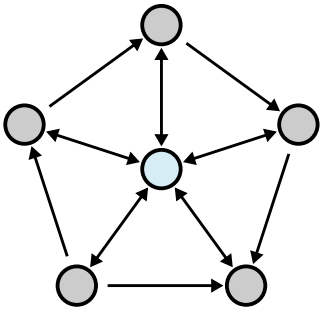


Figure 4.2: 'Wheel' topology with six parties, the leader is the center node.

with each other. Ideally, each assistant would only have to communicate with the leader. This is realized by star-shaped network topology, as seen in Figure 4.1. In the worst case, every party would have to communicate with every other party, which is called a full mesh.

In this thesis we distinguish one more topology that is an extension of the star topology. A 'wheel' topology is a star topology, but in addition, there are one-way communication channels between the outer nodes so that they form a ring. An example of such a topology can be seen in Figure 4.2.

4.2 Correctness requirements

For a semi-honest protocol to be provably privacy-preserving, it must satisfy correctness and privacy requirements [10] . The idea behind the correctness requirement is that the protocol should fail only with negligible probability.

[10]: Lindell (2017), 'How to Simulate It - A Tutorial on the Simulation Proof Technique'

Exact operations

According to our setup, an exact multi-party private set or multiset protocol π is a functionality that receives n party's inputs and outputs the result of the set or multiset operation only to the first party, who is the leader. Formally, the protocol aims to realize the functionality given by $f : (X_1, \ldots, X_n) \rightarrow ((X_1 \odot \cdots \odot X_n), \Lambda, \ldots, \Lambda)$, where Λ is the empty string. For such a protocol to be correct, the probability that its output is different from the result of functionality f is lower than some negligible function $\mu(\kappa)$, where κ is the computational security parameter:

Definition 4.1 (Correctness) *Protocol π satisfies correctness when:*

$$\Pr\left[\text{output}^{\pi}(X_1, \ldots, X_n, \kappa) \neq f(X_1, \ldots, X_n)\right] \leq \mu(\kappa)$$

Additionally, some exact protocols for computing the cardinality of the set resulting from a set operation realize the functionality given by $f : (X_1, \ldots, X_n) \rightarrow (|X_1 \odot \cdots \odot X_n|, \Lambda, \ldots, \Lambda)$.

Approximate operations

An approximate multi-party private set or multiset protocol computes a given set or multiset operation probabilistically, so that there is a high probability that the result is 'close' to the exact result. Still, such a protocol should satisfy the correctness requirement. Its functionality f is different, though.

1: For example, those that are based on Bloom filters, as discussed in Chapter 6.

Superset approximation

Some approximate protocols[1] output a superset of the exact result. In other words, the resulting set or multiset can contain false positives. On the contrary, the result is guaranteed not to contain false negatives. This kind of approximate protocol realizes the functionality given by $f : (X_1, \ldots, X_n) \to (Z, \Lambda, \ldots, \Lambda)$, where $Z \supseteq X_1 \odot \cdots \odot X_n$.

Cardinality estimation

Some approximate protocols[2] for computing the cardinality of the set resulting from a set operation output an estimate of the cardinality, rather than an exact answer. These protocols realize the functionality given by $f : (X_1, \ldots, X_n) \to (Z, \Lambda, \ldots, \Lambda)$, where Z is a stochastic variable for which it holds that the expected value $\mathrm{E}[Z] = |X_1 \odot \cdots \odot X_n|$.

2: For example, those that are based on Bloom filters or sketches, as discussed in Chapters 6 & 8.

4.3 Privacy requirements

Different private set operations protocols satisfy different privacy criteria. All works discussed in this thesis satisfy the correctness requirements above, so parties do not learn anything about elements that do not appear in the protocol's output, such as a party's input. However, some protocols allow for auxiliary information to be known, such as the size of the input sets. To qualify the privacy of a scheme, we use the labels 'size-hiding' and 'count-hiding', where the latter only applies to threshold operations.

Cerulli, Cristofaro, and Soriente [11] also defines the concept of 'reactive privacy' for set intersections, which inhibits parties from enumerating over all possible elements to discover another party's set. However, this concept is restrictive and to the best of our knowledge, there are no multi-party protocols that satisfy this criterion. For the protocols proposed in this thesis, we guarantee both the count-hiding and size-hiding criteria when they are applicable.

Size-hiding

Size-hiding protocols hide the number of elements in each party's set. To be precise, for a maximum input set size k, it is unknown to all parties how many elements were in other parties' input sets. Again, there are exceptional circumstances where it is possible to tell this simply from the result of an operation, for example when a private set intersection returns k elements, meaning that each party also provided those k elements. The term was introduced by Bradley, Faber, and Tsudik [12] . We use the size-hiding tag to denote size-hiding protocols.

[12]: Bradley et al. (2016), 'Bounded Size-Hiding Private Set Intersection'

Count-hiding

A count-hiding protocols hides the number of input sets that contain each element. We only use this term when comparing threshold set intersections and unions. For set unions, for example, this is already asserted by the correctness requirements. We use the count-hiding tag to denote count-hiding protocols.

Collusion resistance

While the stated privacy requirements give a provable indication of a protocol's privacy, they do not cover the full picture. Most notably, they do not take into account colluding parties. Let us consider again the scenario where colleagues want to find a suitable date: Planning a meeting for 10 participants, it is likely that at least some of them are capable of colluding. It is problematic if the protocol's privacy requirements would not hold in that case. For this reason we also emphasize each protocol's collusion resistance, which gives an upper bound for the number of parties that can safely collude, without breaking other party's guarantee for privacy. In this thesis we work with the following definition:

Definition 4.2 (Collusion resistance) *The collusion resistance of an n-party protocol is the maximum number of colluding parties that does not break the protocol's privacy requirements.*

4.4 Terminology

In this thesis we discuss nine common set and multiset operations, based on the operations discussed in the previous two chapters. One of the most studied type of protocols is a Multi-party Private Set Intersection, which is abbreviated as `MPSI`. If one only wishes to find out the cardinality of the set intersection, one would append the abbreviation to get `MPSI-CA`. If one is interested in the threshold intersection, one would prepend the abbreviation to get `T-MPSI`. Table 1.1 contains an overview of the types of protocols discussed in this thesis and their abbreviations.

Set & multiset representations

In the previous part we introduced notation for sets and multisets. While this notation is suitable for reading, cryptographic protocols typically represent their sets and multisets differently in order to perform computations on them. In this part we discuss the most common set representations used in previous works and in our work and how operations can be performed on them.

5 Bitsets

A bitset is a vector containing a bit for every element in the universe, which is set to 1 if the element is contained in the set, and 0 otherwise. For this reason, bitsets are also referred to as bit vectors.

5.1 Creating bitsets 20
5.2 Operations on bitsets 20
Intersections & unions . . . 20
Cardinality 20
Complement 21
Threshold operations 21
5.3 Count vectors 21

5.1 Creating bitsets

We denote the encoding of a set X by $\hat{X}$. For a bitset, this encoding is constructed as follows:

Definition 5.1 (Bitset) *For a set $X \subset \mathcal{U}$, the ith bit of its corresponding bitset $\hat{X}$ is given by:*

$$\hat{X}[i] = \begin{cases} 1 & \text{if } \mathcal{U}_i \in X \\ 0 & \text{otherwise} \end{cases} \quad \text{for} \quad i = 1, \ldots, |\mathcal{U}|$$

Here, we write $\mathcal{U}_i$ to denote the ith element of the universe $\mathcal{U}$.

5.2 Operations on bitsets

The set operations discussed in Chapter 2 translate directly to bitsets using logical operators such as `AND` and `OR`. As a result, a protocol that can privately execute these logical operators can privately compute set operations on bitsets.[1]

1: In fact, in a paper by Aslı Bay, Zekeriya Erkin, Mina Alishahi and Jelle Vos published at SECRYPT 2021, the authors use these operations on bitsets to realize exact multi-party private set operations that scale linearly with the number of parties.

Intersections & unions

To perform a set intersection between n encoded sets from the same universe, one simply performs the logical `AND` operation between each bit of their bitsets. This is illustrated in Figure 5.1. The union operation follows similarly: Instead, one performs the logical `OR` operation in the same fashion.

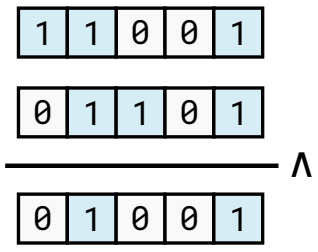


Figure 5.1: The logical `AND` operation applied bit-wise between bitsets results in the bitset encoding of the intersection.

Cardinality

To compute the cardinality of a set encoded as a bitset, one must count the number of bits that are set to 1. Since these bits are either 0 or 1, one can sum the vector to achieve this result. We define the function $\textsc{cardinality}(F) = F$, which takes the number of ones F and outputs the cardinality corresponding to the set represented by the bitset. This function is useful in defining the protocols in Chapter 23.

Complement

Since the complement of set X is the set of elements that are not contained in X, the complement of a bitset is simply the negation of each bit. Suppose that we have an MPSI protocol over bitsets, then using DeMorgan's law, one can use the set complement to implement an MPSU protocol in terms of said MPSI protocol.

Threshold operations

To compute threshold operations on multiple bitsets, the simplest approach is to sum the bitsets together in a bit-wise fashion to get a count vector, as discussed in the next section. Then, one sets every bit in the resulting bitset to 1 if the count is greater or equal to some threshold τ, and 0 otherwise.

5.3 Count vectors

Count vectors are an extension of bitsets to encode multisets. Instead of bits storing 0 or 1, a count vector contains bins storing the multiplicity of its corresponding element. Then, multiset intersections and multiset unions can be realized by taking the minimum value or the maximum value bin-wise between multiple count vectors, respectively. The multiset sum is simply the bin-wise addition of multiple count vectors.

6 Bloom filters

Where bitsets store an exact copy of a set that scales with the number of elements in the universe, Bloom filters store an approximate copy of an arbitrary size. Specifically, Bloom filters distribute elements over a given number of bins that can be significantly smaller. As a consequence, elements might be falsely included in the set, so Bloom filters have a chance of returning false positives.

6.1 Creating Bloom filters 22
Probabilities 22
Parameter selection 23
Hash functions 23
6.2 Operations on Bloom filters 24
Unions 24
Intersections 24
Cardinality 25
6.3 Counting Bloom filters . . . 26

6.1 Creating Bloom filters

Suppose we have a collection of m bins that can be 'set' when they contain a value such as 1, or 'not set' when they are 0. We can map elements to these bins using a hash function that reduces any element to an index of one of the bins. The idea of a Bloom filter is to insert an element by setting the mapped bin of the hash function to 1. The membership of an element can then be queried by checking if its corresponding bin is indeed set. Instead of using one hash function, the Bloom filter is often more accurate when using h hash functions side-by-side; membership of an element then requires all mapped bins to be set.

Probabilities

Assuming that the probability that a bin is set is independent from other bins and h perfect hash functions, so they perfectly distribute elements over the m bins, we express the expected number of false positives when performing k membership queries. We provide two lemmas:

Lemma 6.1 *The probability that a given bin in the Bloom filter is still* 0 *after* N *inserted elements is:*

$$\Pr\left[\textit{bin is } 0\right] \approx \exp\left(-\frac{hN}{m}\right)$$

Proof. The probability that a bin is not set to 1 after an element is inserted is $\left(1-\frac{1}{m}\right)^h$, since any of the h hashes have a $\frac{1}{m}$ probability of selecting that bin. After inserting N elements, that probability is $\left(1-\frac{1}{m}\right)^{hN} \approx \exp\left(-\frac{hN}{m}\right)$. □

Using the identity:

$$\exp(-1) = \lim_{m\to\infty}\left(1-\frac{1}{m}\right)^m$$

What follows, is the Bloom filter's false positive rate ε_N:

Lemma 6.2 *The probability* ε_N *that a random element is included in the set*

encoded by a Bloom filter is:

$$\varepsilon_N \approx \left(1 - \exp\left(-\frac{hN}{m}\right)\right)^h$$

Proof. For a membership query to falsely report a positive result, all h hashes must point to bins containing a 1. Since $\Pr[\text{bin is } 1] = 1 - \Pr[\text{bin is } 0]$, the probability that h uniformly random bins are still set to 1, in other words the false positive rate, is given by $\left(1 - \exp\left(-\frac{hN}{m}\right)\right)^h$. □

We can now approximate the accuracy of the Bloom filter:

Theorem 6.3 *The expected number of false positives after k membership queries on a Bloom filter containing N elements is bounded by:*

$$\mathrm{E}[\mathcal{F}] \approx k\varepsilon_N$$

Proof. Let $Y = \{y_1, \ldots, y_k\}$ be the set of all elements that for which we check membership in the Bloom filter. In the worst case, the Bloom filter falsely claims all of these elements to be contained within it, when none of them are, so the number of false positives $\mathcal{F} \leq k$. Let us say that F_i is a random variable that is 1 when y_i is falsely said to be in the original set, and 0 otherwise. Then, $\mathcal{F} = \sum_{i=1}^{k} F_i$. The expected value $\mathrm{E}[F_i] \approx \varepsilon_N$, so by linearity of expectation, $\mathrm{E}[\mathcal{F}] \approx k\varepsilon_N$. □

Parameter selection

As specified before, a Bloom filter has m bins and h hash functions. To decide on the values of these parameters, it is customary to specify a maximum number of inserted elements N. To ensure that the Bloom filter is practically usable, we need to keep the number of false elements in the filter $\mathcal{F}$ small and at the same time keep the number of bins m as low as possible.

Accordingly, given an expected number of false elements $\mathrm{E}[\mathcal{F}]$, we use Algorithm 1 to select the minimum numbers of bins m for which this expected value is an upper bound, and a corresponding number of hashes h. The algorithm works by counting up, starting from one hash function, and checking which number of hash functions results in the smallest Bloom filter while satisfying the maximum false positive rate ε. One can show that $m = O(k)$ for a fixed ε. We use the upper bound for ε by Goel and Gupta [13]:

$$\varepsilon \leq \left(1 - e^{-\frac{h(N+0.5)}{m-1}}\right)^h . \tag{6.1}$$

Hash functions

None of the protocols in this thesis require Bloom filters to use cryptographically-secure hash functions, since security is provided by cryptographically hiding the bins of the filter. This allows for faster hash functions to be

Algorithm 1 Compact Bloom filter parameters

```
procedure CompactBF(N, ε)
    h ← 1
    m_prev ← ∞
    while true do
        ▷ Compute required filter size m given h using Equation 6.1
        m ← ⌈exp(−h(N+0.5) / ln(1−ε^{h^{−1}}))⌉ + 1
        if m_prev < m then
            ▷ This is the lowest m for any h
            return (h, m_prev)
        h ← h + 1
        m_prev ← m
```

used, so long as they are 'sufficiently' uniform. That is to say, the hash function distributes elements from the universe evenly and without obvious patterns. Many practical systems choose for MurmurHash3 [14] or xxHash3 [15], mmh3 and xxh3 in short.[1]

1: In our implementations we use both mmh3 and xxh3, without preference.

6.2 Operations on Bloom filters

As studied in other works [6, 16], set operations on Bloom filters can be performed similarly to bitsets, with some limitations.

Unions

To perform a set union between n Bloom filters, one applies the logical `OR` operation between respective bins. For this to hold, all Bloom filters must use the same hash functions and must have the same number of bins m. Since the resulting Bloom filter can contain at most nk elements, the false positive rate is given by ε_{nk}. Interestingly, while it is straight-forward to form the Bloom filter encoding the union of n other Bloom filters, it is not straight-forward to decode it when the universe is large.[2] To do so, we need to reverse the Bloom filter. In Chapter 20 we propose a solution to this problem.

2: For example, for $|\mathcal{U}| = 10^6$ and $h = 1$, the naive approach would require a decoder to compute 10^6 calls to the hash function.

Intersections

To perform a set intersection between n Bloom filters, one applies the logical `AND` operation between respective bins. Again, all Bloom filters must use the same hash functions and must have the same number of bins m. While the resulting Bloom filter indeed encodes a superset of the intersection, it is not necessarily the same Bloom filter that would be created if one encodes the actual intersection as a Bloom filter. This is the case, because different elements mapping to the same bins would falsely be included in the resulting Bloom filter [17]. However, since the number of bins that are set to true in the resulting Bloom filter is less than or equal to the number of set bins in any of the input Bloom filters, the resulting false positive rate is still at most ε_k. Decoding the intersection

is not a problem, since the intersection is always a subset of any of the input sets.

Cardinality

Since Bloom filters are approximations, it is only possible to estimate the number of elements that one encodes, rather than exactly. It suffices to count the number of set (filled) bins F and to know the Bloom filter's parameters. Note that the distribution of the number of filled bins given the number of inserted elements N is binomial, following the probability from Lemma 6.2:

$$\Pr\left[\mathbf{F} = F \mid \mathbf{N} = N\right] \sim \mathcal{B}\left(m, \left(1 - \exp\left(-\frac{hN}{m}\right)\right)\right) \tag{6.2}$$

Theorem 6.4 *The expected number of elements encoded by a Bloom filter with F filled bins is given by:*

$$\mathrm{E}\left[N\right] \approx -\frac{m}{h} \ln\left(1 - \frac{F}{m}\right)$$

Proof. Assuming that the hash functions are uniform and independent, the probability for a bin to be 1 is expected to be equal to the ratio of filled bins $\Pr\left[\text{bin is 1}\right] = \frac{F}{m}$. It follows that:

$$\begin{aligned}
\frac{F}{m} &\approx 1 - \exp\left(-\frac{hN}{m}\right) \\
N &\approx \log_{\exp\left(-\frac{h}{m}\right)}\left(1 - \frac{F}{m}\right) \\
&\approx \frac{\ln\left(1 - \frac{F}{m}\right)}{\ln\left(\exp\left(-\frac{h}{m}\right)\right)} \\
&\approx -\frac{m}{h} \ln\left(1 - \frac{F}{m}\right)
\end{aligned}$$

□

For use later, we define the cardinality function for Bloom filters as $\textsc{cardinality}(F) \approx E[N]$. Unfortunately it is hard to give guarantees about this estimation because the reversed conditional probability of Equation 6.2 is not a binomial. In fact, as seen in Figure 6.1, the distribution is asymmetric. The probability is proportional to:

$$\Pr\left[\mathbf{N} = N \mid \mathbf{F} = F\right] \propto \Pr\left[\text{bin is 1}\right]^{F} \Pr\left[\text{bin is 0}\right]^{m-F} \tag{6.3}$$

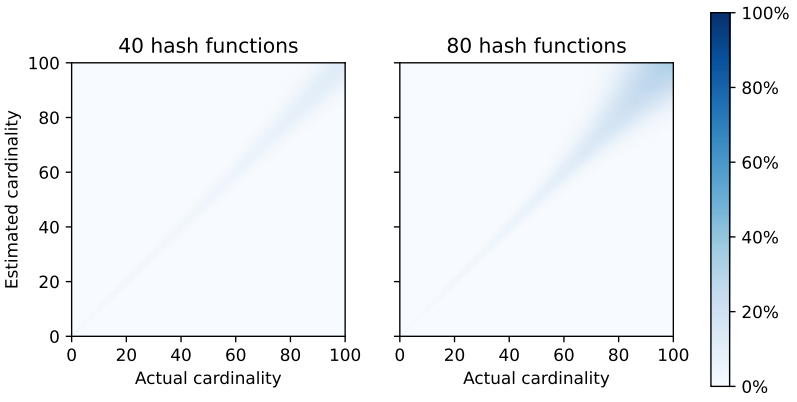


Figure 6.1: Probabilities for estimating the cardinality of a Bloom filter given its actual cardinality. Note that the probability distribution is narrow for lower cardinalities, and wide and more asymmetric for larger cardinalities.

6.3 Counting Bloom filters

We can extend Bloom filters to encode multisets rather than sets. Such a Bloom filter is called a *counting Bloom filter*. Instead of a Boolean, every bin now stores a multiplicity. All the bins start at 0. When inserting an element,the bins selected by the hash functions are increased, but never decreased, to the multiplicity of said element. Then, to check membership of an element, we find the bins that this element is mapped to by the hash functions and return their minimum multiplicity. This means that when at least one of the bins that is mapped to contains 0, the element is not part of the encoded multiset.

7 Polynomial roots

7.1 Polynomial multiset encoding 27
7.2 Operations on polynomials 27
Intersections 27
Unions 28
Multiset sums 28
Reduce by one 28
Extracting elements 28

While bitsets and Bloom filters consist of bins and are based on logic operations, one can also encode sets in the roots of polynomials, on which operations are defined by arithmetic. Actually, polynomials encode multisets. To prevent confusion with the polynomial variable x, in this chapter we denote sets and multisets by S rather than X.

7.1 Polynomial multiset encoding

One can encode a set of integers S in the roots of a polynomial $\hat{S}[x]$ using:

$$\hat{S}[x] = \prod_{s \in S} (x - s) \tag{7.1}$$

The roots of a polynomial form a multiset. So, we can generalize the statement above by interpreting sets as multisets to the following definition of the polynomial encoding:

Definition 7.1 (Polynomial encoding) *The polynomial encoding multiset S with multiplicity function $\mathcal{M}_S(s)$ is given by:*

$$\hat{S}[x] = \prod_{s \in S} (x - s)^{\mathcal{M}_S(s)}$$

7.2 Operations on polynomials

Because polynomials are not made up of bins, and since they are actually encoding multisets, the intersection, union and other operations follow differently than for bitsets and Bloom filters. In our work, we do not use this encoding, but it forms the basis of many previous works.

Intersections

The intersection between n sets encoded as polynomial roots $\hat{S}_1, \ldots, \hat{S}_n$ is formally given by the polynomial equivalent of the greatest common divisor $\gcd\left(\hat{S}_1, \ldots, \hat{S}_n\right)$. However, Kissner and Song [18] prove that it is sufficient to randomize the polynomials by multiplying them with random polynomials and then summing them, without revealing more information than what would be known from the greatest common divisor. More specifically, they show that:

$$r_1\hat{S}_1 + \cdots + r_n\hat{S}_n = u \gcd\left(\hat{S}_1, \ldots, \hat{S}_n\right) \tag{7.2}$$

Here, $r_1, \ldots, r_n$ are the aforementioned random polynomials and u is some uniformly random polynomial. The roots contained in u are false positives, but since they are uniformly distributed over the entire space of the roots, there is a negligible probability that the roots also appear in one of the input sets S_i, so it is unlikely they will be wrongly interpreted as being part of the intersection. This operations works on encoded sets and multisets alike.

Unions

The union between n sets encoded as polynomial roots can be realized using the least common multiple $\operatorname{lcm}\left(\hat{S}_1, \ldots, \hat{S}_n\right)$. Seo, Cheon, and Katz [19] show that this operation too can be achieved through addition when using a rational polynomial representation $\frac{r_i}{\hat{S}_i}$, for a random polynomial r_i without leaking any more information than when computing the least common multiple. Specifically, they show that:

$$\frac{r_1}{\hat{S}_1} + \cdots + \frac{r_n}{\hat{S}_n} = \frac{u}{\operatorname{lcm}\left(\hat{S}_1, \ldots, \hat{S}_n\right)} \tag{7.3}$$

The authors show that rational polynomials can be efficiently manipulated using reversed Laurent series. Similarly to intersections, with overwhelming probability the numerator and denominator do not have any roots in common, so the randomization is unlikely to affect the final result. This operation also works on multisets.

Multiset sums

By the definition of multiset sums and the polynomial encoding, the multiset sum between n polynomial set encodings $\hat{S}_1, \ldots, \hat{S}_n$ is simply their product:

$$\prod_{s \in S_1} (x - s)^{\mathcal{M}_{S_1}(s)} \cdots \prod_{s \in S_n} (x - s)^{\mathcal{M}_{S_n}(s)} = \prod_{s \in S_1 \uplus \cdots \uplus S_n} (x - s)^{\mathcal{M}_{S_1 \uplus \cdots \uplus S_n}(s)} \tag{7.4}$$

Reduce by one

It is possible to reduce the multiplicity of all roots by one by computing the polynomial's derivative. This operation is useful for computing threshold operations. For example, by taking the derivative three times, we are left with a polynomial encoding of the elements that appeared at least four times in the multiset.

Extracting elements

Differently from integers, polynomials can be factored in polynomial time, after which one can easily extract the roots and their corresponding multiplicity. One common algorithm for polynomial factoring is the algorithm by Cantor and Zassenhaus [20].

8 Other set representations

8.1 Garbled Bloom filters 29
8.2 Min-max sketches 29

While the three previously discussed set and multiset representations are common in multi-party private set operations, many other types of representations were used in previous works. Note how the previously discussed representations were versatile in that there are simple ways to perform intersections, unions and other operations on between encoded sets or multisets. In this chapter we highlight two other representations that generally only serve one purpose. Additionally, some works [21] do not use a specific set encoding. Instead, the elements are individually encoded.

8.1 Garbled Bloom filters

Dong, Chen, and Wen [22] introduced garbled Bloom filters. While in a regular Bloom filter we check if the bins selected by the hash functions are set, in a garbled Bloom filter we perform an `XOR` operation between those bins to examine if the result is some well-formed value. Instead of storing 1 or 0, every bin contains a λ-bit value. Specifically, a garbled Bloom filter can encode a mapping between key-value pairs (x, y). Kolesnikov et al. [5] describe how to construct the filter as follows:

1. Initialize all bins to $\perp$.
2. For each key-value pair (x, y), fill the bins selected by the hash functions $\mathcal{H}_1(x), \ldots, \mathcal{H}_h(x)$ that still contain $\perp$ with random values so that $\text{GBF}[\mathcal{H}_1(x)] \oplus \cdots \oplus \text{GBF}[\mathcal{H}_h(x)] = y$.
3. Replace bins that still contain $\perp$ with a random value.

One can compute intersections using garbled Bloom filters by encoding each set, so that the when queried for an element from the set, the result is 0. Then, when the garbled Bloom filters are combined using an `XOR` operation, the resulting filter will return 0 with overwhelming probability for only those elements in the intersection. The values of the filter appear to be randomly distributed.

8.2 Min-max sketches

Dong and Loukides [23] propose a protocol based on min-max sketches [24]. A min-max sketch works using h functions and two vectors of h bins. For each bin in the first vector, a party computes the hash of all elements in the input set using the corresponding hash function, and fills the bin with the element with the lowest hash. For the second vector, the party fills each bin with the element with the highest hash. Then, the probability that a bin in two min-max sketches for sets S_1 and S_2 contains the same element is equal to the Jaccard index, defined as $\frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$. By doing this for all bins, the min-max sketch is an unbiased estimator for the Jaccard index. By knowing the intersection-cardinality, one can estimate the union-cardinality, and vice versa.

Cryptographic building blocks

In the previous part we discussed the most common set and multiset representations, and how we can perform operations on them. However, typically these operations cannot be performed privately without the use of cryptographic building blocks. In this chapter we go through common cryptographic building blocks used in previous works, as well as those that form the basis of our proposed protocols. We also discuss what hardness assumptions they rely on. At the end of the chapter we provide an overview of the notation used in the remainder of this thesis.

Hardness assumptions & elliptic curves

9

9.1 Decisional Diffie-Hellman . . 31
9.2 Elliptic curve groups 31
Efficient elliptic curves . . . 32
Elliptic curve co-factors . . . 32
9.3 Pairing-based cryptography 32
Pairing function 33
Hardness assumptions 33
Implementations 33

In our proposed protocols we make heavy use of elliptic curves and some rely on forms of the decisional Diffie-Hellman assumption. In this chapter we present a high-level introduction to these topics.

9.1 Decisional Diffie-Hellman

Consider a cyclic group $\mathbb{G}$ and a generator g. We use the multiplicative notation for this group, so the entire group can be generated by g^i for $i = 1, \ldots, |\mathbb{G}|$. Now, let us say we are given an element that corresponds to $g^r \in \mathbb{G}$, for some randomly chosen but unknown r, and we are tasked to find this r. From studying the real numbers we might be tempted to say that this problem is easily solved by taking the logarithm. However, for some sufficiently large cyclic groups, this problem turns out to be exceedingly hard. In fact, for this reason, this problem is called the Discrete Log Problem (DLP), and it forms the basis of countless cryptographic techniques:

Definition 9.1 (Discrete Log Problem) *The DLP states that it is computationally infeasible to determine some random r given only g and g^r.*

The Decision Diffie-Hellman (DDH) is a stronger assumption than the DLP because it implies the DLP. Since these assumptions are used for cryptographic purposes, they are also referred to as hardness assumptions. Multiple protocols for multi-party private set operations are based on this assumption [6, 25]:

Definition 9.2 (Decisional Diffie-Hellman) *The DDH assumption states that given g^a and g^b for some random $a, b \in \mathbb{Z}_{|\mathbb{G}|}$, $g^a b$ is computationally indistinguishable from some random $r \in_R \mathbb{G}$.*

9.2 Elliptic curve groups

Among such groups where the DDH assumption holds are cyclic subgroups of elliptic curves over finite fields. These elliptic curves form the basis of many modern cryptographic protocols that are at the core of the internet, such as for key agreement schemes and digital signatures [26]. The reason for this is that elliptic curve groups can be significantly smaller for the same level of security, which typically allows for smaller key sizes and depending on the operation, for faster computations. Specifically, to achieve 128 bits of security, NIST recommends 3072 bit keys for factoring modulus-based public key cryptosystems such as Paillier, and only 256 bit keys for elliptic curve-based cryptography [26].

Efficient elliptic curves

Much research has gone into finding types of elliptic curves that allow for efficient arithmetic on computers. One of the most efficient types of curves is the Montgomery curve [27] . A Montgomery curve over a finite field F is given by the following equation:

[27]: Bernstein et al. (2021), *Genus-1 curves over large-characteristic fields*

$$By^2 = x^3 + Ax^2 + x \tag{9.1}$$

That is, any point that lies on this curve parameterized by $A, B \in F$ is an element of the elliptic curve group $\mathbb{G}$. The curve describes an operation that we usually denote in additive notation, which is called the group law or the group operation. This operation takes two curve points P and Q, and returns a curve point $P + Q = R$. The implementation differs per type of curve group, but in essence, the idea behind the operation is to 'draw a line' through P and Q, where R is the point where this line crosses the curve again. If the line does not cross the curve again, we say that R is the point at infinity $\mathcal{O}$, which is the identity of the curve group.

It is also possible to perform the group operation on a point P with itself, which is commonly referred to as point doubling. The slope of the line through P is in this case the derivative of the curve equation at P. This methods allows us to compute $P + P = 2P$. As such, by carefully choosing which points to double, we can efficiently perform the equivalent of multiplication using repeated addition. For example, we can compute $12P = 6P + 6P$, where $6P = 2P + 2P + 2P$ and $2P = P + P$, using just four calls to the group operation, rather than twelve.

Note that in this work we use the additive notation in some places, and the multiplicative notation in others. In the multiplicative notation, the group operation is denoted by $PQ = R$, and point doubling is denoted using an exponent, such as P^{12}.

Elliptic curve co-factors

In this work, we use the Montgomery curve called Curve25519 [28]. This curve has a co-factor of 8, which means that the prime order subgroup that we actually use in cryptographic applications is one eight of the size of the total group. However, when implementing protocols, it is possible to introduce bugs related to these co-factors. For this reason, we use a highly-optimized abstraction that allows us to use a curve with a co-factor to realize a prime-order group [29, 30], so there is no co-factor. Additionally, this technique allows for faster equality checks.*

9.3 Pairing-based cryptography

Pairing-based cryptography concerns three groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, so that there exists an efficiently-computable bilinear mapping function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, which is also called the pairing function. Here, groups

* https://ristretto.group/details/equality.html

$\mathbb{G}_1$ are $\mathbb{G}_2$ written in additive notation, while we write group $\mathbb{G}_T$ in multiplicative notation.

Pairing function

For two generators $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, and two numbers a and b it must hold that $\mathrm{e}(aP, bQ) = \mathrm{e}(P, Q)^{ab}$. Additionally, the pairing function can never evaluate to 1. This is a powerful functionality that allows us to essentially perform multiplication of the numbers a and b in the exponent of the generator $\mathrm{e}(P, Q)$ in group $\mathbb{G}_T$. In Section 18.3 we explain its use in non-interactive secret sharing.

Hardness assumptions

Note that using the pairing function, we can break the DDH assumption: If we have $aP \in \mathbb{G}_1$, $bQ \in \mathbb{G}_2$ and we can decide whether an element is either $g^{ab} \in \mathbb{G}_T$ or a random element. Instead, there is the Bilinear Diffie-Hellman (BDDH) assumption, which says that when we add one more tuple, such as cP or cQ resulting, g^{abc} is indeed computationally indistinguishable from randomness.

Currently bilinear pairings are only realized using specifically-chosen elliptic curves that we refer to as pairing-friendly curves. For several well-studied elliptic curves such as the Tate pairing it is believed that the Decisional Bilinear Diffie-Hellman (BDDH) assumption holds [31].

Implementations

In this work, we use the Barreto-Naehrig curve [32]. Without going into detail, there are parameters for this curve that satisfy 128 bits of security. Additionally, there exist hash-to-curve schemes [33] that allows us to hash arbitrary strings to curve points, in our case to group $\mathbb{G}_2$, without revealing the discrete logarithm of that element. This allows us to use bilinear pairings as a pseudo-random function using a *nonce*; a number used only once. We explain this in more detail in Chapter 18.

Secret sharing methods 10

10.1 (n, n)-secret sharing 34
Integers modulo x 34
Coprimes modulo x 35
10.2 (t, n)-secret sharing 35

In the previous chapter we discussed elliptic curve groups that relied on certain hardness assumptions for their security. In this chapter, we describe secret sharing methods, which are information-theoretically secure. That is, provided that private information stays private, the secret shares are unconditionally secure: the best attack that exists is to randomly guess. To make sure that the probability of guessing correctly is negligible, it is customary to retain at least 40 bits of statistical security [5]. As a result, the probability of guessing correctly is less than 2^{-40}.

Consider a party who wants to share a password with three friends, so that if necessary, the friends can work together to retrieve the password, but otherwise it stays hidden. This is the premise behind secret sharing. In this case, we say that the secret password is 3-shared: It is split into three shares that each reveal nothing about the secret until they are combined. In this chapter we discuss schemes where a secret is shared among all n parties so that they all have to collaborate to recover the secret, and those where a secret is shared among all n parties but only a subset of t parties is required to recover it.

10.1 (n, n)-secret sharing

In an (n, n)-secret sharing setup, we have n parties who all need to cooperate to recover a shared secret. In this section we discuss the most commonly used additive and multiplicative secret sharing schemes over the integers. In Chapter 18 we discuss another (n, n)- secret sharing setup that is particularly suited to our applications.

Integers modulo x

The integers modulo an integer x form a group that is closed under addition, containing all integers $[0, x)$. Suppose that we want to share the secret value 0 among the n parties using this group. Then, we can choose n uniformly chosen values that collectively sum to 0.

Example 10.1 Consider a group with $x = 256$ and $n = 3$ parties, then the secret 0 could be shared like:

$$v_1 + v_2 + v_3 \equiv 0 \pmod{256}$$
$$169 + 4 + 83 \equiv 0 \pmod{256}$$

Clearly, v_1 and v_2 can be chosen completely at random, and only v_3 is specifically chosen. As such, $n - 1$ secret shares do not reveal any information.

Coprimes modulo x

The coprimes modulo an integer x also form a group, but it is closed under multiplication rather than addition. Importantly, while the previous group contained all numbers from 0 until $x-1$, the group of coprimes does not satisfy this property. Rather, the number of elements in this group is decided by Euler's totient function $\phi(\_)$. For a prime number p, $\phi(p) = p-1$, so when x is prime, the multiplicative group contains the elements $[1, x-1)$. One can check that these are indeed all coprime to x.

Example 10.2 Let us take $x = 41$ and $n = 3$, and we wish to share 1, then it could be shared like:

$$v_1 \cdot v_2 \cdot v_3 \equiv 1 \pmod{41}$$
$$15 \cdot 8 \cdot 27 \equiv 1 \pmod{41}$$

10.2 (t, n)-secret sharing

In a (t, n)-secret sharing scheme, it suffices for t parties to cooperate to recover the secret. One of the most common schemes is Shamir's secret sharing [34] . This scheme works by generating a t-degree polynomial, so that when it is evaluated at its origin 0, it returns the secret. As such, the polynomial has the form $s + ax + bx^2 + \ldots$, where s is the secret and a and b are random coefficients. The key behind this scheme is that when a party has t points of the polynomial, they can perform Lagrange interpolation [35] to recover the polynomial's coefficients, and thereby the encoded secret.

[34]: Shamir (1979), 'How to Share a Secret'

Example 10.3 Consider a modulus modulus $x = 40$, a $(2, 3)$-setup and a secret 5. One realization of a random 2-degree polynomial for Shamir's secret sharing is:

$$5 + 31x + 16x^2$$

Let us evaluate this polynomial at $x = i$ for v_i, then $v_1 = 12$, $v_2 = 11$ and $v_3 = 2$. One can check that for two of these secret shares, there is only one 2-degree polynomial that fits these points.

11 Homomorphic cryptosystems

11.1 ElGamal cryptosystem 36
Exponential ElGamal 37
ElGamal over elliptic curves 37
Threshold ElGamal 37
(n, n)-key generation 37
11.2 Paillier cryptosystem 38
11.3 Decryption to zero 38

Public Key Encryption (PKE) is a tool that allows any party to encrypt data with a public key so that it can only be decrypted by the private key holder. We refer to such encrypted data as a ciphertext. However, some situations require that no one party can decrypt a ciphertext alone; instead they require a given number of parties to do so collaboratively. Threshold cryptosystems realize such behavior. Specifically, a (t, n)-cryptosystem requires t out of n parties to collaborate in order to decrypt a ciphertext.

The data that is encoded by a ciphertext is called a plaintext. Some cryptosystems create ciphertexts that are malleable; users can manipulate such a ciphertext to achieve another valid ciphertext that relates in some way to the original plaintext. In this thesis we are particularly concerned about partially homomorphic cryptosystems, which permit users to perform either addition or multiplication on the encrypted plaintexts, by exploiting the ciphertexts' malleability. We will refer to a threshold cryptosystem that is also partially homomorphic as a homomorphic threshold cryptosystem.

11.1 ElGamal cryptosystem

The ElGamal cryptosystem allows the use of any group $\mathbb{G}$ in which the DDH assumption holds [36] . Depending on how the group operation is defined, the cryptosystem is either additive or multiplicative, but we choose to write it in multiplicative notation here. The four standard methods for the ElGamal cryptosystem are:

[36]: ElGamal (1984), 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'

- **Setup**: An authority chooses a group $\mathbb{G}$ of order q and a generator g in $\mathbb{G}$.
- **KeyGen**: The dealer chooses a secret key $sk \in_R \mathbb{Z}_q$. The public key is $pk = g^x$.
- **Encrypt**: To encrypt an element $M \in \mathbb{G}$, a party computes $\langle \alpha, \beta \rangle = \langle g^y, M\ pk^y \rangle$ using randomness $y \in_R \mathbb{Z}_q$.
- **Decrypt**: To decrypt $\langle \alpha, \beta \rangle$, party $\mathcal{P}_i$ computes $M = \beta \alpha^{-sk}$.

Here, **Setup** is a method to select the high-level cryptosystem parameters. After that, a dealer can use **KeyGen** to generate keys. Then, for any message encrypted by **Encrypt**, **Decrypt** returns the original message corresponding to the ciphertext.

When we perform ElGamal over a multiplicative group of integers, the scheme is multiplicatively homomorphic. That is, multiplying two ciphertexts $\langle \alpha_1, \beta_1 \rangle$ and $\langle \alpha_2, \beta_2 \rangle$ together, we get a new ciphertext $\langle \alpha_1 \alpha_2, \beta_1 \beta_2 \rangle$ encoding the product of the original messages. This property also allows a user to re-randomize a ciphertext. To do so, they multiply the original ciphertext with a fresh ciphertext of the identity element.

Exponential ElGamal

Exponential ElGamal is a variant of ElGamal that enables additive homomorphism in an otherwise multiplicative group. Instead of encrypting a message M directly, we encrypt g^M. The product of two such elements g^{M_1} and g^{M_2} results in $g^{M_1+M_2}$. On the downside, finding the resulting message in the exponent requires computing the discrete logarithm. Fortunately, if we know that this message is in a small range, we can pre-compute a lookup table and traverse it.

ElGamal over elliptic curves

Since ElGamal is secure for any group for which the DDH assumption holds, we can use elliptic curves to more efficiently implement this cryptosystem. However, this means that the messages we encrypt are also elliptic curve points. In that sense, if we want to encrypt integer messages, we arrive at a similar setup as exponential ElGamal. We provide more details about this construction in Chapter 23.

Threshold ElGamal

One can use Shamir's secret sharing to transform ElGamal into a threshold cryptosystem. This technique can be used to transform some other cryptosystems as well, such as Paillier, which we discuss in the next section. We demonstrate this technique for ElGamal:

- **Setup**: An authority chooses a group $\mathbb{G}$ of order q and a generator g in $\mathbb{G}$.
- **KeyGen**: The dealer chooses a polynomial $f(x) = mk + a_1 x + \cdots + a_{t-1}x^{t-1} \in \mathbb{Z}_q[x]$, where $mk \in_R \mathbb{Z}_q$ is the master key. The public key is $pk = g^x$. The dealer generates partial secret keys $sk_i = f(i)$ for $i = 1, \ldots, n$.
- **Encrypt**: To encrypt an element $M \in \mathbb{G}$, a party computes $\langle \alpha, \beta \rangle = \langle g^y, M\ pk^y \rangle$ using randomness $y \in_R \mathbb{Z}_q$.
- **Decrypt**: To partially decrypt $\langle \alpha, \beta \rangle$, party $\mathcal{P}_i$ computes $D_i \leftarrow \alpha^{sk_i}$.
- **Combine**: To complete decryption of $\langle \alpha, \beta \rangle$, a party combines partial decryptions D_i of at least t parties through Lagrange interpolation [35].

In the threshold cryptosystem, the **Decrypt** method changes and we gain the **Combine** method. **Decrypt** now performs partial decryptions. After all, a single party cannot decrypt. The **Combine** method is used for the final user to combine t partial decryptions into the original message.

(n, n)-key generation

Debnath et al. [37] use a threshold version of ElGamal where the threshold is n, so every party must be involved to decrypt. The benefit of their setup is that they do not use Lagrange interpolation, and the keys can be generated in a distributed fashion, without the need of a trusted dealer and without expensive computations.

11.2 Paillier cryptosystem

The Paillier cryptosystem [38] is an additively-homomorphic public key encryption scheme based on the Decisional Composite Residuosity (DCR) assumption over integers. While the original scheme generates one secret key and one public key, the scheme can be altered using the principal behind Shamir's secret sharing in the same way as above to form an additive threshold cryptosystem.

[38]: Paillier (1999), 'Public-Key Cryptosystems Based on Composite Degree Residuosity Classes'

Paillier ciphertexts are additively homomorphic, so one can add encrypted integers together and multiply them with constants. As a consequence, we can use Paillier to obliviously evaluate encrypted polynomials by their coefficients, among others. Generally, generating Paillier keys requires a trusted dealer or an expensive distributed key generation protocol that lowers collusion resistance to at most $\lfloor \frac{n}{2} \rfloor$ [39].

11.3 Decryption to zero

Regular decryption of a ciphertext returns the message encoded within it. However there are cases where we do not want to learn the actual message, but only whether it contains the identity element or not. Since in Paillier and exponential ElGamal this element is 0, such a protocol is called a decrypt-to-zero protocol. Typically, in such a protocol, all decrypting parties multiply the ciphertext with a random number, so that when combined, the result is either 0 or a random number based on all parties' random numbers. Importantly, parties can do so in parallel, rather than sequentially, so it can be merged into the partial decryption operation in a threshold cryptosystem.

Summary of preliminaries 12

12.1 Notation

In Table 12.1 we give a summary of the notation used throughout this thesis. The list does not contain all symbols, but it does contain those that we use consistently. We prevent duplicate symbols as much as possible, but in some cases the meaning of a symbol is different depending on its context.

Table 12.1: Description of a selection of symbols in this work.

Symbol	Description
	Sets
n	Number of parties
$\mathcal{P}_i$	Party i
x_i	Party i's input bit
X_i	Party i's input set or multiset
k	Maximum set size, so $k \geq \lvert x_i \rvert$
$\mathcal{U}$	Universe of elements
τ	Threshold of threshold operations
$\hat{X}_i[j]$	Bin j of party i's set representation
F	Number of filled bins
	Multisets
$\mathcal{M}(\_)$	Multiplicity function
W'	Maximum multiset size, so $W' \geq \lvert x_i \rvert$
M'	Maximum multiplicity
W	Maximum resulting multiset size
M	Maximum resulting multiplicity
	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
$\mathcal{H}$	Some hash function
ε	Error rate of a membership query
$\mathcal{F}$	Number of false positives
	Pairing-based Cryptography
pk_i	Public key of party i
sk_i	Secret key of party i
$\mathcal{H}_2(\_)$	Hash onto a random point in $\mathbb{G}_2$
$e(\_,\_)$	Pairing function
	Security
π	A protocol
$\overset{c}{\equiv}$	Computationally indistinguishable
κ	Computational security parameter
λ	Statistical security parameter
Λ	Empty string
	Interactive protocols
#	Number of communication stages
π_i	Party i's permutation function
$\langle \alpha, \beta \rangle$	ElGamal ciphertext components
D_i	Party i's partial decryption

12.2 Hardness assumptions

In our previous work chapters, we abbreviate the hardness assumptions on which protocols rely. Table 12.2 contains a summary of those abbreviations that appear in this work.

Short	Name	Application
DCR	Decisional Composite Residuosity	Paillier cryptosystem
QDR	Quadratic Residuosity Assumption	Goldwasser-Micali cryptosystem
TDP	Trapdoor Permutations	Oblivious transfer
DDH	Decisional Diffie-Hellman	ElGamal cryptosystem
SI-DDH	Short Interval DDH	Polynomial ElGamal [7]
BDDH	Bilinear DDH	Asymmetric construction
PRF	Pseudo-Random Function	Symmetric construction
PRP	Pseudo-Random Permutations	Feather protocol [40]
CKHF	Commutative Keyed-Hash Function	Vaidya and Clifton [21]

Table 12.2: Descriptions and abbreviations of relevant hardness assumptions.

Previous work

In this part we provide a comprehensive comparison between numerous works on multi-party private set and multiset operations. Even still, there exist many more papers discussing variants of the aforementioned operations. However, we aim to cover the most common operations and include important works as well as recent work. We do not claim to give an exhaustive overview.

13 Multi-party private set intersections

Compared to other operations, multi-party private set intersection protocols make up the majority of MPSO works. In this chapter we give an overview of prevalent works in this area. It turns out that many works fulfill the same objective, but using different means. For example, some use secret sharing while others use homomorphic cryptosystems, but the operations are generally the same. For this reason we go through the protocols based on their set representation and which objective they fulfill. Note that in this thesis we only consider multi-party protocols, so we omit two-party protocols from the comparison. The protocols discussed here are all exact or they are superset approximations.

In addition, Table 13.1 gives a summary of these protocol's properties. Under topology, the table refers to those network topologies discussed in Chapter 4. By # we refer to the number of stages that a protocol requires, where we define a stage as a part of the protocol where every party should be online at some point to be able to proceed to the next stage. The parameters κ and λ are the computational and statistical security parameters, respectively, so they depend on the security assumptions of each scheme. The asymptotic complexities marked with * are ones that we adapted from the original works, see Appendix A. To conclude, at the end of this chapter we summarize our general findings in comparing these works.

13.1 Polynomial roots-based MPSI 42
Randomized polynomial sum 43
Oblivious evaluation 43
13.2 Bitset-based MPSI 44
Logical operations on bitsets 44
13.3 Bloom filter-based MPSI . . 44
Logical operations on filters 44
Counting Bloom filters . . . 45
13.4 Other MPSI 45
Garbled Bloom filters 46
Individual element encodings 46
13.5 Summary 46

Table 13.1: Overview of prevalent works proposing MPSI protocols, including asymptotic complexities and security properties.

Work		Communication				Computation		Security			Privacy
Ref.	Year	Topology	Leader	Assistant	#	Leader	Assistant	Collusion	Assumption	Mal?	Size
General MPC		Mesh	$\widetilde{O}(nk\lambda)$*	$\widetilde{O}(nk\lambda)$*	-	-	-	$n-1$	-	-	-
[41]	2004	Mesh	$O(nk\lambda)$*	$O(nk\lambda)$*	3	$O(nk\kappa)$*	$O(nk\kappa)$*	$n-1$	DCR		
[18]	2005	Wheel	$O(nk\lambda)$*	$O(tk\lambda)$*	4	$O(tk^2\kappa)$*	$O(tk^2\kappa)$*	$n-1$	DCR	✓	✓
[42]	2007	Mesh	$O(n^2k^2\lambda)$*	$O(n^2k^2\lambda)$*	3	$O(nk^2\lambda)$*	$O(nk^2\lambda)$*	$\lfloor\frac{2}{3}n\rfloor$	none	✓	✓
[43]	2012	Star	$O(ntm\lambda)$*	$O(nm\lambda)$*	3	$O(nm\kappa)$*	$O(nm\kappa)$*	$n-1$	QDR	✓	✓
[25]	2012	Mesh	$O(n^2k\lambda)$*	$O(n^2k\lambda)$*	3	$O(nk\kappa)$*	$O(nk\kappa)$*	$\lfloor\frac{1}{2}n\rfloor$	DDH	✓	✓
[6]	2015	Star	$O(tm\lambda)$*	$O(m\lambda)$*	2	$O(nm\kappa)$*	$O(m\kappa)$*	$n-1$	DDH		✓
[44]	2017	Star	$O(nk\lambda)$	$O(k\lambda)$	2	$O(nk^2\kappa)$ *	$O(k\kappa)$*	$n-1$	DCR	✓	
[5]	2017	Mesh	$O(nk\lambda)$	$O(tk\lambda)$	2	$O(n\kappa)$	$O(t\kappa)$	$n-1$	TDP		
[45]	2018	Mesh	$O(nm\lambda)$*	$O(nm\lambda)$*	2	$O(nm\lambda)$*	$O(nm\lambda)$*	$n-1$	TDP		✓
[40]	2020	Star	$O(nk\lambda)$*	$O(k\lambda)$*	2	$O(nk\lambda)$*	$O(k\lambda)$*	1	PRP		✓
[37]	2021	Star	$O(nk\lambda)$*	$O(k\lambda)$*	2	$O(nkh\kappa)$*	$O(k\kappa)$*	$n-1$	DDH		
Exact (Ours)		Star	none	$O(\lvert\mathcal{U}\rvert\lambda)$	1	$O(nk\kappa)$	$O(\lvert\mathcal{U}\rvert k+nk\kappa)$	$n-2$	PRF		✓
Approx. (Ours)		Star	none	$O(k\lambda)$	1	$O(nk\kappa)$	$O(nk\kappa)$	$n-2$	PRF		✓

13.1 Polynomial roots-based MPSI

We first consider those protocols that encode elements as the roots of polynomials. Since polynomials can be factored in polynomial time, all these protocols either encrypt or use other methods to hide each party's polynomial until all polynomials are combined.

Randomized polynomial sum

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomials $S_1(x), \ldots, S_n(x)$ and pick uniformly random polynomials $r_1(x), \ldots, r_n(x)$ from the same degree to compute $S_1(x)r_1(x) + \cdots + S_n(x)r_n(x)$.

Kissner and Song [18] published one of the first work describing an MPSI protocol. The key idea is to sum the polynomials that encode sets to compute the intersection as in Equation 7.2, and to randomize each polynomial to lower the probability of false positives and simultaneously hide the original encoding. The protocol uses an additively homomorphic threshold cryptosystem to privately aggregate the polynomials; the authors recommend the threshold Paillier cryptosystem.

Similarly, Li and Wu [42] use Shamir's secret sharing scheme to fulfill the same objective, which allows to disregard the hardness assumptions that homomorphic cryptosystems make at the cost of efficiency.

The later work by Cheon, Jarecki, and Seo [25] uses exponential ElGamal rather than Paillier, which allows for distributed key generation. While the decryption algorithm does not return the actual message m, but rather an element g^m, this is not a problem because for every element x_i in their set, a party can check if g^{x_i} corresponds to g^m.

Due to the properties of polynomials, these protocols should also be able to function as multiset intersections.

Oblivious evaluation

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomial $S_1(x), \ldots, S_n(x)$ and let $\mathcal{P}_1$ obliviously evaluate them, computing $rP_i(x) + y$ with randomness r to decrypt to payload y when x was included in the set.

Hazay and Venkitasubramaniam [44] describe an approach in which the leader performs a two-party PSI with each other party and concludes the intersection by aggregating the results. In more detail, the other parties encode their sets as encrypted polynomials using an additively homomorphic threshold cryptosystem and the leader evaluates it for each of its elements and multiplies the results with randomness. This means that the two-party intersection returns an encrypted result to the leader for each of its elements, encoding 0 when the considered element is in the intersection and randomness otherwise. Finally the leader can collaborate with the other parties to decrypt-to-zero and find which elements are indeed in the intersection. In this case the payload y in the objective is 0 because the leader can map the ciphertexts to its elements using their index and this makes it possible to decrypt-to-zero.

The work by Freedman, Nissim, and Pinkas [41] from 2004 describes a different scheme that uses the same payload-encoding private matching technique. Apart from the leader all parties again encode their sets as encrypted polynomial roots, after which the leader performs the private matching payload-embedding to insert a payload y_i for each

other party so that they sum to the element $x = \oplus_i y_i$. The parties decrypt this ciphertext. To prevent the leader from recognizing which party has which elements they mask the results so that the masks `XOR` to 0 when fully aggregated. After aggregation the leader can tell which elements were in the intersection by checking if the resulting y equals the original element x.

Sang and Shen [46] propose a protocol that fulfills the same objective, but as randomness they compute a hash of the encrypted values. The polynomial that is evaluated is the randomized polynomial sum described in the previous objective. The authors do not choose a specific additively homomorphic cryptosystem for the semi-honest model, since the work focuses on a scheme that is secure in the universal composability model, which means that it can be used provably securely in any combination of protocols. For this reason we do not mention this work in Table 13.1.

By default, all of these protocols do not hide the size of the leader's set as described in Section 4.3. This fact does make them more communication efficient, since there is only one encryption per item in the leader's set and communication is strictly in a star topology.

13.2 Bitset-based MPSI

Logical operations on bitsets

Objective: Let parties $\mathcal{P}_1, ..., \mathcal{P}_n$ encode their sets as encrypted bitsets and compute the logical `AND` between all of them.

In *Multi-party Private Set Intersection Protocols for Practical Applications*, Bay et al. propose an MPSI protocol using bitsets. To encrypt the bits, they use the threshold Paillier cryptosystem. Since bitsets are exact set encodings, the result is exact, but when the universe grows too large the operations become too long to be practical.

13.3 Bloom filter-based MPSI

Since Bloom filters store an approximate copy of a set, they reduce communicational complexity by limiting the number of transmitted elements. However, protocols must typically encrypt the filters, because one can trivially check if it contains a specific element. Encrypting each bin separately, though, means that there is a trade-off between the accuracy of the filter and the runtime and bandwidth of the protocol.

Logical operations on filters

Objective: Let parties $\mathcal{P}_1, ..., \mathcal{P}_n$ encode their sets as encrypted Bloom filters and compute the logical `AND` between all of them.

Kerschbaum [43] shows how to use the Goldwasser-Micali (GM) cryptosystem to aggregate Bloom Filters into a single Bloom Filter representing the intersection. While the homomorphic property of the GM cryptosystem only allows evaluation of the `XOR` operation in the plaintext domain, the protocol uses the Sander Young Yung (SYY) technique [47] to extend it to perform a probabilistic `AND` operation as well. The authors also show how to outsource the protocol so to obliviously evaluate the resulting Bloom Filter. For this they use the Boneh-Gos-Nissim cryptosystem instead, because the SYY technique cannot be used to perform both the Bloom Filter intersection and evaluation due to the required multiplicative depth. Actually, this paper proposes a 2-party protocol[1] but we pose that it can be extended to a multi-party protocol by using a threshold version of the GM scheme [35].

1: The malicious protocol is ambiguously called MPSI, but it is a 2-party protocol.

Counting Bloom filters

Objective: Aggregate the sets into a counting Bloom filter representing the multiset sum and extract a Bloom filter for elements with multiplicity n.

Miyaji and Nishida [6] use the exponential ElGamal cryptosystem to sum up the separate Bloom Filters into one Counting Bloom Filter. For each bin, they then subtract the number n and decrypt-to-zero, which results in an instance of a Bloom filter with a 1 where one would expect it and random values for all the other positions.

In *Practical Multi-party Private Set Intersection Protocols*, Bay et al. propose a similar construction that uses Paillier rather than exponential ElGamal, and which inverts the Bloom filter so not to have to subtract n from each ciphertext. The authors also provide an open source implementation.

Debnath et al. [37] provide an exponential ElGamal-based protocol as well, which functions more like the protocol by Bay et al. Importantly, their protocol saves bandwidth by having the leader aggregate the bins pertaining to its set elements and decrypting those, rather than submitting all those bins for decryption. However, this does reveal the size of the leader's set.

13.4 Other MPSI

We identified two oblivious transfer-based protocols that rely on different set representations. Both protocols use symmetric key techniques rather than expensive homomorphic asymmetric cryptography. However, because of that, both of these schemes require messages to be sent between all parties.

Garbled Bloom filters

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as garbled Bloom Filters and compute the logical XOR between all of them to compute the aggregated garbled Bloom filter of the intersection.

Inbar, Omri, and Pinkas [45] propose a multi-party version of a similar 2-party garbled Bloom filter-based approach [22]. In this protocol, each party first creates a regular Bloom filter as well as a garbled Bloom filter (GBF) and t secret shares for each bin, for which the authors use XOR secret sharing. Then each party performs multiple oblivious transfer interactions with each other party: For each bin the sender requests a random string where their regular Bloom Filter is 0 and the receiver's GBF contents when it is 1. Finally each party computes the logical XOR over these results per bin and sends the resulting shares to the leader to combine the shares. The leader can check which of its elements are contained in aggregated GBF.

Individual element encodings

Kolesnikov et al. [5] proposed another protocol a year before, which is also based on symmetric key techniques. The authors introduce a new oblivious transfer primitive that they call Oblivious Programmable Pseudo-Random Functions (OPPRFs). In short, a programmable PRF is a pseudo-random function that returns a pre-programmed result for certain inputs, and an OPPRF allows to evaluate such a function obliviously.

There are two phases to this protocol: First, each party programs n OPPRFs so that they output a secret share when they inputs an element that is in this party's set. When these secret shares are combined using an XOR operation, the result is 0. The parties then query each others OPPRF with the elements in their own sets. In the second phase, the leader combines the secret shares by having each assistant XOR the resulting shares that they received, and then using an OPPRF again to access those results.

The authors have implemented and open-sourced this protocol. In their work, they show that the protocol takes approximately 70 seconds for 5 parties to find the intersection of sets with one million items each, which makes it faster than all known previous works for large sets. For this reason we consider this protocol to be the state of the art in MPSI protocols. Unfortunately, like the previously discussed protocol, this scheme does suffer from a high degree of interactivity.

13.5 Summary

Looking at Table 13.1, it may seem that the general MPC solution compares well to the other protocols. However, the asymptotic complexities do not capture that this protocol has a large constant factor when it computes to computation, as well as the (poly)logarithmic terms. Moreover, the table shows how over time, protocols have required less stages of interaction.

Currently, the fastest protocol for large sets is that by Kolesnikov et al. [5], but it does suffer from a high communication cost and a high degree of interaction. In general, symmetric key technique-based protocols are experimentally faster for large sets than techniques relying on more expensive asymmetric key techniques. Particularly, those based on homomorphic threshold cryptosystems over integers such as Paillier and exponential ElGamal.

Multi-party private threshold-set operations

14

14.1 Polynomial roots-based T-MPSO 48
Multiply-then-derive 48
14.2 Bloom filter-based T-MPSO 49
Counting Bloom filters . . . 49
14.3 Other T-MPSO 50
Exhaustive search 50
14.4 Summary 50

Threshold operations come in many variants. For example, there are threshold intersections and thresholds unions, where the intersection only reveals the elements that are in at least τ sets as well as the leaders set. Moreover, there are threshold operations and over-threshold operations, where unlike for threshold operations, the over-threshold operations reveal the multiplicity of elements in the resulting set. In other words, over-threshold operations are not count-hiding. Do note that the 'threshold' in threshold operations is not related to the term in threshold cryptosystems.

Apart from these variants, there are recent works [48–50] that propose protocols named threshold intersections, but they achieve a different functionality than the threshold intersections explained in Chapter 2. Specifically, rather than returning the elements that appear at least τ times, they return the intersection, if and only if the cardinality of the intersection exceeds a threshold τ. In that sense, they are *conditional* set intersections. We do not take these protocols into account in our comparison. Table 14.1 gives an overview of the threshold intersection and threshold union protocols discussed in this chapter.

Table 14.1: Overview of prevalent works proposing T-MPSO protocols, including asymptotic complexities and security properties.

Work		Communication				Computation		Security			Privacy	
Ref.	Year	Topology	Leader	Assistant	#	Leader	Assistant	Collusion	Assumption	Mal?	Size	Count
General MPC		Mesh	$\tilde{O}(nk\lambda)$*	$\widetilde{O}(nk\lambda)$*	-	-	-	$n-1$	-	-	-	
[18]	2005	Wheel	$O(n^2k\lambda)$*	$O(n^2k\lambda)$*	4	$O(ntk\kappa + k^2\kappa)$*	$O(ntk\kappa + k^2\kappa)$*	$n-1$	DCR	✓		
[18]	2005	Wheel	$O(n^2k\lambda)$*	$O(n^2k\lambda)$*	4	$O(ntk\kappa + k^2\kappa)$*	$O(ntk\kappa + k^2\kappa)$*	$n-1$	DCR			✓
[51]	2007	Mesh	$O(nk\lambda)$*	$O(nk\lambda)$*	8	$O(nk^2\kappa)$*	$O(nk^2\kappa)$*	$n-1$	DCR			
[6]	2015	Star	$O(n^2k\lambda)$*	$O(nk\lambda)$*	2	$O(nk\kappa)$*	$O(nk\kappa)$*	$n-1$	DDH		✓	
Exact (Ours)		Wheel	none	$O(\lvert\mathcal{U}\rvert n^2\lambda)$	2	$O(\lvert\mathcal{U}\rvert n^2\kappa)$	$O(\lvert\mathcal{U}\rvert n^2\kappa)$	$n-2$	PRF		✓	✓
Approx. (Ours)		Wheel	none	$O(n^2k\lambda)$	2	$O(n^2k\kappa)$	$O(n^2k\kappa)$	$n-2$	PRF		✓	✓

14.1 Polynomial roots-based T-MPSO

Polynomial roots are suitable for performing threshold operations, because we can reduce the multiplicity of the roots by one for every time we compute its derivative.

Multiply-then-derive

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomials $S_1(x), \ldots, S_n(x)$ to compute their product $S_1(x) \ldots S_n(x)$, then compute its $\tau - 1$th derivative.

To the best of our knowledge, Kissner and Song [18] proposed the first multi-party private threshold protocols. Similar to their other protocols,

the authors propose to use the threshold Paillier cryptosystem to manipulate encrypted polynomials by their coefficients. The protocol works by passing around an encrypted polynomial and having each party multiply their polynomial set encoding with it. Then, the parties compute the derivative of this polynomial, randomize it, multiply it with a fixed polynomial containing no roots from the universe $\mathcal{U}$ and add it to their original but randomized encoding. This results in a polynomial encoding the over-threshold set union, without leaking information about the elements in the input sets.

While the polynomial can be factored in polynomial time, the authors suggest that the parties evaluate the polynomial for each element in their set, multiply it with a random number and add that element to it. When such a ciphertext is decrypted, it is a random number when the element does not belong in the threshold union, and otherwise it is the decryption of the corresponding element. The authors also propose a threshold set union protocol that does not reveal the final counts.

Sang and Shen [46] propose a protocol that fulfills the same objective. Instead of randomizing using random numbers, they randomize using a hash of the encrypted values, and instead of factoring the final result, they let a party evaluate the polynomial for the elements in their set and collaboratively decrypt the ciphertext to show whether or not that element was indeed in the resulting set. For this reason, this protocol implements a threshold set intersection rather than a threshold set union. We exclude this protocol from the table, since it does not provide concrete choices for its cryptographic primitives.

Frikken [51] propose an over-threshold union protocol that is an extension from their set union protocol that we discuss in the next chapter. Like the other protocols, each party submits an encrypted polynomial and the parties multiply these together, after which they take the $\tau - 1$th derivative. This time, instead of hiding the elements' multiplicities the parties simply evaluate the resulting polynomial for each element in their sets and randomize the results, after which they proceed with the MPSU protocol. For this reason the protocol requires many stages in which each party must be online. The protocol uses a bulletin board to post messages to but we replace this by broadcasts in our comparison.

14.2 Bloom filter-based T-MPSO

Counting Bloom filters

Objective: Aggregate the sets into a counting Bloom filter representing the multiset sum and extract a Bloom filter for elements with multiplicity $\geq \tau$.

Following their set intersection protocol, Miyaji and Nishida [6] show how to extend it to a threshold set intersection. As before, they compute a counting Bloom filter that encodes the multiset sum of the input sets, by performing addition on encrypted Bloom filters that encode the input sets using exponential ElGamal. To extract a Bloom filter only encoding the threshold set intersection, they create $n - \tau$ ciphertexts of each bin,

by subtracting a number x from each, where $x = \tau, \ldots, n$. As a result, when the ciphertexts are decrypted-to-zero, if one ciphertext relating to a bin is zero, that bin is included in the resulting counting Bloom filter. Which ciphertext is zero, reveals the count. The resulting counting Bloom filter can be evaluated by the leader. So, this protocol implements an over-threshold set intersection.

14.3 Other T-MPSO

Exhaustive search

Mahdavi et al. [52] propose a protocol that is based on a new method that the authors call oblivious pseudo-random secret sharing (OPR-SS). From a high level, the protocol works by having each party engage with the key holder in an OPR-SS protocol to generate some secret shares that depend on the elements in their sets. The parties then send these shares to a reconstructor as part of a hash table, after which the reconstructor exhaustively tries all combinations of parties' sets to check if the shares combine to form a pre fixed secret, which is 0 in their example. Differently from other threshold protocols, this protocol not only leaks the count but also which parties' had this element in their input sets. For this reason we do not include this protocol in our comparison table.

14.4 Summary

Generally, threshold operations require a high degree of interaction. Only the protocol by Miyaji and Nishida [6] required two stages of communication, but the protocol is not count-hiding. So far, there were no works that are count-hiding and only require two stages of interaction.

15 Multi-party private set unions

Multi-party private set unions are in some ways harder to realize than multi-party private set intersections because they require it to be feasible to extract all the elements contained in a set representation. We pose that for this reason, fewer works propose MPSU protocols. On the other hand, note that a threshold set union protocol with threshold $\tau = 1$ can also take the place of an MPSU protocol, such as those discussed in the previous chapter. Table 15.1 compares the works discussed in this chapter.

15.1 Polynomial roots-based MPSU 51
Oblivious evaluation 51
Randomized polynomial sum 52
15.2 Summary 52

Table 15.1: Overview of prevalent works proposing MPSU protocols, including asymptotic complexities and security properties.

Work		Communication				Computation		Security			Privacy
Ref.	Year	Topology	Leader	Assistant	#	Leader	Assistant	Collusion	Assumption	Mal?	Size
General MPC		Mesh	$\tilde{O}(nk\lambda)$*	$\widetilde{O}(nk\lambda)$*	-	-	-	$n-1$	-	-	-
[51]	2007	Mesh	$O(nk\lambda)$*	$O(nk\lambda)$*	6	$O(nk^2\kappa)$*	$O(nk^2\kappa)$*	$n-1$	DCR	✓	
[19]	2012	Mesh	$O(n^3k^2\lambda)$*	$O(n^3k^2\lambda)$*	2	$O(n^4k^2\lambda)$*	$O(n^4k^2\lambda)$*	$\lfloor \frac{n}{2} \rfloor$	none	✓	✓
Exact (Ours)		Star	none	$O(\lvert\mathcal{U}\rvert\lambda)$	1	$O(\lvert\mathcal{U}\rvert n\kappa)$	$O(\lvert\mathcal{U}\rvert n\kappa)$	$n-2$	PRF		✓
Approx. (Ours)		Star	none	$O(k\lambda)$	1	$O(n^2k\kappa + \lvert\mathcal{U}\rvert)$	$O(n^2k\kappa)$	$n-2$	PRF		✓

15.1 Polynomial roots-based MPSU

Oblivious evaluation

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomial $S_1(x), \ldots, S_n(x)$ and let each set of parties $\{\mathcal{P}_1\}, \{\mathcal{P}_1, \mathcal{P}_2\}, \ldots, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ in turn obliviously evaluate their multiset sum, privately reporting which elements were not contained in the previous set.

The idea of the protocol by Frikken [51] is to use the multiset sum operation for polynomials as described in Section 7.2, and to evaluate it obliviously not to get a sense of an element's multiplicity. Specifically, each party creates an encrypted polynomial set encoding of their input and broadcast it. Then, in turn, each party receives an encrypted polynomial set encoding, multiplies it with their polynomial, and evaluate it for their elements. This returns ciphertexts that decrypt to 0 when the corresponding element was indeed contained in the encoding. The party does not yet decrypt, and broadcasts a tuple containing this ciphertext multiplied with the corresponding element, as well as an encrypted copy of the element. After all parties are done, they randomize and securely shuffle all ciphertext tuples and decrypt them. They discard any zeroes, and otherwise recover an encrypted element that is part of the set union.

Randomized polynomial sum

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomial $P_1(x), \ldots, P_n(x)$ and pick uniformly random polynomials $r_1(x), \ldots, r_n(x)$ to find some $\frac{u'(x)}{L(x)} = \sum_i \frac{r_i(x)}{P_i(x)}$ so to extract $L(x) = \text{lcm}(P_1(x), \ldots, P_n(x))$.

Seo, Cheon, and Katz [19] propose an MPSU protocol that works by computing the least common multiple of sets encoded as polynomials, which they factor to complete the protocol. While polynomial factoring over a finite field can be done in polynomial time, it is an expensive operation. Fortunately, the authors encode the polynomials in such a way that they are more easily factored using Reversed Laurent Series. The protocol revolves around arithmetic on the rational randomized polynomials $\frac{r_i(x)}{P_i(x)}$, which are shared using Shamir's secret sharing. However, this requires the use of a collaborative multiplication protocol, which results in interaction and communication overhead. The protocol is information-theoretically secure.

15.2 Summary

The work by Frikken [51] has small asymptotic complexities, but it requires six stages of communication. On the other hand, the work by Seo, Cheon, and Katz [19] requires just two stages of communication, but its complexities scale poorly with the number of parties involved. While the general MPC solution seems to perform just as good as Frikken [51], this protocol incurs a large constant factor in terms of communication and run time.

Multi-party private set operation-cardinality protocols

16

In this chapter we discuss two kinds of multi-party private set operation-cardinality protocol: set intersection-cardinality and set union-cardinality protocols. There are many different ways to realize such an operation, such as estimating it using sketches, or shuffling bitsets or Bloom filters to allow a party to simply count the number of filled bins. Some of the protocols we discuss here are simple extensions of MPSI and MPSU operations, but do note that there is not always a trivial way to extend a pre-existing protocol in such a way.

16.1 Polynomial roots-based MPSO-CA 53
Oblivious evaluation 53
16.2 Bitset-based MPSO-CA . . . 54
Logical operations on bitsets 54
16.3 Bloom filter-based MPSO-CA 54
Shuffling Bloom filters . . . 54
16.4 Other MPSO-CA 54
Garbled Bloom filters 54
Min-max sketches 55
Individual element encodings 55
16.5 Summary 56

Work			Communication				Computation		Security			Privacy
Ref.	Year	Operation	Topology	Leader	Assistant	#	Leader	Assistant	Collusion	Assumption	Mal?	Size
General MPC		`MPSI-CA`	Mesh	$\widetilde{O}(nk\lambda)$*	$\widetilde{O}(nk\lambda)$*	-	-	-	$n-1$	-	-	-
[21]	2005	`MPSI-CA`	Mesh	$O(nk\lambda)$*	$O(nk\lambda)$*	3	$O(nk\kappa)$*	$O(nk\kappa)$*	1	CKHF	✓	
[18]	2005	`MPSI-CA`	Wheel	$O(n^2k\lambda)$*	$O(n^2k\lambda)$*	4	$O(ntk\kappa + k^2\kappa)$*	$O(ntk\kappa + k^2\kappa)$*	$n-1$	DCR	✓	
[51]	2007	`MPSU-CA`	Mesh	$O(nk\lambda)$*	$O(nk\lambda)$*	6	$O(nk^2\kappa)$*	$O(nk^2\kappa)$*	$n-1$	DCR		
[53]	2010	`MPSU-CA`	Mesh	$O(ntk^2\lambda)$*	$O(ntk^2\lambda)$*	-	$O(ntk^2\lambda)$*	$O(ntk^2\lambda)$*	$\lfloor \frac{n}{2} \rfloor$	none		✓
[37]	2021	`MPSI-CA`	Wheel	$O(nk\lambda)$*	$O(k\lambda)$*	3	$O(nkh\kappa)$*	$O(k\kappa)$*	$n-1$	DDH		
Exact (Ours)		`MPSO-CA`	Wheel	none	$O(\lvert\mathcal{U}\rvert n\lambda)$	2	$O(\lvert\mathcal{U}\rvert n\kappa)$	$O(\lvert\mathcal{U}\rvert n\kappa)$	$n-1$	DDH		✓
Approx. (Ours)		`MPSO-CA`	Wheel	none	$O(nk\lambda)$	2	$O(nk\kappa)$	$O(nk\kappa)$	$n-1$	DDH		✓

16.1 Polynomial roots-based MPSO-CA

Oblivious evaluation

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomial $S_1(x), \ldots, S_n(x)$ and let each party in turn obliviously evaluate their multiset sum, privately reporting how many elements are in the final set.

Kissner and Song [18] propose a protocol that is similar to their MPSI protocol that we discuss in Chapter 13. Instead of revealing the elements after obliviously evaluating the aggregated polynomial, the parties randomize the ciphertexts and shuffle them, after which they count the number of 0s.

The protocol by Frikken [51] that we discussed in Chapter 15 can be extended to perform set union-cardinalities, simply by changing the way a party creates a tuple. Instead of decrypting to the result set item, the tuples' ciphertexts will then decrypt to 0 when it is a duplicate, and a random number otherwise, so it should be counted.

16.2 Bitset-based MPSO-CA

Logical operations on bitsets

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as encrypted bitsets and compute the logical `AND` between all of them, then compute their sum c and return $|\mathcal{U}| - c$.

Burkhart et al. [53] propose SEPIA, which is a library for securely aggregating multi-domain network data using Shamir's secret sharing. In this work they provide a protocol for the distinct-count problem, which is identical to a set union-cardinality. The operation is simple: Encode each party's set as a bitset, invert it, compute the `AND` operation between them, and return the size of the universe minus the total sum of the aggregated inverted bitset. To do so securely, the `AND` and sum operations are executed using secret sharing. Due to the `AND` operation, the number of rounds of interaction in the protocol scales logarithmically with the number of parties n.

16.3 Bloom filter-based MPSO-CA

Shuffling Bloom filters

Objective: Aggregate the sets into a counting Bloom filter representing the multiset sum and extract a Bloom filter for elements with multiplicity n, then shuffle its bins.

Debnath et al. [37] propose a protocol using (n, n)-exponential ElGamal, so the parties can generate a public key without a trusted dealer. From a high level, the protocol has every party encode their sets as a Bloom filter, then aggregate these to get the Bloom filter corresponding to the intersection. At this point, the leader selects those encrypted bins corresponding to the elements in its set and aggregates them. The parties then take turns shuffling the ciphertexts, after which they collaboratively decrypt. The leader can then simply count the number of bins that were 0. This protocol leaks the size of the leader's set because it only submits those bins corresponding to the leader's elements to shuffle and decrypt. Also, while the authors give an argument for how to make the protocol secure in the malicious model, they do not prove that it is indeed.

16.4 Other MPSO-CA

Garbled Bloom filters

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as garbled Bloom filters, then securely aggregate and shuffle the bins, to count the number of filled bins without revealing the resulting set.

Egert et al. [16] propose a protocol based on garbled Bloom filters to outsource the aggregation and evaluation of set intersection and union-cardinalities. The oblivious aggregation takes the shape of t layers of accumulating parties that receive additive secret shares from each party for every bin in the Bloom Filter: All bins that contained a 1 are turned into a random number, while 0s remain 0, and these are shared. For an intersection the Bloom Filters are inverted. The accumulators sum the shares and forward the aggregated shares to the final evaluator in a permutation that they decided collectively. The evaluator counts the number of zeroes to determine an estimate of the resulting set cardinality. The collusion resistance in this protocol comes from external parties: In a non-outsourced setting the protocol would not be collusion resistant, since any party would know the permutation used, which reveals the Bloom filter representing the resulting set. Due to the requirement of having multiple layers of accumulating parties, the protocol requires a high degree of interaction. We exclude this protocol from the table for its dependence on outsourcing, since we cannot define complexities for each party or express its interactivity.

Min-max sketches

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as min-max sketches, then securely aggregate and count how many bins are identical, to estimate the Jaccard index.

Dong and Loukides [23] propose an approximate protocol that uses min-max sketches to estimate the Jaccard index of a set, which in turn allows to compute the set intersection-cardinality. To do so, however, the parties must already have an estimate of their sets' union-cardinality. The authors do give a description about how to construct an MPSU-CA protocol, but they do not describe it in detail. Their MPSI-CA protocol is as follows. Each party encodes their set as a min-max sketch. Then the parties evaluate each bin of the sketches using a secure equality protocol, and count in how many cases the bins were indeed equal. Finally, they compute the estimate. We exclude this work from the table as the authors do not propose concrete choices for the secure equality test, multiplication and floating point division protocols. In any case, all these protocols require some form of interaction when done over additive secret shares.

Individual element encodings

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ compute a keyed commutative one-way hash function on the elements of every party's set, and then count how many hashes are identical.

The protocol by Vaidya and Clifton [21] relies on a commutative one-way hash function to compute the set intersection-cardinality. In short, such a hash satisfies all the requirements of a cryptographic keyed hash function, but in addition it holds for hash function $\mathcal{H}$ that $\mathcal{H}_{k_1}(\mathcal{H}_{k_2}(x)) = \mathcal{H}_{k_2}(\mathcal{H}_{k_1}(x))$ for two keys k_1 and k_2 and an element x. The idea behind

this technique is that when two elements are hashed by each party with their own key, no matter the order in which this happened, their hash will be identical. Unfortunately, together, two parties can launch a probing attack, meaning that the protocol is not collusion resistant.

16.5 Summary

No previous work so far manages to take less than three stages of communication. This seems to be because these protocols are split into three stages: aggregation, shuffling and decryption. The smallest asymptotic complexities are achieved by Debnath et al. [37], but this work does not hide the size of the leader's set.

Multi-party private multiset operations

17

We have already discussed several protocols that support multiset operations. In fact, we can realize multiset sums with an over-threshold threshold union with $\tau = 1$. And next to that, many polynomial-based protocols extend naturally to multisets as well. In this chapter we shortly address those multi-party private multiset operation protocols that we have not examined before.

17.1 Polynomial roots-based MPMO 57
Multiply polynomials 57
17.2 Bloom filter-based MPMO . 57
Counting Bloom filters . . . 57
17.3 Other MPMO 58
17.4 Summary 58

17.1 Polynomial roots-based MPMO

Multiply polynomials

Objective: Let parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ encode their sets as polynomial $S_1(x), \ldots, S_n(x)$ and compute the multiset sum $\prod_{i=1}^{n} S_i(x)$.

The work by Hong et al. [7] studies the problem of multiset sums under the misnomer of 'multiset unions'. In [54] they provide experimental results. In their protocol, the authors adapt the ElGamal cryptosystem to encrypt whole polynomials rather than integers, of which security relies on the Short-Interval DDH assumption (SI-DDH). Like Kissner and Song [18], they encode multisets as polynomials and multiply them homomorphically, this time within a single ciphertext, to obtain the polynomial encoding the multiset sum. After decryption, the polynomial must still be factored, which is possible in polynomial time [20]. The protocol's communication is a constant factor lower than the protocol by Kissner and Song [18] in the size of the universe $|\mathcal{U}|$. Computation for the leader takes $O(n^{2.6}k^{1.6})$* operations, while assistants only perform $O(n^{1.6}k^{1.6})$* operations.

17.2 Bloom filter-based MPMO

Counting Bloom filters

Objective: Aggregate the sets into a Counting Bloom Filter representing the multiset and extract all possible elements in the universe $\mathcal{U}$.

The multiset union protocol by Shishido and Miyaji [39] uses a definition of Counting Bloom Filters that is different from the definition we give. The goal for this protocol is to return as accurately as possible the multiplicities of the elements in the resulting union. Every assistant in the protocol actually uses two Bloom Filters alongside each other that are constructed differently so that one represents an upper bound for the multiplicity and the other a lower bound, and they send it to the leader,

who simply aggregates both Bloom Filters separately by summing them. Finally, the leader goes through all possible elements in the universe $\mathcal{U}$ and checks if they are contained in the Bloom Filters. By using the upper and lower bound they can make a best-effort multiset union. It seems that the order of traversal changes the outcome of this multiset union and because we are exhaustively checking each element in the universe, the protocol is either inefficient or inaccurate for large universes.

17.3 Other MPMO

The work by Blanton & Aguiar [9] proposes interactive protocols for several types of set and multiset operations based on secret sharing and some sub-protocols, including oblivious sorting and secure comparisons. They do so using the multiset-to-set transformation that we discussed in Chapter 3, but they also provide specialized protocols for multiset operations. The multiset sum operation is realized by their 'multiset union' protocol. While the authors leave the round complexity as a variable depending on the choice of sub-protocols, it must scale at least linearly with the number of input elements k.

17.4 Summary

As for the other works, no previous works seem to offer a non-interactive solution, but this not necessarily mean that these protocols require large bandwidth. For example, the protocol by Hong et al. [7] fit the whole multiset encoding within one ciphertext. Apart from these work, there are recent works [55, 56] that propose specialized multiset intersection-cardinality protocols in the two-party case.

Non-interactive protocols

So far there were no known protocols for private set operations between multiple parties that required no interactions. In this part we propose the first non-interactive protocols for this task, which is our own contribution. We discuss two constructions: In the first construction, every party must have some shared secret with other parties beforehand, after which they can execute our protocols non-interactively. In the second construction, parties only have to share a public key beforehand, after which the protocols are non-interactive.

Non-interactive secret sharing 18

18.1 Outline 60
18.2 Symmetric construction 61
Mask generation 61
Implementation choices . . . 62
Security 62
18.3 Asymmetric construction . . 62
Mask generation 63
Implementation choices . . . 63
Security 64

Several of the works in multi-party private set operations use threshold cryptosystems to securely transmit elements of a set encoding to another party [6, 18], including our earlier bitset-based and Bloom filter-based works. However, as discussed in Chapter 11, such a threshold cryptosystem necessarily introduces interactivity when decrypting a ciphertext: Several parties must receive the ciphertext, partially decrypt it and transmit the resulting decryption shares so they can be combined. For this reason, all our non-interactive protocols rely on secret sharing methods instead, which is discussed in Chapter 18.

In this chapter we introduce two constructions for multiplicative secret sharing. In the first construction, a party must have some pre-shared secret key with several other parties. In the second construction, it is only necessary for a party to share their public key. These constructions are analogous to symmetric (private-key) and asymmetric (public-key) cryptosystems, respectively, and they enjoy similar benefits: The symmetric construction is significantly faster than the asymmetric construction, but it is harder to deploy in practice. That is, it does not require expensive pairing operations, but it requires a setup phase between all pairs of parties.

18.1 Outline

Recall from Chapter 18 that for a multiplicative secret sharing scheme, the secret is revealed after multiplying all shares together. Consider the case where we have shares w_i for $i = 1, \ldots, n$ that we call masks, since they multiply to the identity:

$$\prod_{i=1}^{n} w_i = 1$$

We describe constructions for generating shares w_i that abide to this equation. The intuition behind both non-interactive secret sharing setups is the following: Let each party $\mathcal{P}_i$ generate $i - 1$ unknown numbers $u_{i,j}$ for $j = 1, \ldots, i - 1$. Now, let each party $\mathcal{P}_i$ compute:

$$w_i = \prod_{j=1}^{i-1} u_{i,j} \prod_{j=i+1}^{n} u_{j,i}^{-1} \tag{18.1}$$

In other words, let party $\mathcal{P}_i$ multiply together all its unknown numbers, and the inverse of the ith random numbers of parties with a higher index

i. Then, the product of w_i for $i = 1, \ldots, n$ is indeed the identity:

$$\prod_{i=1}^{n} w_i = \prod_{i=1}^{n} \left(\prod_{j=1}^{i-1} u_{i,j} \prod_{j=i+1}^{n} u_{j,i}^{-1} \right) \tag{18.2}$$

$$= \left(\prod_{i=1}^{n} \prod_{j=1}^{i-1} u_{i,j} \right) \left(\prod_{i=1}^{n} \prod_{j=i+1}^{n} u_{j,i}^{-1} \right) \tag{18.3}$$

$$= \left(\prod_{i=1}^{n} \prod_{j=1}^{i-1} u_{i,j} \right) \left(\prod_{j=1}^{n} \prod_{i=1}^{j-1} u_{j,i}^{-1} \right) \tag{18.4}$$

$$= \prod_{i=1}^{n} \prod_{j=1}^{i-1} u_{i,j} u_{i,j}^{-1} \tag{18.5}$$

$$= 1 \tag{18.6}$$

The ith unknown numbers of parties $\mathcal{P}_j$ with $j = i + 1, \ldots, n$ are the same as the ith unknown numbers of all parties $\mathcal{P}_j$ for $i = 1, \ldots, j - 1$.

In the coming sections we discuss how to construct the unknown numbers so that w_i can be used as a secure mask for multiplicative secret sharing. Note that when we say unknown, we mean that this number is only known to those parties who need it to compute their mask. In the coming sections we provide a formal security proof for these constructions, to prove that the constructions are secure against up to $n - 2$ colluding parties and to clarify what it means for these numbers to be unknown. To ease notation we assume $u_{i,j} = u_{j,i}$ in the remainder of this work.

18.2 Symmetric construction

The outline of our scheme relied on some 'unknown' numbers $u_{i,j}$ but we did not specify in what way they are unknown. In this section we discuss a symmetric construction where these unknown numbers are chosen by a pseudo-random function, and number $u_{i,j}$ is only known to parties $\mathcal{P}_i$ and $\mathcal{P}_j$. This construction was discussed by Erkin and Tsudik [57]. The reason that we call this construction symmetric is that each pair of parties share a secret key $sk_{i,j}$. This is analogous to symmetric-key cryptography, where parties share the same secret key, rather than a practically irreversible public key.

In the remainder of our work we assume that the secret keys are already in place. So for example, each pair of parties has previously communicated a truly random secret key over a secure channel, or each pair of parties has executed a secure key exchange. The security of this symmetric setup relies on a cryptographically-secure Pseudo-Random Function (PRF). In addition, we prove information-theoretic security under the random oracle model.

Mask generation

We denote a PRF by the function PRF $\left(sk_{i,j}, t\right)$, which represents the tth output of the PRF that is seeded with $sk_{i,j}$. We shall denote the group of the mask by $\mathbb{G}_T$, to be consistent with the asymmetric construction. The

mask w_i of party $\mathcal{P}_i$ for run t of a protocol is then given by:

$$w_i = \prod_{j=1}^{i-1} \text{PRF}\left(sk_{i,j}, t\right) \prod_{j=i+1}^{n} \text{PRF}\left(sk_{i,j}, t\right)^{-1} \in \mathbb{G}_T \tag{18.7}$$

Note that every t must be used only once.

Implementation choices

The description of this construction does not consider any particular group. For our purposes we are not restricted to a group over the integers, except for our specialized multiset sum protocol described in Chapter 21. So, for the other parts of this thesis we use more efficient groups over elliptic curves. The reason such a group is more efficient is that there exist specialized attacks to find the discrete logarithm in integer-based groups. As such, an elliptic curve group can be significantly smaller for the same level of security. Note that this construction does not require the use of pairing-friendly curves. Hence, we use the Ristretto version of Curve25519.

To realize the pseudo-random function, we choose to implement this using a cryptographically-secure hash function. This allows us to use the hash-to-curve functionality for Ristretto curves, to efficiently generate a corresponding point on the curve without performing point multiplication. We call the hash function with the seed $sk_{i,j}$, appended by the nonce t. Importantly, the hash function should be secure against length-extension attacks, to prevent an adversary from reducing the search space when attempting to recover the seed. One such hash function is SHA3-512 [58] .

[58]: Dworkin (2015), *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*

Security

Let us replace the cryptographically-secure PRF with a random oracle. Instead of generating the pseudo-random element $\text{PRF}\left(sk_{i,j}, t\right)$, party i supplies the random oracle with the seed $sk_{i,j}$, appended by the index of this run of the protocol t. Then, clearly mask w_i is indistinguishable from randomness for any party that does not know $sk_{i,j}$. By extension, since our group is cyclic, the entire secret share is indistinguishable from randomness. In other words, as long as there is a seed $sk_{i,j}$ that remains unknown to a group of colluding parties, the secret shares are indistinguishable from randomness. This is always the case when at least two honest parties remain, so the secret sharing scheme is collusion resistant for up to $n - 2$ colluding parties under the random oracle model.

18.3 Asymmetric construction

Our asymmetric construction is entirely based on the secret shares in the secure aggregation protocol by Kursawe, Danezis, and Kohlweiss [59], and as further studied by Patsakis, Clear, and Laird [60]. In fact, for multiset sums between bitsets we use their protocol without adaptations.

The key takeaway of this protocol is the asymmetric construction that enables parties to non-interactively generate masks for a group of parties only by knowing their public keys. As a consequence, parties only have to secure one secret key and the total number of keys in the setup grows linearly rather than quadratically when compared to the symmetric setup.

Mask generation

Recall from Section 9.3 that using bilinear pairings we can compute multiplications in the exponent of multiplicative elements in a group $\mathbb{G}_T$. Specifically, this is achieved by the pairing function $\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. In addition, we can hash objects to curve points in the elliptic curve group $\mathbb{G}_2$ using a hash function that we denote by $\mathscr{H}_2 : \mathbb{Z} \rightarrow \mathbb{G}_2$. Importantly, this hash function has the property that it is infeasible to know the discrete logarithm of the curve point in terms of the generator base. It is this property that allows us to replace the PRF from the symmetric setup. Note that t must be a nonce.

First, all parties once execute the one-time setup:

One-time setup

1. The leader $\mathscr{P}_1$ broadcasts the public parameters: $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and generator $g = e(P, Q) \in \mathbb{G}_T$. We denote the order of these groups as q.
2. Every party $\mathscr{P}_i$ for $i = 1, \ldots, n$ generates their public key $pk_i \leftarrow sk_i P \in \mathbb{G}_1$ from a secret key $sk_i \in_R \mathbb{Z}_q$ and broadcasts this public key.

The one-time setup requires a party to perform a single pairing and an elliptic point multiplication in $\mathbb{G}_1$. This should take at most $O(\kappa)$ time. It is possible to extend the setup with a step to pre-compute the inverse of all $i' > i$ public keys $pk_{i'}$ for a constant time speedup of the remainder of the protocol.

The mask w_i of party $\mathscr{P}_i$ for run t of a protocol is then given by:

$$w_i = \prod_{j=1}^{i-1} \mathrm{e}\left(pk_j, \mathscr{H}_2(t)\right)^{sk_i} \prod_{j=i+1}^{n} \left(\mathrm{e}\left(pk_j, \mathscr{H}_2(t)\right)^{sk_i}\right)^{-1} \in \mathbb{G}_T \tag{18.8}$$

It is clear that in this setup, $u_{i,j} = \mathrm{e}\left(pk_j, \mathscr{H}_2(t)\right)^{sk_i}$. We provide the equation in this form to adhere to the format of Equation 18.1, but in practice, it is more efficient for each party $\mathscr{P}_i$ to compute the inverse of pk_j for $j = i + 1, \ldots, n$ in advance and perform the exponentiation with sk_i once after completing the products.

Implementation choices

In this thesis we use a Barreto-Naehrig curve that satisfies 128-bit security in terms of AES security, as explained in Section 9.3. The reason for this

choice is that it is implemented in the MIRACL C++ library, and that it is used in other proof of concept implementations of research work [59].

Security

We give a proof that to show that the secret shares in the asymmetric construction are computationally indistinguishable from random. Note that our proof relies on the BDDH assumption as discussed in Section 9.2. For this scheme to be secure it should be infeasible for an adversary to find the discrete log for an output of the hash function $\mathcal{H}_2$ to group $\mathbb{G}_2$.

Corollary 18.1 *Following the Bilinear Decisional Diffie-Hellman (BDDH) assumption it holds that:*

$$(P, Q, aP, bP, cQ, e(P, Q)^{abc}) \stackrel{c}{\equiv} (P, Q, aP, bP, cQ, r)$$

for generators $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, numbers $a, b, c \in \mathbb{Z}_q$ and random point $r \in_R \mathbb{G}_T$.

The BDDH assumption in turn implies the infeasibility of the Elliptic Curve Discrete Log Problem (ECDLP), which is the task of retrieving a given only tuple (P, aP) with generator P and a number a. The adversary is constrained as follows:

- The adversary will only form collusions of $n - 2$ parties.
- The adversary cannot compute the discrete logarithm for an output of the hash function to group $\mathbb{G}_2$.
- The adversary does not know the secret key sk_i of a non-colluding party $\mathcal{P}_i$.

Theorem 18.2 *For all $i \in \{1, \ldots, n\}$ share w_i is computationally indistinguishable from randomness under the BDDH assumption: $w_i \stackrel{c}{\equiv} r'_i$ for some random point $r'_i \in_R \mathbb{G}_T$, given the aforementioned constraints.*

Proof. We give a proof by contradiction: Let us assume that there exists a distinguisher $\mathcal{D}$ that can tell apart w_i from r'_i. We show that $\mathcal{D}$ solves the Decisional Bilinear Diffie-Hellman problem. Consider the tuple of public information pertaining to party $\mathcal{P}_i$: $(P, Q, sk_1P, \ldots, sk_nP, h_lQ, w_i)$, where $\mathcal{H}_2(l) = h_lQ$ for some known l.

1. $\mathcal{D}$ can write the sum of all public keys except for pk_i as $s_iP = \sum_{i'=1}^{i-1} sk_{i'}P + \sum_{i'=i+1}^{n} sk_{i'}P$.
2. $\mathcal{D}$ can expand $w_i = e(P, Q)^{s_i h_l sk_i}$.
3. Note that after substituting $pk_i = sk_iP$, distinguisher $\mathcal{D}$ can create tuple
$$T = (P, Q, s_iP, sk_iP, h_lQ, e(P, Q)^{s_i h_l sk_i})$$
4. $\mathcal{D}$ can create another tuple
$$R = (P, Q, s_iP, sk_iP, h_lQ, r'_i)$$

sk_i is chosen uniformly at random and h_l is uniformly chosen under the statistical properties of a strong hash function. Furthermore, since at least one term in s_i is chosen uniformly at random and unknown to $\mathcal{D}$, s_i is also a uniformly random element in $\mathbb{G}_1$. As a consequence, tuple T should be computationally indistinguishable from R under Corollary 18.1. However, since $\mathcal{D}$ can distinguish between $w_i = e(P, Q)^{s_i h_l sk_i}$ and r'_i, the distinguisher breaks the Decisional Bilinear Diffie-Hellman assumption. □

Multi-party private OR & AND operations

19

At the core of our non-interactive multi-party private set and multi-set protocols are secure, non-interactive OR and AND protocols. These protocols can also be of independent interest. In this chapter we give a description of these protocols, we prove their correctness, provide a formal simulation-based security proof and state their efficiency. Actually, we demonstrate correctness and security for the OR protocol, and we pose that by DeMorgan's law, the AND protocol should inherit these properties as we can express it in terms of the OR protocol.

19.1 Protocol description 66
19.2 Correctness 67
19.3 Security 67
Collusion of assistants 68
Collusion with the leader . . 69
Efficiency 69

19.1 Protocol description

The intuition behind these protocols is that we can implement the OR operation using any group, which we show in Tables 19.1 & 19.2 for a multiplicative group. Let us map an input of 0 to the identity element, and an input of 1 to a random group element. Then, we can perform an OR operation by applying the group operation between all inputs. If the result is the identity element, depending on the size of the group, there is an overwhelming probability that all the inputs were the identity element, so the result should be 0. Otherwise, the result should be 1, because at least one input must not have been 0. Formally, the protocol is as follows:

Table 19.1: Truth table of an OR operation.

a	b	$a \vee b$
0	0	0
1	0	1
0	1	1
1	1	1

Table 19.2: Truth table of a multiplicative secret sharing-based OR operation.

a	b	ab
w_a	w_b	1
r_a	w_b	$r_a w_b$
w_a	r_b	$w_a r_b$
r_a	r_b	$r_a r_b$

Secure OR protocol

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ set
   $v_i \leftarrow \begin{cases} w_i \in \mathbb{G}_T & \text{if } x_i = 0 \\ r_i \in_R \mathbb{G}_T & \text{if } x_i = 1 \end{cases}$, where w_i is the generated mask
2. All assistants $\mathcal{P}_i$ for $i = 2, \ldots, n$ send their supposed share v_i to $\mathcal{P}_1$.
3. The leader $\mathcal{P}_1$ determines the operation's result:
   $z \leftarrow \begin{cases} 0 & \text{if } \prod_{i=1}^{n} v_i \equiv 1 \\ 1 & \text{otherwise} \end{cases}$

Protocol 19.1: Protocol for securely computing an OR between several parties' input bits.

Note that due to the randomness, it is not possible to know how many parties submitted 0 or 1. Also note that this protocol can be computed many times in parallel, meaning that parties can non-interactively perform multiple OR operations without requiring additional interactions.

Now we can describe the AND protocol in terms of the OR protocol, which allows the AND protocol to inherit its properties. Of course, it is also possible to describe a stand-alone version of the protocol. The protocol is as follows:

Secure AND protocol

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ invert their input $x'_i = \neg x_i$.
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ take part in the **secure OR protocol** on x'_i, and the leader $\mathcal{P}_1$ outputs z'.
3. The leader $\mathcal{P}_1$ determines the operation's result through inversion: $z = \neg z'$.

Protocol 19.2: Protocol for securely computing an AND between several parties' input bits.

19.2 Correctness

The secure OR operation can output 0 or 1. We show that the protocol outputs the correct result with overwhelming probability. In the previous chapter we demonstrated that:

$$\prod_{i=1}^{n} w_i \equiv 1 \tag{19.1}$$

Now, let us prove that the protocol performs the functionality expected of an OR operation. First, the result should be 0 when all parties' input bit is 0:

Theorem 19.1 *When* $\forall i \in \{1, \ldots, n\}$ *it holds that* $x_i = 0$*, then* $z = 0$*.*

Proof. Note that it suffices to prove that $\prod_{i=1}^{n} v_i \equiv 1$ following the last step of the protocol. At the start of our protocol, if $x_i = 0$ then $v_i = w_i$. This is the case for all $i = 1, \ldots, n$. By Equation 19.1 we have $\prod_{i=1}^{n} v_i \equiv 1$. So $z = 0$. □

Secondly, the result should be 1 when at least one party's input bit is 1:

Theorem 19.2 *When* $\exists i \in \{1, \ldots, n\}$ *so that party* $\mathcal{P}_i$ *has* $x_i = 1$*, then* $z = 0$ *with negligible probability.*

Proof. It suffices to prove that $\prod_{i=1}^{n} v_i \equiv 1$ with a negligible probability. There are some i for which $x_i = 1$, let us say that these form a set I_1 and $I_0 = \overline{I_1}$. We will give a proof by contradiction. To satisfy Equation 19.1 it must hold that $\prod_{i \in I_1} v_i \equiv \prod_{j \in I_0} w_j$. Since v_i for $i \in I_1$ are random elements of $\mathbb{G}_T$ we can say that the product of all but one of these random elements is some other seemingly random element $r' \in \mathbb{G}_T$. So we have $\prod_{i \in I_1 \setminus \{x\}} v_i = r'$. Now the probability that a random $v_{i'} \in_R \mathbb{G}_T$ satisfies $r' v_{i'} \equiv \prod_{j \in I_0} w_j$ is only $\frac{1}{|\mathbb{G}_T|}$, which is negligible for a large enough group $\mathbb{G}_T$. So with negligible probability $z = 0$. □

19.3 Security

We now provide simulation-based proofs for the secure OR protocol. Specifically, we prove that there is no inadvertent leakage of information,

using the proofs from the previous chapter that demonstrate that the shares are computationally indistinguishable from randomness.

To prove that there is no inadvertent leakage of information in our protocol, we follow the formulation of security for deterministic functionalities from Lindell [10]. Since we have already demonstrated correctness, we proceed with a proof for the privacy requirement. We show that (1) a collusion of at most $n-2$ assistants will learn no more than what they knew collectively before colluding, and (2) a collusion of at most $n-2$ parties that includes the leader will only leak the result of the operation. Consequently, an honest party's input stays private for collusions up to $n-2$ parties. We do so by showing that there exists a probabilistic-polynomial time simulator $\mathcal{S}_i$ for each party $i \in \{1,\ldots,n\}$ so that $\mathcal{S}_i(1^\kappa, x_i, f_{i,\mathrm{OR}}(x_1,\ldots,x_n))$ is computationally equivalent to the actual view of the party $\mathrm{view}_i^{\pi_{\mathrm{OR}}}(x_1,\ldots,x_n,\kappa)$, where functionality $f_{i,\mathrm{OR}} : (x_1,\ldots,x_n) \mapsto (x_1 \vee \cdots \vee x_n, \Lambda, \ldots, \Lambda)$ and π_{OR} is the protocol that performs this functionality. In other words, the leader retrieves the logical OR of all inputs, while the other parties receive the empty string Λ. Additionally, the parties receive nonce l as auxiliary information. To demonstrate security over the entire protocol, the π_{OR} protocol encompasses both the one-time setup and a finite number of `OR` operations.

Collusion of assistants

We proceed to prove case (1), where a collusion is made up of a group of at most $n-2$ assistants. We denote the set of colluding parties as $C_{(1)} \subseteq \{2,\ldots,n\}$ and its complement as $\overline{C_{(1)}}$. The goal of this proof is to show a distinguisher cannot confidently tell apart the simulated view from the actual view of the collusion.

Theorem 19.3 *For a collusion of parties $\mathcal{P}_i$ where $i \in C_{(1)}$, there exists a probabilistic-polynomial time simulator:*

$$\mathcal{S}_{(1)}(1^\kappa, x_i, f_{i,\mathrm{OR}}(x_1,\ldots,x_n)) \overset{c}{\equiv} \bigcup_{i \in C_{(1)}} \mathrm{view}_i^{\pi_{\mathrm{OR}}}(x_1,\ldots,x_n,\kappa)$$

Proof. We can construct simulator $\mathcal{S}_{(1)}$ for corrupted assistants:

1. $\mathcal{S}_{(1)}$ simulates $P \in_R \mathbb{G}_1$, $Q \in_R \mathbb{G}_2$ and $g = e(P,Q)$.
2. $\mathcal{S}_{(1)}$ generates a secret and public key pair (sk_i, pk_i) for each $i \in C_{(1)}$ as usual.
3. $\mathcal{S}_{(1)}$ generates a random public key $pk_j \in_R \mathbb{G}_1$ for each $j \in \overline{C_{(1)}}$, which the collusion would normally receive.
4. $\mathcal{S}_{(1)}$ computes v_i for each $i \in C_{(1)}$ as usual.
5. $\mathcal{S}_{(1)}$ outputs x_i, sk_i and v_i for each $i \in C_{(1)}$, as well as pk_j for each $j \in \{1,\ldots,n\}$.

Outputs P, Q, pk_j and sk_i are indistinguishable from what a party would receive ordinarily since they are generated in the same process as they would be in a normal run of the protocol. □

Collusion with the leader

In case (2) the leader is part of the collusion, which requires an extension of the simulator from case (1). This is because the leader also receives each assistant's share v_i, where $i \in \{2, \ldots, n\}$. The goal of this proof is again to demonstrate that a distinguisher cannot confidently tell apart the simulated view from the actual view of the collusion.

Theorem 19.4 *For a collusion of parties $\mathcal{P}_i$ where $i \in C_{(2)}$, there exists a probabilistic-polynomial time simulator:*

$$\mathcal{S}_{(2)}(1^\kappa, x_i, f_{i,\mathrm{OR}}(x_1, \ldots, x_n)) \overset{c}{\equiv} \bigcup_{i \in C_{(2)}} \mathrm{view}_i^{\pi_{\mathrm{OR}}}(x_1, \ldots, x_n, \kappa)$$

Proof. We can construct simulator $\mathcal{S}_{(2)}$ for corrupted leaders and assistants:

1. $\mathcal{S}_{(2)}$ chooses $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and $g = e(P, Q)$ as the leader usually would.
2. $\mathcal{S}_{(2)}$ generates a secret and public key pair (sk_i, pk_i) for each $i \in C_{(2)}$ as usual.
3. $\mathcal{S}_{(2)}$ generates a random public key $pk_j \in_R \mathbb{G}_1$ for each $j \in \overline{C_{(2)}}$, which the collusion would normally receive.
4. $\mathcal{S}_{(2)}$ computes v_i for each $i \in C_{(1)}$ as usual.
5. $\mathcal{S}_{(2)}$ randomly chooses x_j for each assistant $j \in \overline{C_{(2)}}$ and computes v_j as usual, which the collusion would normally receive.
6. $\mathcal{S}_{(2)}$ computes z as usual.
7. $\mathcal{S}_{(2)}$ outputs x_i and sk_i for each $i \in C_{(1)}$, pk_j and v_j for each $j \in \{1, \ldots, n\}$ and finally z.

Again, the outputs P, Q, pk_j and sk_i are indistinguishable from what a party would receive ordinarily since they are generated in the same process as they would be in a normal run of the protocol. Furthermore, outputs v_j and z are generated in the same way as before, but x_i is random this time. A distinguisher could tell apart the simulated view if they could determine what x_i was, but this is infeasible following Theorem 18.2. □

Efficiency

The efficiency of this protocol is based on a party's input. In fact, we show that when the input bit is 0, regardless of the construction, the computation required scales like $O(n\kappa)$, while an input bit of 1 only requires $O(\kappa)$. This means that in the worst case over the whole protocol, computation for every party requires $O(n\kappa)$ time. In the best case an assistant $\mathcal{P}_i$ for $i = 2, \ldots, n$ has $x_i = 1$, so their computation only takes $O(\kappa)$ time.

Communication in this scheme is limited to the assistants. Each assistant only has to send a single element from $\mathbb{G}_T$ to the leader, which requires $O(\lambda)$ bits.

Symmetric construction

For an input bit that is 1 it is enough for a party to select a random element, which takes at most $O(\kappa)$ computations. This can be done more efficiently than generating a random number and performing point multiplication. When the input bit is 0, a party must perform $n-1$ PRF calls, which takes $O(n\kappa)$. In the final step the leader performs n point additions in $\mathbb{G}_T$, which takes $O(n\kappa)$.

Asymmetric construction

For an input bit that is 0, a party must perform $n-1$ pairings, as well as $n-1$ multiplications and 1 exponentiation in $\mathbb{G}_T$. This takes $O(n\kappa)$. For an input bit that is 1, a party only has to choose a random number to compute a random point. This takes $O(\kappa)$. In the final step the leader performs n multiplications in $\mathbb{G}_T$, which takes $O(n\kappa)$.

Multi-party private set and multiset operations

20

In this chapter we discuss one of our main contributions: Non-interactive protocols for MPSI, MPSU, MPMI, MPMU and MPMS, in both exact and approximate variants. For the exact variants we use bitsets, while the approximate variants are based on Bloom filters. Since both these set representations allow for intersections and unions to be computed using bit-wise or bin-wise `OR` and `AND` operations respectively, we present the protocols in a generic fashion, calling those elements that need to be combined using logical operations bins. Note that in all protocols we assume party $\mathcal{P}_1$ is the leader, without loss of generality.

For each operation we describe its protocol and state its efficiency. We rely on Chapters 5 & 6 to argue for these protocols' correctness, and we rely on the security proofs of the secret sharing scheme and the `OR` and `AND` protocols from the previous chapter. We also evaluate the protocols on small instances and compare them against available implementation of previous works. In Chapter 22 we evaluate the protocols on real-life data. We summarize our protocols in Chapter 26.

20.1 Set intersections 71
Efficiency 71
Results 72
20.2 Set unions 72
Approximation 73
Efficiency 75
20.3 Multiset intersection & union 76
Optimizations 77
Efficiency 77
20.4 Multiset sums 78
Count vectors 78
Counting Bloom filters . . . 78

20.1 Set intersections

To perform set intersections, parties encode their sets and aggregate the encodings using an `AND` operation:

MPSI protocol size-hiding

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ encode their set X_i as a bitset or a Bloom filter $\hat{X}_i$.
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ partake in a **Secure `AND` protocol** for each bin j of their set encoding $\hat{X}_i[j]$, in parallel. The leader only finishes the protocol for the bins of its encoding that are filled.
3. The leader $\mathcal{P}_1$ determines the operation's result by checking which of its elements are indeed in the resulting set encoding.

Protocol 20.1: Protocol for securely and non-interactively performing multi-party private set intersections.

Efficiency

The leader only has to perform computations on the elements that are in its set. Recall that in the `AND` protocol the computation complexity for a party depends on its input bit, for both the symmetric and asymmetric construction. Specifically, a 1 requires $O(n\kappa)$ operations, and a 0 only $O(\kappa)$.

Bitset-based

Since each party's input to the secure `AND` protocols contains at most k 1s, the asymptotic computation complexity for an assistant is $O(|\mathcal{U}|\kappa + kn\kappa)$ when $k \ll |\mathcal{U}|$, and only $O(kn\kappa)$ for the leader. Communication-wise, each assistant sends $O(|\mathcal{U}|\lambda)$ bits.

Bloom filter-based

We assume the worst-case scenario, which is that all m bins of the Bloom filter are set to 1, although this should never happen in practice. As a consequence, the parties will perform m runs of the `AND` protocol with an input of 1, which leads to a computation complexity of $O(kn\kappa)$ for a single party because m is bounded by $O(k)$. The communication complexity for an assistant is $O(k\lambda)$.

Results

We implement the exact MPSI protocol for both the symmetric and asymmetric construction. We compare their runtime to the runtime of two other exact MPSI protocols that had an implementation available in Figure 20.1. We performed 5 runs of each protocol and measured its total runtime, ignoring its setup, for an increasing number of parties n. For every experiment we generated sets with k elements randomly drawn from a universe $|\mathcal{U}| = 256$.

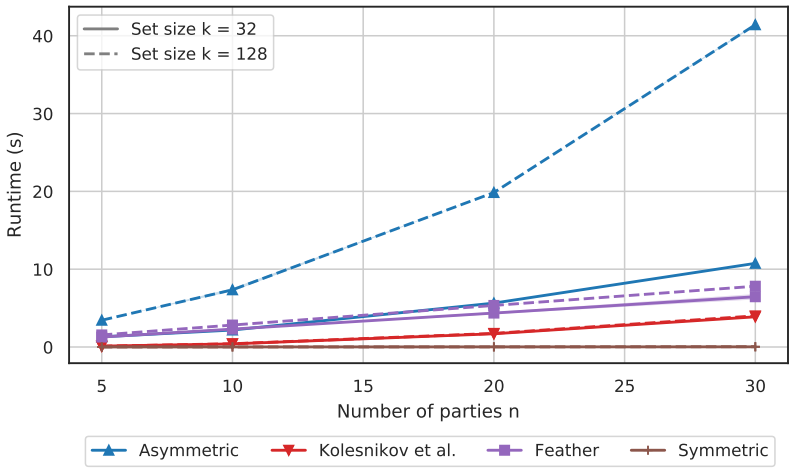


Figure 20.1: Runtime comparison between our exact MPSI protocols and the MPSI protocols of Kolesnikov et al. [5] and Abadi, Terzis, and Dong [40]. The shaded area represents the standard deviation.

20.2 Set unions

To perform set unions, parties again encode their sets but they aggregate the encodings using an `OR` operation:

MPSU protocol size-hiding

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ encode their set X_i as a bitset or a Bloom filter $\hat{X}_i$.
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ partake in a **Secure `OR` protocol** for each bin j of their set encoding $\hat{X}_i[j]$, in parallel.
3. The leader $\mathcal{P}_1$ determines the operation's result by extracting all elements contained in the resulting set encoding.

Protocol 20.2: Protocol for securely and non-interactively performing multi-party private set unions.

Approximation

For set intersections it is straight-forward how to extract the resulting elements from a Bloom filter, but for a set union we must extract all elements encoded within it. Naively, one would query every element in the universe, but this requires $h|\mathcal{U}|$ calls to hash functions. Instead, we propose reversible Bloom filters, which use hashes that can be efficiently reversed to significantly decrease the number of calls to hash functions.

Reversible hash function

Our reversible Bloom filter is based on efficiently reversible hash functions. That is, for the hash function $\mathcal{H}$ there should exist some efficient function $\mathcal{H}^{-}1$ to find the set of preimages of an element y from the codomain:

$$\mathcal{H}^{-1}(y) = \{x \in \mathcal{U} \mid \mathcal{H}(x)\} \tag{20.1}$$

At the same time, the hash function must uniformly distribute its inputs over the codomain. To this end, we define a reversible hash function that exploits the uniformity pre-existing hash functions in a Feistel structure, see Figure 20.2. The round function of the Feistel structure is a pre-existing hash function, where every round uses a different seed. There are d rounds.

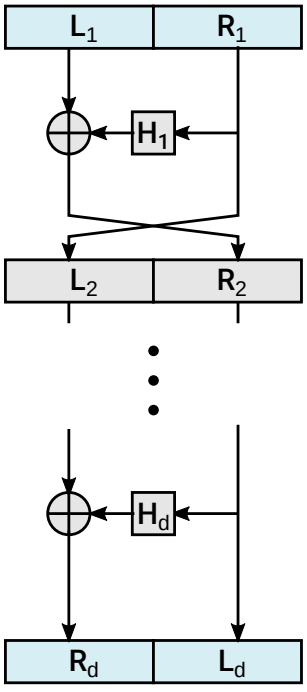


Figure 20.2: Our feistel-structure based reversible hash function with d rounds.

As a first step to ensure that the hash function is indeed uniform, we perform the χ^2 test [61]. This test gives an indication of the 'goodness of fit', which is in our case with regards to the uniform distribution. We use the pre-existing hashes `mmh3` [14] and `xxh3` [15] to both implement and compare our reversible hash function. We define six hash functions: `fm1`, `fm2` and `fm3` are based on `mmh3` and their number denotes the number of rounds d. In the same way we define `fx1`, `fx2` and `fx3` for `xxh3`.

We perform the test as follows. We let each hash function compute the hash of 600 inputs and divide them over 31 bins using a modulus operation, where the inputs are $0, 1c, 2c, \ldots, 599c$. We repeat this process 75 times for increasing c. We use the χ^2 test to evaluate if the inputs were indeed equally distributed over the bins. A bad hash function might for example cause all inputs to be placed into one bin when $c = 31$. Given that the number of bins $m = 31$, the number of inputs $N = 600$ and $|B[j]|$ is the number of elements placed in bin j, we compute the χ^2 as follows:

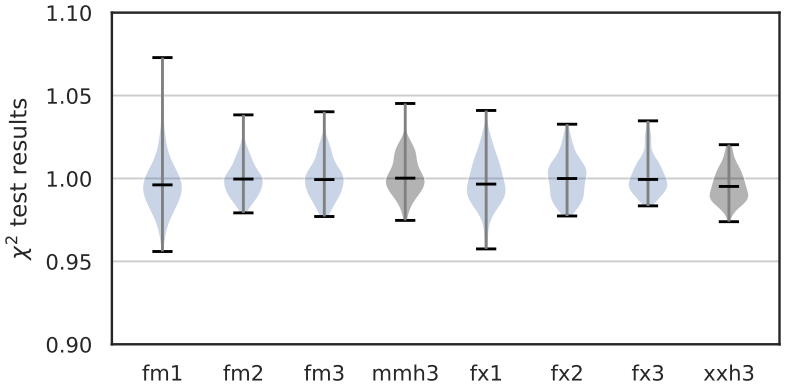


Figure 20.3: χ^2 test results for our reversible hash functions (blue) and reference functions (gray). For example, fm2 denotes an mmh3-based Feistel structure of 2 rounds.

$$\chi^2 = \frac{\sum_{j=0}^{m-1} \frac{1}{2}|B[j]|(|B[j]|+1)}{\frac{N}{2m}(N+2m-1)} \tag{20.2}$$

Figure 20.3 shows a violin plot displaying the min, max, median and densities of the 75 tests performed on each hash function. A perfectly uniform hash function would always achieve $\chi^2 = 1$. Clearly, if the number of rounds $d = 1$, the hash functions perform less uniformly than the reference hash functions, while two or three rounds seem to achieve a similar uniformity in our experiment.

Reversible Bloom filter

The reversible Bloom filter we propose is in all ways identical to a regular Bloom filter, except that it uses reversible hash functions and it supports a fast extraction operation. Algorithm 2 uses the reversible hash functions $\mathcal{H}_i$ with their corresponding $\mathcal{H}_i^{-1}$ for $i = 1, \ldots, h$ to efficiently implement this operation. That is, it computes the elements contained in Bloom filter BF. To check whether element x is contained in BF, we use a function Contains(BF, x).

Assuming that the hash functions are perfectly uniform, in the average case $\mathcal{H}_i^{-1}(j)$ returns a set of $\frac{|\mathcal{U}|}{m}$ elements, as the elements are equally spread out over the bins. The set of candidates C_i corresponding to the hash function with index $i = 1, \ldots, h$ then contains at most $F\frac{|\mathcal{U}|}{m}$ elements, where F denotes the number of filled bins. The probability that an element is contained in each C_i is then at most $\left(\frac{F}{m}\right)^h$. As a result, we expect the final set of candidates C to contain $|\mathcal{U}|\left(\frac{F}{m}\right)^h$ elements, so the number of candidates reduces drastically with the number of hash functions h, although it does increase the number of filled bins F.

Considering that we have reversible hash functions that in turn call d pre-existing hash functions, the candidate finding stage requires Fhd calls to such hash functions. Then, in the average case, we must perform at most $|\mathcal{U}|hd\left(\frac{F}{m}\right)^h$ more calls in the stage where we check membership of the candidates. Recall that the naive method of checking membership of every element in the universe requires $h|\mathcal{U}|$ calls to hash functions. So, while the naive method scales directly with the size of the universe, the

Algorithm 2 Efficient extraction of Bloom filter elements

```
1:  procedure Extract(BF, H_i, H_i^{-1})
2:      ▷ Create sets of candidates for each hash
3:      for i = 1, ..., h do
4:          C_i ← {}
5:      for j = 1, ..., m do
6:          for i = 1, ..., h do
7:              if BF[j] = 1 then
8:                  C_i ← C_i ∪ H_i^{-1}(j)
9:
10:     ▷ Create final set of candidates
11:     C ← C_1 ∩ ··· ∩ C_h
12:
13:     ▷ Check membership of the candidates
14:     result = {}
15:     for c ∈ C do
16:         if Contains(BF, c) then
17:             result ← result ∪ {c}
18:
19:     return result
```

first stage of our method scales with the number of elements in the Bloom filter, and the second stage reduces computation by a factor $d\left(\frac{F}{m}\right)^h$.

To evaluate whether this extraction technique is indeed faster than the naive solution, we implement the reversible Bloom filter in Python for the `mmh3` and `fm3` hash functions. We choose a Bloom filter with $m = 8192$ bins, $h = 4$ hash functions and $N = 10$ inserted elements. We perform 10 experiments with randomly chosen elements from a growing universe $|\mathcal{U}| = 2^{14}, 2^{16}, \ldots, 2^{26}$. The results are in Figure 20.4. Note that the hatched bars are projected, since these take long to compute. The experiments were executed on the machine described in Table 20.1. We believe that the `fm3` hash is faster than the `mmh3` hash in this experiment because it processes only half the number of bits in the Feistel structure.

Note that when executed on the IPv4 space, which contains 2^{32} IP addresses, we project the naive approach using `fm3` to take almost 5 hours to extract the elements from a Bloom filter with the parameters mentioned before, while our approach would take roughly 50 minutes.

Table 20.1: Machine specifications.

Component	Specification
CPU	Intel i7-1065G7
Cores	8×1.30 GHz
OS	64-bit Unix
Memory	16 GB

Efficiency

Bitset-based

In the first step, every party generates a bitset, the cost of which is negligible. All parties perform the secure `OR` protocol over the whole universe, of which each party has at most k 1s as input and $|\mathcal{U}|$ 0s. Consequently, for a fixed universe $\mathcal{U}$, the more elements an assistant has in their set, the faster the protocol. However, assuming $k \ll |\mathcal{U}|$, the asymptotic computation complexity for any party comes out to $O(|\mathcal{U}|n\kappa)$. In the final step, the leader can extract the resulting set in $O(k)$ time. So the MPSU protocol requires $O(|\mathcal{U}|n\kappa)$ computations for each party, as it is dominated by the runs of the secure `OR` protocol. As a small

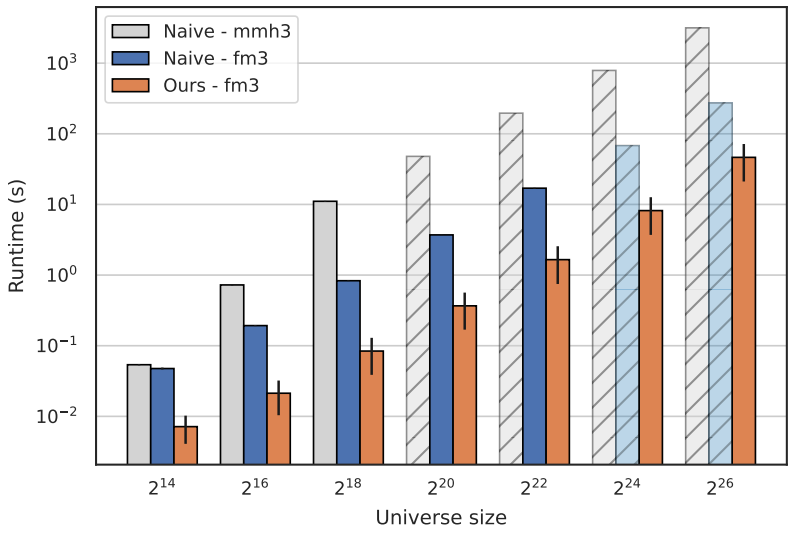


Figure 20.4: Runtime comparison between the naive approach to extract all elements from a Bloom filter and our reversible Bloom filter. Note that the y-axis is logarithmic. The error bars denote the standard deviation.

optimization, the leader can skip elements that are in their own set and add these to the result immediately.

Communication for the MPSU protocol is limited to the number of runs of the secure `OR` protocol: Each assistant has to send $O(|\mathcal{U}|\lambda)$ bits.

Bloom filter-based

Bloom filters allow for significantly less bins than those required for bitsets. For example, when 5 parties each have 100 elements, the total number of elements in the Bloom filter is at most 500. Consider a universe $|\mathcal{U}| = 1{,}000{,}000$ and that we accept an expected number of false positives $\mathrm{E}[\mathcal{F}] = 5$, then using Algorithm 1 we find that the most compact Bloom filter to satisfy $\varepsilon \leq \frac{5}{1{,}000{,}000}$ has $h = 18$ hash functions and $m = 12,719$ bins, which is almost two orders of magnitude less than the number of bits required in a bitset representation for the same universe.

When we determine asymptotic complexities in terms of a constant false positive rate ε, the cryptographic computations for a Bloom filter-based MPSU scale with the set size k, rather than the size of the universe $|\mathcal{U}|$. As mentioned previously, though, extracting the final set does scale with $|\mathcal{U}|$, but these computations do not involve cryptographic operations. In the worst case, all elements in the Bloom filter are mapped to one bin. Then, we must perform the secure `OR` protocol over $m - 1$ zeroes, which scales with nk, so the protocol requires $O(n^2k\kappa)$ operations for each assistant, and $O(n^2k\kappa + |\mathcal{U}|)$ for the leader. Communication-wise, each assistant sends $O(nk\lambda)$ bits.

20.3 Multiset intersection & union

For multiset intersections and unions we apply the transformation as described in Section 3.3 to turn any multiset into a valid set. Next, we can use the set intersection and set union protocols to perform multiset intersections and unions. The protocol for multiset unions is as follows:

MPMU protocol size-hiding

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the multiset-to-set transformation on their input X_i to form set X'_i.
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ partake in a **secure MPSU protocol** on their input X'_i, so that the leader receives the resulting set Z'.
3. The leader $\mathcal{P}_1$ determines the operation's multiset result Z where the multiplicity $\mathcal{M}(j)$ is the maximum multiplicity for each element $j \in Z'$.

Protocol 20.3: Protocol for securely and non-interactively performing multi-party private multiset unions, which can be transformed easily to perform multiset intersections as well.

The protocol for multiset intersections follows similarly, but the protocol uses an MPSI protocol and the result contains the element with the minimum multiplicity.

Optimizations

Based on the leader's set, we can make some optimizations in how the leader aggregates the assistants' encoded sets within the MPSI and MPSU protocols. For multiset intersections, as before, the leader only has to aggregate those bins corresponding to its encoding, and given an element, the leader only has to check multiplicity up to the multiplicity of that item in its own multiset. Similarly, for multiset unions, it is sufficient for the leader to check multiplicity starting from the multiplicity that an element has in its multiset.

Efficiency

Bitset-based

Efficiency for the MPMU and MPMI protocols is similar to the MPSU and MPSI protocols, but we execute them with a larger domain and a larger number of elements. Let us define an upper bound for an element's multiplicity M' of one party. Also, let $W' \geq \max(|x_1|, \ldots, |x_n|)$ denote the maximum cardinality of the input multisets, so the total sum of their multiplicities, then the number of a party's submitted elements is at most W' and the domain size is $M'|\mathcal{U}|$. Note that generally $k \leq W' \leq M'k$. Assuming $k \ll |\mathcal{U}|$, the asymptotic computation complexity for any party in the MPMU protocol comes out to $O(M'|\mathcal{U}|n\kappa)$. Following the same assumption, the asymptotic computation complexity for an assistant in the MPMI protocol is $O(M'|\mathcal{U}|\kappa + W'n\kappa)$, and $O(W'n\kappa)$ for the leader. When it comes to communication, each assistant sends $O(M'|\mathcal{U}|\lambda)$ bits in both protocols.

Bloom filter-based

Similarly, the approximate MPMU and MPMI protocols use the Bloom-filter based MPSU and MPSI protocols, but with a larger domain. In the MPMU protocol, this comes out to a computational complexity of

$O(n^2W'\kappa)$ for an assistant and $O(n^2W'\kappa + M'|\mathcal{U}|)$ for the leader. In the MPMI protocol, the computations for any party has a complexity of $O(nW'\kappa)$. The communication complexity for each assistant is $O(W'\lambda)$ bits in both protocols.

20.4 Multiset sums

The multiset sum is a simple operation to perform in the count vector representation: Instead of storing 1s and 0s, we let each bin in the bins of a bitset denote the multiplicity of the corresponding element as a count vector. After that, we can sum the count vector together bin-by-bin using a non-interactive secure aggregation scheme such as that by Kursawe, Danezis, and Kohlweiss [59], on which we based the non-interactive logic operations. Instead of using a group over the integers, we use the elliptic curve group that we also used for the other protocols. As such, to decrypt the counts in each bin, the leader creates a lookup table with all possible total counts. Additionally, we can approximate the multiset sum using a counting Bloom filter.

MPMS protocol size-hiding

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ encode their set X_i as a count or a counting Bloom filter $\hat{X}_i$.
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute $v_i[j] = w_i[j] + \hat{X}_i[j]$ to $\mathcal{P}_1$ for $j = 1, \ldots, |\hat{X}_i|$, where $w_i[j]$ is the generated mask.
3. All assistants $\mathcal{P}_i$ for $i = 2, \ldots, n$ send their shares $v_i[j]$ for $j = 1, \ldots, |\hat{X}_i|$ to the leader $\mathcal{P}_1$.
4. The leader $\mathcal{P}_1$ determines each bin's result $\hat{X}[j] = \prod_{i=1}^{n} v_i[j]$ and computes the discrete logarithm using a lookup table for $j = 1, \ldots, |\hat{X}_i|$, and extracts the resulting set Z.

Protocol 20.4: Protocol for securely and non-interactively performing multi-party private multiset sums.

Count vectors

For a count vector, we exclude the encoding and extraction steps from the complexities because they do not require cryptographic operations. Creating the shares requires $O(|\mathcal{U}|n\kappa)$ operations for each party. In addition, the leader must aggregate and compare the received curve points to a lookup table, if done linearly this takes at most $O(M'|\mathcal{U}|n\kappa)$ operations. Each assistant sends $O(|\mathcal{U}|\lambda)$ bits to the leader.

Counting Bloom filters

The complexities for counting Bloom filters are similar to those for the MPSU protocol where every bin requires a party to create a share. For the complexities, we assume a constant false positive rate ε. Then, the protocol requires $O(n^2k\kappa)$ operations for each assistant, and $O(M'n^2k\kappa + |\mathcal{U}|)$ operations for the leader. Communication-wise, each assistant sends $O(nk\lambda)$ bits.

Multiset sums within one symmetric share

21

21.1 Prime factors as multisets . . 79
Encoding 79
Homomorphism 80
Decoding 80
21.2 Multiset sum protocol 82
Group selection 82
Protocol description 82
Security 83
Efficiency 83
21.3 Results 84
Run time 84
Communication 85

In the case of a symmetric setup we realize a multiset sum protocol where each party only has to send one share. As such, a party only has to perform the expensive mask generation algorithm once. The intuition behind our protocol is a multiset encoding that is multiplicatively homomorphic, so that when two encodings are multiplied they produce the encoding of the corresponding multiset sum. To fit such an encoding within a single share, the share must be thousands of orders larger than regular shares. However, we demonstrate that by choosing a particular multiplicative group we can perform all the necessary operations in the order of milliseconds, without compromising security. In fact, the protocol is information-theoretically secure in the random oracle model.

While this protocol works in the symmetric setup, we cannot execute it in the asymmetric setup as that would require solving the discrete logarithm problem in the underlying elliptic curve. However, if the number of possible outcomes is limited to the point where the receiving party can iterate over them and a pairing-friendly curve exists that fits the encoding, then this setup is feasible. The receiving party can extract the result simply by comparing if the received encoding is the same as one of the possible inputs.

This research was done in collaboration with Martin Koster, a fellow thesis student, under the supervision of Zekeriya Erkin. Martin's thesis [62] revolves around using superincreasing sequences to count elements using additive secret sharing and additively homomorphic cryptosystems, within a single ciphertext. In this work, however, we use a multiplicative encoding that scales more efficiently for larger universes.

[62]: Koster (2021), 'Multi-Functional Privacy-Preserving Data Aggregation'

21.1 Prime factors as multisets

In this section, we present a technique for encoding multisets in a single group element using prime factors. To limit the size of the group required, we consider two parameters: M is the maximum multiplicity of any element in the universe and K is the maximum number of different elements that are present in a multiset. In other words, for any element x in multiset X, the multiplicity is bounded: $\mathcal{M}_X(x) \leq M$, and there are only K distinct elements for which it holds that $\mathcal{M}_X(x) \geq 1$.

Encoding

The prime factor-based encoding is based on the notion that each number has a unique prime factorization. In fact, the prime factors of a number form a multiset. We exploit this by mapping the elements from the

universe to the first $|\mathcal{U}|$ primes $\mathcal{U} \mapsto \{p_0, \ldots, p_{|\mathcal{U}|-1}\}$. Specifically, the encoding $\hat{X}$ of multiset X is as follows:

$$\hat{X} = \prod_{i=0}^{|\mathcal{U}|} p_i^{\mathcal{M}_X(i)}. \tag{21.1}$$

Homomorphism

As a result, the encoding is multiplicatively homomorphic. Consider three multisets X_1, X_2 and X_3, then the multiplication of multiple encoded multisets results in the encoding of the multiset sum:

$$X = X_1 \uplus X_2 \uplus X_3 \tag{21.2}$$

$$\hat{X} = \prod_{i=0}^{|\mathcal{U}|} p_i^{\mathcal{M}_{X_1}(i)} \prod_{i=0}^{|\mathcal{U}|} p_i^{\mathcal{M}_{X_2}(i)} \prod_{i=0}^{|\mathcal{U}|} p_i^{\mathcal{M}_{X_3}(i)} \tag{21.3}$$

$$= \prod_{i=0}^{|\mathcal{U}|} p_i^{\mathcal{M}_{X_1}(i)+\mathcal{M}_{X_2}(i)+\mathcal{M}_{X_3}(i)} \tag{21.4}$$

To use this encoding in a finite field we must select a modulus that will never be surpassed. The modulus required to store such a multiset is decided by the largest possible encoding. Such an encoding happens when the top K coefficients appear M times. Every modulus m that satisfies $m \geq (p_{|\mathcal{U}|-1})^{MK}$ satisfies this constraint.

From the prime number theorem it follows that there is an upper bound of the xth prime: $p_x < x(\ln x + \ln \ln x) = O(x \ln x)$ [63], which holds for $6 \leq x$. For this reason, the modulus scales with $O((|\mathcal{U}| \ln |\mathcal{U}|)^{MK})$. In addition, these first x primes can be efficiently generated using the Sieve of Eratosthenes, which requires $O(x \log \log x)$ operations [64].

Decoding

To find the multiset encoded in $\hat{X}$ we must factor it. The most efficient special-purpose factoring algorithm for numbers with small prime factors is the Elliptic Curve factorization Method (ECM) [65]. This method's complexity is proportional to the smallest prime factor p of the integer it is factoring. Specifically, it requires $L_p[\frac{1}{2}, \sqrt{2}]$ operations in the L-notation, which is analogous to a complexity of:

$$O\left(\exp \sqrt{(\ln p \ln \ln p)(1 + O(1))}\right) \tag{21.5}$$

ECM consists of two stages that require parameters B_1 and B_2, respectively. Ideally, these parameters would be chosen based on the size of the smallest prime factor p. The higher B_1 is, the longer the stage takes, and the same applies to the second stage for B_2. Specifically for our case, we apply the algorithm to composites that only contain primes that are in the order of several digits. For this reason, we start with $B_1 \leftarrow 10$ and increase B_1 by 100 for every unsuccessful application of the algorithm. ECM libraries have built-in functionalities to select values for B_2 that result in both stages taking approximately the same amount of time. Algorithm 3

shows how we apply ECM to find factors in composites with many small prime factors.

Algorithm 3 Method for finding a factor in composite number n

```
1: procedure FindFactor(n)
2:     B_1 ← 10
3:     while true do
4:         f ← ECM(n, B_1)
5:         if f then
6:             return f
7:         B_1 ← B_1 + 100
```

We present our actual routine for extracting the multiset encoded in $\hat{X}$ in Algorithm 4. After running the algorithm, the resulting multiset contains the prime factors and their multiplicity, which can be mapped directly back to the original universe $\mathcal{U}$ using the prime sieve.

On a high level, the decoding algorithm works as follows: First, we create an empty multiset. We then apply the FindFactor algorithm to find one factor f of the encoding. If the factor is small enough to be factored by the sieve, we do so, otherwise the algorithm calls Decode on it again. After this, F contains the prime factorization of f. For each of those primes we check how often they occur in the encoding $\hat{X}$. We then add these factors and their multiplicity to the multiset, while removing these factors from the encoding. If the remainder is now small enough to factor using the sieve, we do so, otherwise we do a recursive call again. After this step, M contains all prime factors of $\hat{X}$ and their multiplicity.

Algorithm 4 Recursively decode $\hat{X}$ using sieve S into prime factors

```
1: procedure Decode(X̂, S)
2:     M ← {}                                  ▷ Create an empty multiset
3:     if X̂ = 1 then
4:         return M
5:     f ← FindFactor(X̂)                       ▷ Call to Algorithm 3
6:     if S.CanFactor(f) then                  ▷ If f is small enough to use the sieve
7:         F ← S.Factor(f)
8:     else
9:         F ←Decode(f, S)
10:    r = X̂                                   ▷ the remaining encoding that is left to decode
11:    for p ∈ F do                            ▷ For each prime p
12:        c ← 0
13:        while r ≡ 0 (mod p) do              ▷ While remainder r is divisible by p
14:            c ← c + 1
15:            r ← r/p
16:        M ← M ⊎ {p_c}                       ▷ Prime p occurred c times
17:    if S.CanFactor(r) then                  ▷ If r is small enough to use the sieve
18:        M ← M ⊎ S.Factor(r)
19:    else
20:        M ← M⊎ Decode(r, S)
21:    return M
```

21.2 Multiset sum protocol

Now, we present our protocol that outputs the sum of multisets non-interactively. There are n parties that want to compute the multiset-sum of their private multisets. There is a known maximum multiplicity M and maximum number of elements K that apply to the resulting multiset. During the protocol, each party sends one message to an aggregator who aggregates the received messages to privately retrieve the resulting multiset.

Group selection

Rather than using the group discussed in Section 18.2, this protocol requires a multiplicative group over the integers. Preferably, the modulus contains few prime factors, since any prime factor in the modulus is one that we cannot use for the multiset encoding. For this reason, we describe our protocols for the multiplicative group of integers modulo 2^x, for some $x > 40$ to ensure that the group is large enough to provide at least 40 bits of statistical security. The elements in this group are the coprimes of its modulus, which are the odd numbers.

Unless $x = 2$, a group with modulus 2^x is not cyclic [66], but this is not strictly necessary for our purposes. The choice of this group has several other favorable properties, though:

- The modulus reduction on a computer can be achieved by discarding all but the least significant x bits.
- All outputs from the PRNG can be coerced to group elements by setting their least significant bit to 1.
- There exist efficient special-purpose algorithms for computing the multiplicative inverse in groups of this form [67] .
- The only prime factor in the modulus is 2: We can use all primes apart from the first.

[67]: Dumas (2014), 'On Newton-Raphson Iteration for Multiplicative Inverses Modulo Prime Powers'

Protocol description

Our protocol consists of a setup, which is performed only once, followed by four stages, namely encoding, masking, aggregation and decoding. The latter four stages are performed for every run of the protocol. During this protocol, we assume the aggregator and each party acts according to the honest-but-curious security model.

During setup, the parties sieve the first $|\mathcal{U}|$ prime factors as required for the encoding scheme. The initialization of the encoding scheme returns the list of coefficients and a modulus. The final modulus used in the protocol is set to 2^x, where x is the lowest value for which 2^x is larger than the required modulus. Additionally, to ensure a statistical security of 40 bits, $x > 40$.

All parties also have access to a PRF. We force every output of this PRF to be an element of $\mathbb{Z}_m^\times$ by setting the least significant bit to 1. To realize a non-interactive secret-sharing setup, every pair of parties shares a common seed, denoted with $S_{i,j} = S_{j,i}$ for parties i and j. In the

MPMS protocol size-hiding

1. **Encoding**: Each party $\mathcal{P}_i$ for $i = 1, \ldots, n$ encodes its multiset X_i according to the encoding scheme as $\hat{X}_i$.
2. **Masking**: Each party $\mathcal{P}_i$ for $i = 1, \ldots, n$ generates a mask w_i using nonce t, and uses it to create share $c_i = \hat{X}_i + w_i$, and each assistant $\mathcal{P}_i$ for $i = 2, \ldots, n$ sends it to the leader $\mathcal{P}_1$.
3. **Aggregation**: The leader $\mathcal{P}_1$ aggregates the shares into the final encoded multiset $\hat{X}_t = \prod_{i=1}^{n} c_{i,t}$.
4. **Decoding**: The leader $\mathcal{P}_1$ decodes $\hat{X}$ using Decode and returns the result.

Protocol 21.1: Protocol for securely and non-interactively performing multi-party private multiset sums.

remainder of this work we assume the parties have successfully executed the setup stage.

Security

The security is given by the security of the symmetric non-interactive secret sharing construction that we discuss in Section 18.2.

Efficiency

Next, we describe the efficiency of our protocol. We assume that the one-time setup has been executed so that every pair of parties has a shared seed and they have executed the prime sieve. We specify the total complexity of all parties, collectively.

Encoding In the encoding stage, the parties collectively encode K elements at most M times. This is synonymous to MK multiplications of prime factors in the multiplicative group.

Masking To compute the masks, each party performs $n - 1$ multiplications on numbers in the multiplicative group. These numbers are generated by $n(n-1)$ calls to the PRF, and half of them must be inverted.

Aggregation The aggregation is performed by multiplying the n secret shares together in the multiplicative group.

Decoding The recursive algorithm makes K calls to FindFactor to factor out each prime factor. To eliminate every factor from the encoding, this requires dividing out each prime factor at most M times, so in addition, MK division are necessary.

The computation complexity of each stage in our protocol is summarized in Table 21.1. While the operations in the table scale linearly, one should note that the operations themselves are not strictly linear. In particular,

	Encoding	Masking	Aggregation	Decoding
Multiply / divide	$O(MK)$	$O(n)$	$O(n)$	$O(MK)$
Modular inverses	-	$O(n)$	-	-
PRNG calls	-	$O(n)$	-	-
FindFactor calls	-	-	-	$O(K)$

Table 21.1: Total computation complexity of our protocol, n is the number of parties.

the FindFactor algorithm scales sub-exponentially with the lowest prime factor p, for which it holds that $p \leq |\mathcal{U}| \ln |\mathcal{U}|$. We denote the complexity of finding a factor by $f(u)$, which is the complexity of ECM on a composite number with smallest prime factor $p = u \ln u$ for a large enough B_1. Moreover, the multiplications scale with the size of the modulus m of the multiplicative group, which is bounded by $O((|u| \ln |u|)^{MK})$.

We can write the final computational complexity for the encoding and masking steps combined as $O(MK + n)$, and the computational complexity for aggregation and decoding as $O(n + Kf(|\mathcal{U}|))$. If we substitute K for the worst-case scenario where each party has a disjoint set of k elements, then $K = nk$ so the complexities becomes $O(Mnk)$ and $O(nkf(|\mathcal{U}|))$, respectively.

With respect to the communication complexity, every run, $O(n)$ masked encoded multisets are sent. The size of a masked encoded multiset depends on the aforementioned modulus m. The size of a masked encoded multiset is $\log_2(m)$ bits, which we denote by a function $G_{M,K}(u) = O(\log_2(|u| \ln |u|)^{MK})$. Every run, therefore requires $O(nG_{M,K}(|\mathcal{U}|)$ bits of communication.

21.3 Results

We have implemented the prime factor-based protocol in Rust using bindings for the C libraries GMP* provided by rug†, and GMP-EMC‡, which provide the arbitrary-length modular arithmetic and the elliptic curve factoring method respectively. Our implementation uses a single thread to execute all parties sequentially, but in practice these parties could perform computation in parallel

Note that this implementation uses a pseudo-random number generator (PRNG) to implement the pseudo-random function, in this case the Mersenne twister. Unfortunately, this PRNG is not cryptographically secure [68]. We believe that it is sufficient for a proof of concept because we only call the PRNG once, but for an actual implementation that reuses the setup multiple times, the PRF should be cryptographically secure, such as the one discussed in Section 18.2.

Run time

We perform two experiments to measure the run time of our protocol. In the first experiment we set $K = 10$ and $M = 50$, and in the second experiment $K = 25$ and $M = 50$. We measure the run time of three stages

* https://gmplib.org/
† https://gitlab.com/tspiteri/rug
‡ http://ecm.gforge.inria.fr/

in our protocol. The setup stage contains both the setup of each party's seeds as well as a prime sieve. Then, the encryption stage lets each party encode their multiset and form a secret share. Finally, the decryption stage consists of a party who aggregates the shares and decodes the result by factoring it. Our experiments were performed on a the machine described in Table 20.1. The results are in Figure 21.1.

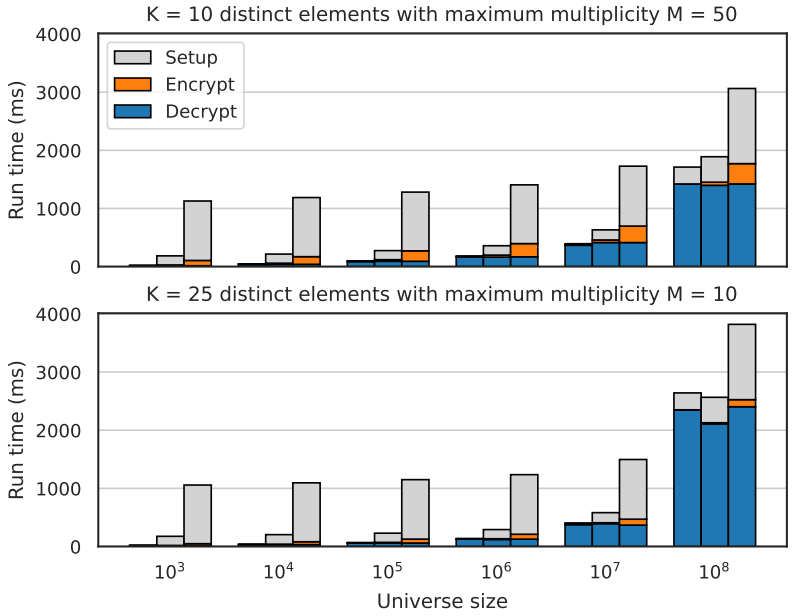


Figure 21.1: Run time for $n = 5$, $n = 20$ and $n = 50$ parties, averaged over 50 runs.

Communication

The size of the secret share is entirely dependent on the size of the modulus required for the encoding scheme, and each party sends exactly one secret share. Table 21.2 shows the size of such a secret share for the same parameters as in our run time experiments. At least for these instances, the communication cost for each party is limited to several kilobytes. Note that while the modulus grows rapidly for a growing universe, the size of the modulus in bits grows significantly slower.

	$\lvert\mathcal{U}\rvert = 10^3$	$\lvert\mathcal{U}\rvert = 10^4$	$\lvert\mathcal{U}\rvert = 10^5$	$\lvert\mathcal{U}\rvert = 10^6$	$\lvert\mathcal{U}\rvert = 10^7$	$\lvert\mathcal{U}\rvert = 10^8$
$K = 10, M = 50$	6,473	8,339	10,155	11,943	13,710	15,463
$K = 25, M = 10$	3,233	4,169	5,078	5,972	6,855	7,732

Table 21.2: Number of communicated bits per party for a growing universe $\mathcal{U}$.

Private collaborative detection of malicious IP addresses

22

22.1 Dataset 86
Dataset generation 87
Subnet datasets 87
22.2 Results 88
Asymmetric construction . . 88
Symmetric construction . . . 89

Cybercrime continues to grow every year due to an increase in the number of Internet users, the number of connected devices, and digitalization [69]. Damage to the world economy caused by cybercrime is expected to reach 6 trillion US dollars per year in 2021 [69]. These crimes affect a large number of organizations, but those same organizations may be discouraged from collaboratively addressing them out of concerns over privacy, a lack of trust and reputation [70].

Specifically, cybercrime often involves malware: Adware, ransomware, trojans, and botnets are among the most popular [69]. These types of malware can be commanded to perform malicious tasks externally through so-called Command & Control (C2) servers. If the IP address of a C2 server is known, organizations can blacklist it [71], and even though devices stay infected, they might stop sending spam or taking part in DDoS attacks, for example. However, identifying these C2 servers among benign network traffic can be a daunting task for one organization alone, especially considering the evasion techniques that modern malware employs [69].

One way for organizations to work together to identify C2 servers and other malicious hosts is by sharing the sets of IP addresses that were communicated with from their network. However, these sets of IP addresses contain sensitive information: The IP addresses reveal the websites that users have visited, details about the network architecture and even location information of peers through IP geolocation. Besides, if the information could reveal that an organization was indeed infected by malware, the organization may be reluctant in sharing it in the first place, for the sake of their reputation [70]. For this reason we employ our MPSI, MPSU, MPMI, MPMU and MPMS to privately perform statistical analyses on this IPv4 data. We evaluate them on real-life data.

22.1 Dataset

The Stratosphere Lab creates real long-term malware captures that consist not only of malicious behavior but importantly also of benign user behavior. For this reason, we choose to run our protocols on the Mixed Captures dataset from Stratosphere [72] , which contains five malware captures of adware, ransomware, and trojans, each spanning one or four days. From these captures, we extract the IP addresses of outgoing packets using Wireshark and we record the multiplicity of these IP addresses as the number of packets to create multisets. We refer to the resulting data as IP/32-REAL. Depending on the application, one could instead extract only incoming packets.

[72]: Stratosphere (2015), *Stratosphere Laboratory Datasets*

Dataset generation

The IP/32-REAL dataset contains real data but it only features five parties. To evaluate our protocols on instances with more cooperating organizations with longer captures we designed a simple procedure to generate similar looking data for larger problem sizes. The procedure inputs the number of IP addresses per party I, the number of parties n, the maximum bias of a party $B_{\max}$ that is used to distribute the addresses over the parties, the ratio of the intersection size versus the total size $R_{\text{intersection}}$ and the ratio of the union R_{union}. The procedure builds n sets denoted by S_i for $i \in \{1, \ldots, n\}$:

1. Select a random union $S_\cup \subset_R \{0, \ldots, 256^4\}$ of size $|S_\cup| = InR_{\text{union}}$.
2. Select a random intersection $S_\cap \subset_R S_\cup$ of size $|S_\cap| = InR_{\text{intersection}}$.
3. Select a random bias for each party $B_i \in_R \{1, \ldots, B_{\max}\}$. We always assign $B_{\max}$ to the leader and 1 to the last assistant.
4. Assign intersection $S_\cap$ to each party's set S_i.
5. Distribute the elements in union $S_\cup$ over the party's sets with weighted random chance B_i.
6. Conclude the sets by distributing random elements from $S_\cup$ over the parties with a weighted random chance B_i until $|S_1|+\cdots+|S_n| = In$.

The previous procedure generates sets, but not multiplicities. To do so, we sample from a Pareto distribution, which has become popular in internet traffic modeling due to its ability to capture the heavy tail of internet traffic count data distributions [73] . In our implementation we use a Pareto type I distribution with parameter $\alpha = 0.55$. We then sample the desired number of multiplicities from this distribution until the samples lie within 10% of a desired mean μ. We use these methods to generate another small, medium, and large-sized dataset. The small dataset is intended to closely resemble the real dataset to demonstrate the effectiveness of the described generator.

[73]: Chen et al. (2015), 'Performance Evaluation of a Queue Fed by a Poisson Lomax Burst Process'

In the final datasets, we use the order of magnitude of packet counts rather than the precise multiplicity. To ensure that a multiplicity of 1 does not result in an order of magnitude of 0 we define the order of magnitude as $\text{OM}(x) = \lfloor \log x \rfloor + 1$. The parameters of the final datasets can be seen in Table 22.1, the parameters marked with (*) cannot be higher given the domain size.

Subnet datasets

In some cases, it suffices to analyze subnets rather than full IP addresses, for example for entities higher up in the internet hierarchy. Another reason for considering IP ranges is to preserve privacy through lower granularity. For example, when the goal is to collaboratively identify malicious servers through set and multiset operations, then it could be enough to limit the search space to a set of suspicious IP ranges, rather than intending to immediately blacklist the IP addresses resulting from an operation. These subnet IP addresses reduce the domain size by several orders of magnitude. We use the CIDR block representation of appending a slash character after the IP address followed by the number of prefix bits that we consider. To generate datasets for IP/12 subnets we

changed the generation parameters so the subnet data has roughly the same total count as the full IP data. Moreover, instead of sampling from $\{0, \ldots, 256^4\}$ we sampled the possible values from $\{0, \ldots, 2^{12}\}$.

Table 22.1: High-level parameters of the datasets. M_{OM} and W_{OM} indicate the maximum multiplicity and the maximum total multiplicity for one party after applying the order of magnitude operation.

Dataset	I	n	$B_{\max}$	R_{union}	$R_{\text{intersection}}$	μ	Smallest set	Largest set	M_{OM}	W_{OM}	|union|	|intersection|
IP/32-real	975.4	5	9.5	79%	1.4%	539	280	2659	6	5028	3842	69
IP/32-small	975	5	10	79%	1.4%	543	266	2057	6	2862	3919	68
IP/32-medium	1500	20	12	85%	0.7%	4114	440	2968	8	4133	25710	210
IP/32-large	2500	50	15	90%	0.2%	10305	514	4410	8	6193	112750	250
IP/12-real	230.2	5	6.7	61%	3.2%	2285	84	563	6	1180	706	37
IP/12-small	230	5	7	61%	3.2%	2187	86	372	6	517	686	36
IP/12-medium	375	20	9	20%*	1.5%	16339	167	583	8	811	1394	108
IP/12-large	625	50	12	5%*	0.5%	40637	217	893	8	1268	1497	146

22.2 Results

We developed a reference implementation for our protocols in the asymmetric construction in C++ using the MIRACL library, and we evaluated their runtime and bandwidth. For the symmetric construction, we built an implementation in Rust. Our test setup is as follows: Each party is assigned a single thread to run on, so the parties are single-threaded but they are executed in parallel. We measure the runtime of the share creation part until the last assistant finishes and proceed to measure the set extraction part separately. We do not simulate a communication delay. In our implementations we have counted the full size of the curve points for bandwidth, but in practice the bandwidth can be halved by compressing curve points before transmitting them. For the Bloom filters we choose the `mmh3` hash.

For the exact protocols, we use the 12-bit subnets resulting in a domain size $|\mathcal{U}| = 4096$. Had we used the full IP addresses of 32-bits this would result in domain size $|\mathcal{U}| \approx 4.3 \times 10^9$. Since our approximate protocols do not scale with the domain size we used the full 32-bit IP addresses to evaluate them. Table 22.2 contains the parameters selected using Algorithm 1.

Asymmetric construction

The experiments were executed once on the machine described in Table 22.3, and the results can be found in Table 22.4. Headers t_{shr} and t_{extr} indicate the share-creation time of the slowest assistant and the leader's extraction time in seconds, respectively. Header b_{tot} denotes the total bandwidth required in megabytes.

For the real and small datasets, the MPSI protocol takes less than half a minute to compute, but other operations take in the order of minutes to hours. Still, we believe the asymmetric construction can be useful in cases where deploying the symmetric construction would be infeasible, or where the keys are already in place. This may be the case when multiple parties want to compute such an operation incidentally.

Table 22.2: Compact Bloom filter parameters, where we chose N and computed h and m using Algorithm 1 to satisfy $\varepsilon \approx 1\%$.

	N	h	m
IP/32-small	5500	7	52768
IP/32-medium	5000	7	47941

Table 22.3: Machine specifications.

Component	**Specification**
CPU	Intel Xeon
Cores	30 × 3100 MHz
OS	64-bit Unix
Memory	120 GB

Table 22.4: Runtime and bandwidth for the multi-party private set and multiset operations on real as well as synthetic malicious host data for the asymmetric construction, written in C++. We omit experiments that would take several hours to compute.

	REAL			SMALL			MEDIUM			LARGE		
	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]
Exact operations on IP/12												
MPSI	15	9	0.5	17	6	0.5	76	47	2.4	581	172	6.1
MPSU	96	61	0.5	95	65	0.5	574	281	2.4	2869	602	6.1
MPMI	84	68	3.0	68	43	3.0	208	485	19.0	1037	1626	49.0
MPMU	577	602	3.0	577	550	3.0	-	-	-	-	-	-
MPMS	98	77	0.5	98	77	0.5	614	328	2.4	3344	775	6.1
Approximate operations on IP/32												
MPSI	357	395	3.5	308	322	3.5	1775	2415	16.7	-	-	-
MPMI	672	808	6.4	477	640	6.4	2550	4091	27.8	-	-	-

Table 22.5: Runtime and bandwidth for the multi-party private set and multiset operations on real as well as synthetic malicious host data for the symmetric construction, written in Rust. We omit experiments that would take several hours to compute.

	REAL			SMALL			MEDIUM			LARGE		
	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]	t_{shr}[s]	t_{extr} [s]	b_{tot} [MB]
Exact operations on IP/12												
MPSI	0.07	0.00	0.6	0.07	0.00	0.6	0.20	0.00	2.5	0.68	0.01	6.3
MPSU	0.12	0.01	0.6	0.11	0.01	0.6	0.49	0.02	2.5	2.21	0.05	6.3
MPMI	0.43	0.00	3.8	0.42	0.00	3.8	1.82	0.01	20.0	4.92	0.02	50.0
MPMU	0.80	0.01	3.8	0.79	0.01	3.8	5.99	0.06	20.0	26.1	0.15	50.0
MPMS	0.13	0.04	0.6	0.13	0.04	0.6	0.71	0.20	2.5	3.29	0.44	6.3
Approximate operations on IP/32												
MPSI	0.56	0.01	4.4	0.57	0.01	4.4	2.65	0.04	17.6	15.08	0.12	65.9
MPMI	1.05	0.01	8.1	0.99	0.01	8.1	4.16	0.04	29.3	20.64	0.12	87.8

Symmetric construction

We also evaluate the performance in the symmetric construction, albeit on a less powerful machine. The specifications of this machine are described in Table 20.1. We repeated the experiments 5 times and report the means. The results can be seen in Table 22.5. The symmetric construction runs almost two orders of magnitude faster than the asymmetric construction. We believe that this is the case for several reasons. First, the symmetric construction does not require the computation of expensive pairing functions. Secondly, the Montgomery curve is extremely fast on modern CPUs. Finally, there may be a difference in the amount of optimizations that the C compiler and the Rust compiler can make to the implementations.

Interactive protocols

While non-interactive protocols are simplest to deploy, we cannot perform all operations non-interactively. In this part we propose interactive protocols for common operations that still require less interactions than state-of-the-art protocols. We demonstrate their effectiveness in comparing 'threat intelligence providers'. This part contains our own contributions.

Set intersection & union cardinality

23

23.1 Shuffle-decrypt protocol . . . 91
Protocol description 92
Using a threshold t 92
Correctness 93
Security 93
Efficiency 94
23.2 MPSO-CA protocol 94
Approximation 95
Efficiency 95

In the previous part we proposed the first non-interactive protocols for operations such as set intersections and set unions. However, in Chapter 1 we also discussed applications where parties may only find out the cardinality of set intersections and unions. It turns out that it is not trivial to alter our previously proposed protocols to hide the resulting set and return only its size. A simple solution would be to shuffle the bits in the resulting bitset, or the bins in the resulting Bloom filter to estimate the cardinality without being able to identify the resulting elements, but this leads to a contradiction: the leader must receive the bins in their original order to aggregate them, but the bins must be shuffled before they are aggregated or the elements would be revealed.

Instead, we propose a protocol using an asymmetric construction that is based on a threshold cryptosystem, so we can covertly shuffle the encrypted bins, which is similar to the protocol by Debnath et al. [37]. As a consequence, the resulting protocol requires interaction, since the ciphertexts must be decrypted. Fortunately, we propose a shuffle-decrypt protocol to simultaneously shuffle and decrypt, so each party only has to come online once in this stage. To the best of our knowledge, this is a novel protocol that may also be of independent interest. We provide results based on practical scenarios in Chapter 25.

23.1 Shuffle-decrypt protocol

When shuffling and decryption are performed separately using a threshold cryptosystem, the shuffling requires each party to sequentially permute the ciphertexts and re-randomize them, then the decryption requires each party to partially decrypt in parallel. If a party is unavailable in any of those two stages, the protocol cannot progress. By shuffling and decrypting in one pass, we require parties to be online for only one stage and we save the parties from sending unnecessary messages.

For our shuffle-decrypt protocol we adapt the classic ElGamal cryptosystem to behave like an (n, n)-cryptosystem. The intuition behind this protocol is to let each party in turn shuffle the ciphertexts and 'peel back' one layer of encryption. At the same time, such a party must re-randomize the ciphertexts to ensure that the ciphertexts that they received are indistinguishable from those that they send to the next party.

Actually, this protocol is more like an aggregate-shuffle-decrypt protocol, since the first step homomorphically aggregates the submitted ciphertexts. To use it as a shuffle-decrypt protocol, the leader submits a ciphertext corresponding to the message to be recovered, and the assistants all submit an encryption of the identity, for example.

Protocol description

We proceed with a description of the protocol, where we use a notation for indexing vector to ease interpretation. So, for a vector x, $x[i]$ denotes the ith element of x. Additionally, q is the order of the cyclic group $\mathbb{G}$ of the ElGamal cryptosystem. Finally, each party $\mathcal{P}_i$ has a secure permutation function $\pi_i(\_)$ that takes a vector like x and returns a permuted vector of the same length. We assume that each party has a private key $k_i \in_R \mathbb{Z}_q$ and all parties known its corresponding public key h_i.

Aggregate-shuffle-decrypt protocol

1. Each party $\mathcal{P}_i$ for $i = 1, \ldots, n$ sends m ciphertexts of messages M_i to the leader $\mathcal{P}_1$, encrypted with their own public key h_i using randomness $y \in_R \mathbb{Z}_q$, and for $j = 1, \ldots, m$:

$$\langle \alpha_i[j], \beta_i[j] \rangle = \langle y_i[j]g, M_i[j] + y_i[j]h_i \rangle$$

2. Leader $\mathcal{P}_1$ constructs a vector of tuples $\alpha[j] = \langle \alpha_1[j], \ldots, \alpha_n[j] \rangle$ and $\beta[j] = \sum_{i=1}^{n} \beta_i[j]$, and sends them to party $\mathcal{P}_n$.
3. In turn, each party $\mathcal{P}_i$ for $i = n, \ldots, 2$ shuffles and partially decrypts the ciphertexts, sending the results to the next party $\mathcal{P}_{i-1}$:
   - Party $\mathcal{P}_i$ uses permutation function π_i to permute $\alpha \leftarrow \pi_i(\alpha)$ and $\beta \leftarrow \pi_i(\beta)$.
   - Party $\mathcal{P}_i$ removes its corresponding entry from each tuple in α and randomizes it like:

$$\alpha[j] \leftarrow \left\langle \alpha_1[j] + y'_{i,1}[j]g, \ldots, \alpha_{i-1}[j] + y'_{i,i-1}[j]g \right\rangle,$$

     for $j = 1, \ldots, m$, using randomness $y'_i[i'] \in_R \mathbb{Z}_q$.
   - Party $\mathcal{P}_i$ partially decrypts each $\beta[j]$ and uses the same randomness $y'_i[i']$ to compute:

$$\beta[j] \leftarrow \beta[j] - k_i\alpha_i[j] + \sum_{i'=1}^{i-1} y'_{i,i'}[j]h_{i'}$$

4. Leader $\mathcal{P}_1$ decrypts $M = \beta[j] - k_1\alpha_1[j]$ for $j = 1, \ldots, m$.

Protocol 23.1: Protocol to aggregate m messages of n parties using ElGamal, and both shuffle and decrypt them in one pass.

Using a threshold t

To save on bandwidth, one can reduce the collusion resistance of the protocol to a certain threshold t. As such, the parties choose t parties that perform the shuffle-decrypt protocol. For those parties, the protocol is as described above, but the other parties must encrypt their messages with the public keys of each decrypting party to achieve collusion threshold t. To stay consistent with our other proposed protocols, we do not use this alteration in our evaluation. Instead, the protocol is secure against $n - 1$ colluding parties.

Correctness

Let us denote the composition of permutations as $\circ$, and the final permutation as $\pi = \pi_1 \circ \cdots \circ \pi_n$. To prevent complicated notation, we say that when these permutation functions are supplied with an index, that index is mapped to the permuted index. Then, the protocol is considered correct when it holds that $M[\pi(j)] = M_1[j] + \cdots + M_n[j]$ for all $j = 1, \ldots, m$. Since the permutation functions shuffle both α and β according to the same permutation, their elements are not shuffled relative to each other. For this reason we provide a correctness proof where all permutation functions are the identity function, to ease notation.

Theorem 23.1 *When all permutation functions are the identity, it holds that* $M[j] = \sum_{i=1}^{n} M_i[j]$ *for every* $j = 1, \ldots, m$.

Proof. After step 4, we can express $a_i[j]$ as:

$$\alpha_i[j] = y_i[j]g + \sum_{i'=i+1}^{n} y'_{i',i}[j]g \tag{23.1}$$

Combining steps 3 and 4, and substituting $\beta[j] = \sum_{i=1}^{n} \beta_i[j]$, we get:

$$\begin{aligned}
M[j] &= \sum_{i=1}^{n} \beta_i[j] - \sum_{i=1}^{n} k_i \alpha_i[j] + \sum_{i=1}^{n} \sum_{i'=1}^{i-1} y'_{i,i'}[j] h_{i'} && (23.2)\\
&= \sum_{i=1}^{n} M_i[j] + \sum_{i=1}^{n} y_i[j] h_i - \sum_{i=1}^{n} y_i[j] h_i - \sum_{i=1}^{n} \sum_{i'=i+1}^{n} y'_{i',i}[j] h_i + \\
&\quad \sum_{i=1}^{n} \sum_{i'=1}^{i-1} y'_{i,i'}[j] h_{i'} && (23.3)\\
&= \sum_{i=1}^{n} M_i[j] - \sum_{i=1}^{n} \sum_{i'=i+1}^{n} y'_{i',i}[j] h_i + \sum_{i=1}^{n} \sum_{i'=i+1}^{n} y'_{i',i}[j] h_i && (23.4)\\
&= \sum_{i=1}^{n} M_i[j] && \square
\end{aligned}$$

Security

To underline the security of the aggregate-shuffle-decrypt protocol, we prove that even with $n - 1$ colluding parties, at least for one honest party its outputs are computationally indistinguishable from randomness. As a result, none of the other parties can learn how the inputs were shuffled. For an argument about the confidentiality of the ciphertexts we refer the reader to the paper by ElGamal [36] . In our proof, we denote the inputs α and β for an honest party as α^{in} and β^{in}, and its outputs as α^{out} and β^{out}. We leave a simulation-based proof as future work.

[36]: ElGamal (1984), 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms'

Theorem 23.2 *For an honest and non-colluding party* $\mathcal{P}_i$*, for each index* $j = 1, \ldots, m$ *it holds that* $\alpha^{out}[j] \overset{c}{\equiv} r$ *and* $\beta^{out}[j] \overset{c}{\equiv} r'$ *where* $r, r' \in_R \mathbb{Z}_q$.

Proof. We express a the outputs of party $\mathcal{P}_i$ in terms of its inputs:

$$\alpha^{\text{out}}[j] \leftarrow \alpha^{\text{in}}[j] + y'_{i,i'}[j]g \tag{23.5}$$
$$\beta^{\text{out}}[j] \leftarrow \beta^{\text{in}}[j] - k_i\alpha_i[j] + \dots \tag{23.6}$$

Since $y'_{i,i'}[j] \in_{\text{R}} \mathbb{Z}_q$ and is only known to this party, $y'_{i,i'}[j]g$ is statistically indistinguishable from randomness. Thereby, $\alpha^{\text{out}}[j] \overset{c}{\equiv} r$. Moreover, since $k_i \in \mathbb{Z}_q$ and is only known to this party, for any other party $k_i\alpha_i$ is computationally indistinguishable from randomness by the DDH assumption. So, $\beta^{\text{out}}[j] \overset{c}{\equiv} r'$. □

Efficiency

We denote the number of ciphertexts by m. In the first step, each party performs m ElGamal encryptions, which takes $O(m\kappa)$ operations. Then, in step 2, the leader sums mn curve points, which takes $mn\kappa$ operations. In step 3, each party performs a permutation, which we deem to be negligible in terms of its runtime. Additionally, assistant $\mathcal{P}_i$ randomizes $m(i-1)$ curve points, which takes at most $O(mn\kappa)$, and partially decrypts, which also takes at most $O(mn\kappa)$ operations. The same goes for the leader in step 4. So, each party performs $O(mn\kappa)$ operations asymptotically. Additionally, each party sends at most $O(mn\lambda)$ bits during steps 2 & 3.

23.2 MPSO-CA protocol

Give example of bitsets Correctness and security is mostly given by the previous proofs and the methods discussed in Chapter 5 To denote the identity we use $\mathcal{O}$, as it denotes the point at infinity of an elliptic curve group. Note however, that the protocol is not restricted to EC-ElGamal.

MPSU-CA protocol size-hiding

1. All parties $\mathcal{P}_i$ for $i = 1, \dots, n$ encode their set X_i to form encoding $\hat{X}_i$.
2. All parties $\mathcal{P}_i$ for $i = 1, \dots, n$ take part in the **aggregate-shuffle-decrypt** protocol with the bins of $\hat{X}_i$, where:

$$M_i[j] = \begin{cases} \mathcal{O} \in \mathbb{G}_T & \text{if } \hat{X}_i[j] = 0 \\ r \in_{\text{R}} \mathbb{G}_T & \text{if } \hat{X}_i[j] = 1 \end{cases}$$

3. The leader $\mathcal{P}_1$ extracts M, counts the points for which it holds that $M[j] \neq \mathcal{O}$ as F, and returns $z \leftarrow \textsc{cardinality}(F)$.

Protocol 23.2: Protocol for securely computing the cardinality of the union between several parties' input sets, without revealing which items are in the union.

Rather than encrypting a randomly generated element, a party can just choose two random elements for alpha and beta. An intersection-cardinality protocol follows similarly, by submitting the identity $\mathcal{O}$ only when $\hat{X}_i = 1$ and counting the number of points which are the identity.

Approximation

To limit the number of bins, one can use Bloom filters rather than bitsets. However, the probability distribution of the possible cardinality estimates for a Bloom filter becomes wide when the number of filled bins is large, as seen in Figure 6.1. As a result, the standard deviation of the approximation increases when the cardinality is large. To alleviate this, we introduce the concept of selectivity, which is inspired by sketches, see Section 8.2.

Instead of encoding every element from a party's set in the Bloom filter, we only insert an element with a specific probability p, which we call the selectivity ratio. Importantly, this is not a coin toss, but rather it is deterministically decided by a public hash function. For example, a simple way to achieve $p = 50\%$ is to only insert element of which the hash starts with a 0 bit. In the case where $p = 100\%$, the Bloom filter functions like a regular Bloom filter.

We need to accommodate for this selectivity in the final estimate. As for the regular Bloom filter, we use the $\mathrm{E}[N]$ as our estimate. We derive the expected value in the same way as we did in Chapter 6.

Theorem 23.3 *The expected number of elements encoded by a Bloom filter with F filled bins and a selectivity ratio p is given by:*

$$\mathrm{E}[N] \approx -\frac{pm}{h} \ln\left(1 - \frac{F}{m}\right)$$

Proof. The probability that a bin is 0 after N inserted elements is $\left(1 - \frac{p}{m}\right)^{hN}$, which we can approximate using:

$$\lim_{m\to\infty} \left(1 - \frac{p}{m}\right)^{kn} = \exp\left(-\frac{hN}{pm}\right) \tag{23.7}$$

What follows, is that the posterior probability of the number of filled bins F given the number of inserted elements N is binomially distributed:

$$\Pr[\mathbf{F} = F \mid \mathbf{N} = N] \sim \mathcal{B}\left(m, \left(1 - \exp\left(-\frac{hN}{pm}\right)\right)\right) \tag{23.8}$$

Using the same derivation as in Chapter 6, it follows that:

$$\frac{F}{m} \approx \left(1 - \exp\left(-\frac{hN}{pm}\right)\right) \tag{23.9}$$

$$N \approx -\frac{pm}{h} \ln\left(1 - \frac{F}{m}\right) \qquad \square$$

Efficiency

The computation of this protocol is based on the aggregate-shuffle-decrypt protocol, since we deem the operations performed in step 1 and 3 to be negligible. As such, each party computes $O(|\hat{X}|n\kappa)$ operations, and sends $O(|\hat{X}|n\lambda)$ bits. In the case of bitsets, $|\hat{X}| = |\mathcal{U}|$, while for Bloom filters, $|\hat{X}| = m$. We can write m as $O(k)$ for a constant approximation quality, since the false positive rate scales with $O(k)$.

24 Threshold set intersections

24.1 Shuffle-decrypt-to-zero . . . 96
24.2 Threshold set intersections . 96
Protocol description 96
Efficiency 97

After Kissner and Song [18] proposed threshold set operations, many works have since followed with increasingly efficient protocols. However, to the best of our knowledge, all works that do not reveal the number of times an element appears in the resulting set require at least three stages of communication, demanding each party to be online. In this chapter we propose a protocol for threshold set operations that requires only two stages of communication. We highlight the special case of count-hiding threshold set intersections, and we extend our earlier shuffle-decrypt protocol to a shuffle-decrypt-to-zero.

24.1 Shuffle-decrypt-to-zero

Similar to a decrypt-to-zero protocol in other threshold cryptosystems, as discussed in Section 11.3, we can extend our shuffle-decrypt protocol from the previous chapter to perform a decrypt-to-identity rather than revealing the true values in the ciphertexts. To do so, in each round of step 3 of the shuffle-decrypt protocol, choose a random number and multiply each $\alpha[j]$ and β with it, for $j = 1, \ldots, m$. We do not give a formal proof, but note that $x\mathcal{O} = \mathcal{O}$ for any $x \in \mathbb{G}_T$, and by ElGamal's well-studied homomorphism, the decrypted result is only the identity $\mathcal{O}$ with overwhelming probability when the ciphertext was indeed the encryption of the identity. The additional operation does not change the asymptotic complexity of the shuffle-decrypt protocol.

24.2 Threshold set intersections

In this section, we propose a protocol for threshold set intersections, but the same technique can be used to construct a threshold set union protocol, among others.

Protocol description

The intuition behind our protocol is to let each party encode their input set as a bitset or a Bloom filter, after which they encrypt each bin separately, where for a 0 they submit an encryption of the identity $\mathcal{O}$, and for a 1 an encryption of the generator g. The leader then sums these ciphertexts to arrive at the total count of each bin. To find the threshold intersection, we wish to identify those bins where the total count surpasses threshold τ, which we do using a subset query.

The subset query is based on the work by Miyaji and Nishida [6]. Suppose we have a ciphertext representing $7g$, then we can test if that is indeed the case by homomorphically subtracting $7g$ and decrypting-to-identity to check if the result is the identity. This is effectively a secure equality

protocol. To transform it into a subset query we perform a secure equality for each number that is greater or equal to the threshold τ, and shuffle the results. We realize this using the aforementioned shuffle-decrypt-to-zero protocol. Rather than a subset query, in some of our previous works[1] we use secure comparison protocols. However, those secure comparisons require an additional stage of interaction, while we can perform our subset query next to decryption as part of the shuffle-decrypt protocol.

1: **Multi-party Private Set Intersection** by Aslı Bay, Zeki Erkin, Mina Alishahi, and Jelle Vos
Practical Multi-party Private Set Intersection Protocols by Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos

Using these primitives, we propose the following multi-party private threshold set intersection protocol:

T-MPSI protocol count-hiding size-hiding

1. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ encode their set X_i to form encoding $\hat{X}_i$.
2. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ encrypt each bin $j = 1, \ldots, |\hat{X}_i|$ of the encoding $\hat{X}_i$ using their own public key h_i and with $y \in_R \mathbb{Z}_q$ and send it to the leader $\mathcal{P}_1$:

$$\langle \alpha_i[j], \beta_i[j] \rangle = \langle y_i[j]g, M_i[j] + y_i[j]h_i \rangle,$$

   where message $M_i[j]$
3. All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ take part in $|\hat{X}|$ **aggregate-shuffle-decrypt-to-zero** protocols at the same time for $j = 1, \ldots, |\hat{X}|$, starting from step 2 using $\alpha[q] = \langle \alpha_1[j], \ldots, \alpha_n[j] \rangle$ and $\beta[q] = \sum_{i=1}^{n} \beta_i[j] - qg$, with $q = \tau, \ldots, n$.
4. The leader $\mathcal{P}_1$ determines the resulting encoding by checking for each bin $j = 1, \ldots, m$ if at least one ciphertext was decrypted to the identity, and setting those bins to 1, while keeping the others at 0. The leader extracts the result by checking which of its set elements are contained in the encoding.

Protocol 24.1: Protocol for multi-party private threshold set intersections, where none of the parties find out how often an element appears in the input sets nor how many elements each party submitted.

It is possible to use the same principle to build an interactive, count-revealing over-threshold scheme, where we do not shuffle the ciphertexts, and only decrypt. Moreover, by changing how the leader creates the ciphertexts after aggregating each party's encryption bins, it is possible to construct a below-threshold set operation protocol. Actually, in that sense it is even possible to define other complex operations, such as to identify elements that appear only an even number of times.

Efficiency

Since the encoding step of the protocol requires no cryptographic operations, we do not consider it in our asymptotic complexities. Step 2 of the protocol requires $O(|\hat{X}_i|n\kappa)$ operations. In the worst case, step 3 of the protocol requires a shuffle-decrypt-to-zero on n ciphertexts per bin. As a result, a party must perform $|\hat{X}|$ shuffle-decrypt-to-zero protocols that each takes $O(n^2\kappa)$ operations. So, the computational complexity of step 3 is $O(|\hat{X}|n^2\kappa)$ for each party. The final step does not require any cryptographic operations, so the total computation complexity is $O(|\hat{X}|n^2\kappa)$.

The worst-case communication complexity follows similarly: $O(|\hat{X}|n^2\lambda)$ bits. In the case of bitsets, $|\hat{X}| = |\mathcal{U}|$, while for Bloom filters, $|\hat{X}| = O(k)$ for threshold set intersections and $|\hat{X}| = O(nk)$ for threshold set unions. Note that threshold set unions incur additional computations.

Privacy-preserving selection of threat intelligence providers

25

25.1 Setup 99
25.2 Results of scenario A 100
25.3 Results of scenario B 100
25.4 Use in practice 101

Cyber criminals reuse the same resources multiple times in an effort to get the most out of their investments. Such a resource can be C2 servers, for example, as we discuss in Chapter 22. As a result, organizations can protect themselves from such criminal activities by keeping a list of the IP addresses associated with such C2 servers, and prevent communication with them. In this case, the IP address is called an indicator of compromise (IoC), and finding them is the goal of cyber threat intelligence (CTI). Since finding IoCs is a time-consuming task, organizations commonly buy them from CTI providers.

In a previous, unpublished paper titled "Compare Before You Buy", Christian Doerr and Zekeriya Erkin found that for organizations to shop around with multiple CTI providers, they want to know how many unique IoCs they are buying, rather than the total number. For example, three CTI providers who are selling 10,000 IoCs may actually only be selling 20,000 unique ones. However, an organization may not find out the actual IoCs, or they would not have to buy them anymore. As such, we can use an MPSU-CA protocol to determine only the cardinality of the resulting union of IoCs.

In this chapter we evaluate the performance of our approximate MPSU-CA protocol on sets of IP addresses in the context of buying cyber threat intelligence. We show that for the earlier example where three CTI providers have 10,000 IoCs each, the total runtime of our protocol takes less than 6 seconds to get an approximation of the union cardinality with a standard deviation of just 250. Additionally, we show how to trade off a longer run time for a more accurate approximation.

25.1 Setup

We evaluate the practical performance of the MPSU-CA protocol described in Chapter 23. In our experiments we perform 20 repetitions and report the means. We measure the time it takes to encrypt, the time it takes to perform the shuffle-decrypt protocol and we keep track of the estimated cardinalities. We executed all experiments on the machine described in Table 20.1. As of yet the implementation runs all parties sequentially on a single core. This is not detrimental to performance, though, since the shuffle-decrypt must be performed sequentially anyways. It does affect the encryption stage.

In our evaluations we analyze two scenarios:

- **Scenario A:** Three parties with $10,000$ elements each whose sets form a union of $20,000$ elements, see Table 25.1.
- **Scenario B:** Five parties with $20,000$ elements each whose sets form a union of $50,000$ elements, see Table 25.2.

For each of the scenarios we evaluate the performance using a lower accuracy Bloom filter with $m = 10,000$ bins, and a higher accuracy Bloom filter with $m = 50,000$ bins. For each combination of parameters we also vary the selectivity ratio p between 100%, 50% and 25%, and adjust the number of bins accordingly. We set the number of hash functions $h = 1$ to make the probability distributions for higher cardinalities as narrow as possible, see Figure 6.1. In all of our experiments, the setup and aggregation stages took at most 100 milliseconds together. For this reason we omit them from the tables.

25.2 Results of scenario A

We present the results of scenario A in Table 25.1. Clearly, in the higher accuracy case with a larger Bloom filter, run time increases but the standard deviation decreases. Moreover, the run time of the shuffle-decrypt stage scales roughly linearly with the number of bins m. As a sign of validation, the estimated mean is indeed centered around 20,000.

	Lower accuracy			**Higher accuracy**		
	$p = 100\%$	$p = 50\%$	$p = 25\%$	$p = 100\%$	$p = 50\%$	$p = 25\%$
Bins m	10,000	5,000	2,500	50,000	25,000	12,500
Encryption [s]	1.4	0.7	0.4	12.2	6.0	3.0
Shuffle-decrypt [s]	4.4	2.3	1.1	21.8	11.0	5.4
Estimate mean	19,986	19,925	20,069	19,995	20,011	20,061
Standard deviation	±251	±386	±418	±62	±136	±285

Table 25.1: Runtime and estimation results over 20 experiments for scenario A, where 3 parties with 10,000 elements each, estimate a union-cardinality of 20,000 elements. The difference between 'lower' and 'higher' accuracy is a higher number of bins in the Bloom filter.

25.3 Results of scenario B

We present the results of scenario A in Table 25.2. Again, we see the trade-off between runtime and accuracy. While runtime for $p = 100\%$ in the higher accuracy case takes more than a minute to compute, we see that the selectivity ratio $p = 25\%$ with $m = 12,500$ brings that time down to less than 20 seconds, while retaining a standard deviation of only 490, which is significantly better than the case where $m = 10,000$ and $p = 100\%$. We think that this is because as the number of filled bins increases, the probability distribution widens, as seen in Figure 6.1. As such, by using a selectivity ratio, we reduce the number of filled bins in the Bloom filter.

	Lower accuracy			**Higher accuracy**		
	$p = 100\%$	$p = 50\%$	$p = 25\%$	$p = 100\%$	$p = 50\%$	$p = 25\%$
Bins m	10,000	5,000	2,500	50,000	25,000	12,500
Encryption [s]	1.5	0.8	0.4	17.5	8.7	4.4
Shuffle-decrypt [s]	12.0	6.1	3.0	60.0	30.4	15.0
Estimate mean	49,539	49,510	50,771	50,066	50,081	50,065
Standard deviation	±1,284	±1,364	±2,381	±165	±307	±490

Table 25.2: Runtime and estimation results over 20 experiments for scenario B, where 5 parties with 20,000 elements each, estimate a union-cardinality of 50,000 elements. The difference between 'lower' and 'higher' accuracy is a higher number of bins in the Bloom filter.

25.4 Use in practice

Since the underlying shuffle-decrypt protocol only requires a party to send ciphertexts encrypted with their own public key, a company only has to perform encryption once for one set of IoCs and one set of Bloom filter parameters, after which the leader can initiate comparisons between any organizations in the future. If the company agrees with the computation, it takes part in the shuffle-decrypt protocol, which only requires one action.

Conclusion

Conclusion & discussion 26

In this thesis we proposed the first non-interactive protocols for multi-party private set and multiset operations, and we decreased the number of communication stages of more complex set operations to a new low of two stages. Table 26.1 gives a summary of all these protocol and their properties. We provide implementations for all non-interactive exact protocols and some approximate protocols. We believe that the protocols evaluated in Chapters 22 & 25 can be used in practice because our experiments verify that they take in the order of seconds to compute on realistic data. In addition, our approximate MPSU-CA allows threat intelligence providers to submit their ciphertexts once, after which every run requires them to only perform one action.

Table 26.1: All of our proposed protocols and their properties. See the respective chapters for a derivation of these protocols' complexities.

Operation		Communication			Computation		Security
Short	Type	Topology	Assistant	#	Leader	Assistant	Collusion
MPSI	Exact	Star	$O(\lvert\mathcal{U}\rvert\lambda)$	1	$O(nk\kappa)$	$O(\lvert\mathcal{U}\rvert\kappa + nk\kappa)$	$n-2$
	Approx.		$O(k\lambda)$		$O(nk\kappa)$	$O(nk\kappa)$	
MPSU	Exact	Star	$O(\mathcal{U}\lvert\lambda)$	1	$O(\lvert\mathcal{U}\rvert n\kappa)$	$O(\lvert\mathcal{U}\rvert n\kappa)$	$n-2$
	Approx.		$O(nk\lambda)$		$O(n^2k\kappa + \lvert\mathcal{U}\rvert)$	$O(n^2k\kappa)$	
MPMI	Exact	Star	$O(M'\lvert\mathcal{U}\rvert\lambda)$	1	$O(W'n\kappa)$	$O(M'\lvert\mathcal{U}\rvert\kappa + W'n\kappa)$	$n-2$
	Approx.		$O(W'\lambda)$		$O(nW'\kappa)$	$O(nW'\kappa)$	
MPMU	Exact	Star	$O(M'\lvert\mathcal{U}\rvert\lambda)$	1	$O(M'\lvert\mathcal{U}\rvert n\kappa)$	$O(M'\lvert\mathcal{U}\rvert n\kappa)$	$n-2$
	Approx.		$O(W'\lambda)$		$O(n^2W'\kappa + M'\lvert\mathcal{U}\rvert)$	$O(n^2W'\kappa)$	
MPMS (two)	Exact	Star	$O(\lvert\mathcal{U}\rvert\lambda)$	1	$O(M'\lvert\mathcal{U}\rvert nk\kappa + \lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert n\kappa)$	$n-2$
	Exact		$O(nG_{M,K}(\lvert\mathcal{U}\rvert))$		$O(nkf(\lvert\mathcal{U}\rvert)\kappa)$	$O(Mnk)$	
	Approx.		$O(nk\lambda)$		$O(M'n^2k\kappa + \lvert\mathcal{U}\rvert)$	$O(n^2k\kappa)$	
MPSU-CA	Exact	Wheel	$O(\lvert\mathcal{U}\rvert n\lambda)$	2	$O(\lvert\mathcal{U}\rvert n\kappa)$	$O(\lvert\mathcal{U}\rvert n\kappa)$	$n-1$
	Approx.		$O(nk\lambda)$		$O(nk\kappa)$	$O(nk\kappa)$	
T-MPSI	Exact	Wheel	$O(\lvert\mathcal{U}\rvert n^2\lambda)$	2	$O(\lvert\mathcal{U}\rvert n^2\lambda)$	$O(\lvert\mathcal{U}\rvert n^2\lambda)$	$n-1$
	Approx.		$O(n^2k\lambda)$		$O(n^2k\kappa)$	$O(n^2k\kappa)$	

26.1 Discussion

In this work we attempted to answer the question: *How can multiple parties collaboratively, but with minimal interactions, compute set and multiset operations —such as set intersections —without revealing their inputs?*. To do so properly, we compared numerous protocols for all different types of set and multiset operations. We used this comparison to identify set encodings that we can aggregate non-interactively and to find the degree of interaction required for the current state of the art. We set out to design protocols that require less interactions than the state of the art.

We select bitsets and Bloom filters, because these can be aggregated to compute intersections and unions using AND and OR operations, re-

spectively. To do so securely, we propose novel non-interactive secure protocols for the aforementioned logical operations.

Apart from these set encodings, many other protocols encode sets and multisets as polynomial roots [18, 51]. However, so far we are not aware of a non-interactive secret sharing-based method to perform arithmetic on encrypted polynomials. There are solutions using homomorphic cryptosystems [7, 18], but these are interactive.

While it is possible to base our secure `AND` and `OR` protocols on groups of integers, we use an elliptic curve group to significantly speed up computation and lower communication. As a drawback, we cannot perform secure aggregation over such a group, because it is infeasible to compute the discrete logarithm. However, in our secure `AND` and `OR` protocols we are only interested in whether the element is the identity or not. Moreover, for the secure aggregation required in our MPMS protocols, we measure run times in the order of seconds using a lookup table.

Non-interactive protocols

In the part about non-interactive protocols we answered the question: *What operations can be performed non-interactively and how can we do so?*

We answered this question extensively by proposing both exact and approximate protocols for each of the primitive set and multiset operations. We designed these protocols so they can be instantiated in both a symmetric or asymmetric secret sharing construction. Due to these constructions, the asymptotic complexity for every party in our protocol necessarily scales linearly with the number of parties n, while complexities for assistants in some other protocols do not scale with n at all.[1] We argue that this is an acceptable overhead, considering that an assisting party only has to perform computations once due to the non-interactivity, rather than actively participate in a protocol.

1: For example, the works by Bay et al.

To verify that our non-interactive protocols are indeed practical, we test their performance on real-looking malware capture data in Chapter 22. While the asymmetric construction can take minutes or hours to compute for certain operations, the symmetric construction requires just seconds. We argue that the asymmetric construction is suitable in cases where the key infrastructure for the symmetric construction would take long to set up. Like the symmetric construction, there are also other works [5] that require every pair of parties to share keys. Note that once the construction has been set up, the parties can take part in all different types of set and multiset operations, without having to set up again.

One limitation of our proposed protocols is that our exact protocols necessarily scale with the size of the universe. Of course, for a small universe this is not limiting, but for large universes such as the IPv4 address space, this becomes infeasible. Our approximate protocols, fortunately, scale with the set size k rather than the size of the universe. Other works [5, 51] do propose exact alternatives that do not scale with the size of the universe. However, they require several interactions.

Minimally-interactive protocols

In the part about interactive protocols we answered the question: *What operations must be performed interactively, and how can we do so with the minimum number of interactions?*

Unfortunately, we have not proven which operations cannot be performed non-interactively. However, we have demonstrated that at least for the techniques that we used for non-interactive protocols, using the same techniques for MPSO-CA and T-MPSO protocols leads to a contradiction. Instead, we propose protocols that require one additional interaction, while hiding the size of each party's set and not leaking the counts of the resulting elements in threshold operations.

Both the MPSA-CA and T-MPSO protocols allow a party to encrypt their sets once in the first steps of the protocol, after which they can be combined with those of any other party without the original party to be involved. Then, finishing the protocol requires only one interaction from the party. This is different to other works, such as the work by Debnath et al. [37], which requires a separate key to be generated for each group of parties, and thereby require the parties to encrypt their sets again using this new key.

26.2 Future work

In the future, we want to perform more tests on the reversible hash functions. In addition, it remains an open problem if it is possible to design a reversible hash function that does not internally rely on calling multiple other hash functions.

Our MPMI and MPMU protocols are actually generalizations of the MPSI and MPSU protocols that we propose, since each party first converts their multiset into a set representation before taking part in the protocol. As such, it stands to reason that there may be more efficient, specialized protocols for performing these multiset operations.

Security-wise, it remains an open problem how to or if it is possible to extend our protocols into the malicious model. Since we use a non-standard shuffling protocol in our interactive protocols, it may not be possible to do so using the current building blocks for maliciously-secure protocols.

Implementation-wise, the proof of concept code is now spread over Python, C++ and Rust. Considering that the Rust code is the fastest so far, we encourage an official implementation in Rust. Specifically, one that is implemented over actual networks, rather than simulating multiple parties in threads on one machine.

Finally, since our protocols require none or minimal interactions, a logical next step would be to examine whether new practical applications can benefit from this. So far, we only evaluate the protocols on real-looking data, but we pose that much can be learned from deploying the protocols in real-life problems, such as private contact tracing.

Adaptations of previous works' complexities

A

In this chapter we present derivations for the adapted complexities in the tables from the part about 'previous works'. Note that whenever we use the number of bins m in the context of Bloom filters, we can replace this by $O(k)$, as discussed in Chapter 6.

A.1 MPSI protocols 106
A.2 T-MPSO protocols 110
A.3 MPSU protocols 111
A.4 MPSO-CA protocols 112
A.5 MPMO protocols 114

A.1 MPSI protocols

Communication complexity general MPC

The communication complexity of a general multi-party computation scheme was given by Kissner and Song [18]. Since then, better general schemes might have been proposed but for the purpose of demonstrating an upper bound we do compare against this general MPC. Because the authors do not explain what the circuit looks like we have only assumed how to rewrite this complexity. Because Kissner and Song [18] mention the total communication complexity rather than the communication per party we divide their complexity by n and include a statistical security parameter λ. This results in $O(nk \operatorname{polylog}(k) \log(|U|))\lambda$. We choose to denote it using a version of the big-oh notation that omits logarithmic terms: $\widetilde{O}(nk\lambda)$.

Communication complexity Freedman, Nissim, and Pinkas [41]

The communication complexity was missing for the multi-party case.

1. Each assistant sends their encrypted polynomial to the leader, which takes $O(k\lambda)$ bits.
2. Each assistant also sends $n-1$ encrypted k-shares to the leader, which takes $O(nk\lambda)$ bits.
3. The leader sends k tuples of $n-1$ encrypted items each, which takes $O(nk\lambda)$ bits.
4. The leader also forwards the encrypted shares, which takes $O(nk\lambda)$ bits.
5. Each assistant sends the XOR of the received items and their generated shares, which takes $O(\lambda)$ bits.

In this protocol the dominating communication is $O(nk\lambda)$ for an assistant and the leader alike.

Computation complexity Freedman, Nissim, and Pinkas [41]

The computation complexity was missing for the multi-party case.

1. Each assistant generates an encrypted polynomial, which takes $O(k\kappa)$.

2. Each assistant also generates $n-1$ encrypted k-shares, which takes $O(nk\kappa)$.
3. The leader evaluates n polynomials (and performs some other operations on them), which takes $O(nk\kappa)$.
4. Each assistant decrypts n ciphertexts, which takes $O(n\kappa)$.

We have ignored some share operations because they are negligible in computation when compared with the public-key operations. This results in a computational cost of $O(nk\kappa)$ for both the leader and an assistant.

Communication complexity Kissner and Song [18]

Kissner and Song [18] already give a computation and communication complexity but we want to know the complexity per party (as opposed to the total complexity).

1. Each party sends their encrypted polynomial to t other parties, which takes $O(tk\lambda)$ bits.
2. Each party sends another encrypted polynomial to one other party, which takes $O(k\lambda)$ bits.
3. The leader sends the final encrypted polynomial to all other parties, which takes $O(nk\lambda)$ bits.
4. Each party participates in a group decryption (for each coefficient) by sending the their decrypted shares to t other parties, which takes $O(tk\lambda)$ bits.

The final communication complexity is $O(nk\lambda)$ for the leader and $O(tk\lambda)$ for an assistant.

Computation complexity Kissner and Song [18]

1. Each party generates an encrypted polynomial, which takes $O(k\kappa)$.
2. Each party homomorphically multiplies $t+1$ polynomials, which takes $O(tk^2\kappa)$.
3. Each assistant adds two encrypted polynomials together, which takes $O(k\kappa)$.
4. Each party participates in a group decryption (for each coefficient), which takes $O(tk\kappa)$.

This leads to a computation complexity of $O(tk^2\kappa)$ for both the leader and an assistant.

Communication complexity Li and Wu [42]

This paper only gives the communication complexity for the full protocol. It is hard to deduce the complexity per party, but we assume that it is $O(n^2k^2\lambda)$ since the whole communication complexity is $O(n^3k^2\lambda)$.

Computation complexity Li and Wu [42]

The paper does not give the computation complexity and the complexity is hard to deduce. We think the dominating operation is the first step of the computation phase. This means the computation complexity per party is approximately $O(nk^2\lambda)$, since a party multiplies n secret-shared polynomials with k coefficients of length λ.

Communication complexity Kerschbaum [43]

1. Each assistant sends their encrypted Bloom filter to the leader, which takes $O(m\lambda)$ bits.
2. The leader sends several ciphertexts for all m bins per party to t assistants to perform a decrypt-to-zero, which takes $O(ntm\lambda)$ bits.
3. The assistants each send $O(nm\lambda)$ bits back.
4. The leader sends the aggregated ciphertexts to t assistants to decrypt, which takes $O(tm\lambda)$ bits.
5. The assistants each send $O(m\lambda)$ bits back.

This results in $O(nm\lambda)$ communicated bits for an assistant and $O(ntm\lambda)$ for the leader.

Computation complexity Kerschbaum [43]

1. Each assistant generates an encrypted Bloom filter, which takes $O(kh + m\kappa)$.
2. Some parties perform several decrypts-to-zero per party per bin, which takes $O(nm\kappa)$.
3. The leader aggregates the results using homomorphic addition, which takes $O(nm\kappa)$.
4. Some parties perform a decryption per aggregated bin, which takes $O(m\kappa)$.
5. The leader extracts the elements, which takes $O(kh)$.

This results in a computational complexity of $O(nm\kappa)$ for all parties, assuming that this complexity dominates $O(kh)$.

Complexities Cheon, Jarecki, and Seo [25]

For both the communication and computation complexity we have added the security parameters. Furthermore, the title of the work calls for a quasi-linear complexity, which can be misleading as the computation and communication is only linear in k and not in the number of parties.

Communication complexity Cheon, Jarecki, and Seo [25]

1. Each assistant sends their encrypted Bloom Filter to the leader, which takes $O(m\lambda)$ bits.
2. The leader sends the encrypted aggregated Bloom Filter to t parties to jointly decrypt, which takes $O(tm\lambda)$ bits.
3. The assistants reply with $O(m\lambda)$ bits.

This results in a communicational complexity of $O(m\lambda$ for an assistant and $O(tm\lambda)$ for the leader.

Computation complexity Miyaji and Nishida [6]

1. Each party generates an encrypted Bloom filter, which takes $O(k + m\kappa)$.
2. The leader aggregates the results using homomorphic addition, which takes $O(nm\kappa)$.
3. Some parties perform a decryption per aggregated bin, which takes $O(m\kappa)$.
4. The leader extracts the elements, which takes $O(kh)$.

This results in a computation complexity of $O(nm\kappa)$ for the leader and $O(m\kappa)$ for an assistant, assuming that this complexity dominates $O(kh)$.

Computation complexity Hazay and Venkitasubramaniam [44]

1. Each assistant encodes their set as encrypted polynomial coefficients, which takes $O(k\kappa)$.
2. The leader evaluates the $n-1$ encrypted polynomials with its k elements, which takes $O(nk^2\kappa)$.
3. The leader sums up the $n-1$ ciphertexts per element, which takes $O(nk\kappa)$.
4. Several assistants help in the decrypt-to-zero of k ciphertexts, which takes $O(k\kappa)$.
5. The leader combines the resulting shares to compute the final intersection, which takes $O(k\kappa)$.

So the computation complexity for the leader is $O(nk^2\kappa)$ and for an assistant is $O(k\kappa)$. At the cost of bandwidth the authors also propose a computation optimization which seems to take the leader only $O(nk \log_2 k\kappa)$.

Communication complexity Inbar, Omri, and Pinkas [45]

1. Each party performs an OT interaction with each other party to share an XOR-secret share, receiving λ bits for each bin in the Bloom Filter, which takes approximately $O(nm\lambda)$ bits.
2. Each assistant sends its share of the final aggregated Garbled Bloom Filter to the leader, which takes $O(m\lambda)$ bits.

This results in a communication complexity of $O(nm\lambda)$ for assistants and the leader alike. The original paper reports a complexity of $O(nhk\lambda)$, where we think $O(hk)$ might have been a substitution for $O(m)$.

Computation complexity Inbar, Omri, and Pinkas [45]

1. Each party builds a Bloom Filter and a t-shared Garbled Bloom Filter, which takes $O(nm\lambda)$.
2. Each party performs an OT interaction with each other party for every bin in the Bloom Filter, which takes approximately $O(nm\lambda)$.
3. Each party XORs their received secret shares from the OT interaction, which takes $O(nm\lambda)$.
4. The leader XORs their received secret shares, which takes $O(nm\lambda)$.

This results in a computation complexity that is also $O(nm\lambda)$ for every party.

Communication complexity of Abadi, Terzis, and Dong [40]

The querying party sends a total of $O(nk)$ messages of λ bits, and the cloud server sends $O(k)$ messages of λ bits, totalling $O(nk)$ transmitted bits for the leader. The assistants each send $O(k)$ times λ bits, which results in $O(k\lambda)$ bits.

Computation complexity of Abadi, Terzis, and Dong [40]

Since the paper by Abadi, Terzis, and Dong [40] fixes the capacity of each bin, the querying party's computation takes $O(nk)$ modular additions in a field of size λ, so $O(nk\lambda)$ in total. The cloud's computation cost has the same asymptotic complexity, so combining these results in a complexity of $O(nk\lambda)$ for the leader. An assistant only has to compute $O(k)$ modular operations in a field of size λ, so the complexity is $O(k\lambda)$.

Communication complexity Debnath et al. [37]

1. Each assistant sends an encrypted Bloom filter, which takes $O(m\lambda)$ bits.
2. The leader sends all assistants at most k aggregated bins, which takes $O(k\lambda)$ bits.
3. Each assistant partially decrypts at most k aggregated bins, which takes $O(k\lambda)$ bits.

This results in a communication complexity of $O(nk\lambda)$ for the leader and $O(k\lambda)$ for an assistant.

Computation complexity of Debnath et al. [37]

1. Each assistant generates an encrypted Bloom filter, which takes $O(k + m\kappa)$.
2. The leader aggregates the results using homomorphic addition for the bins corresponding to its set elements, which takes $O(nkh\kappa)$.
3. Each assistant partially decrypts at most k aggregated bins, which takes $O(k\kappa)$.

As a result, the leader performs $O(nkh\kappa)$ operations, while the assistants perform $(k\kappa)$ operations.

A.2 T-MPSO protocols

Communication complexity general MPC

This is the same as in Section A.3: $\widetilde{O}(nk\lambda)$.

Complexities Kissner and Song [18]

The threshold and over-threshold protocols are very similar to the MPSI-CA protocol of Kissner and Song [18] as described in Section A.4. So, we believe the computation complexity is $O(ntk\kappa + k^2\kappa)$ for each party, while the communication complexity is $O(n^2k\lambda)$ for each party.

Communication complexity of Frikken [51]

1. Each party sends their encrypted polynomial to the other parties, which takes $O(nk\lambda)$ bits.
2. Each party broadcasts the evaluated polynomial for each element in their set, which takes $O(nk\lambda)$ bits.
3. Each party broadcasts a randomized version of at most nk tuples, which takes $O(nk\lambda)$ bits.
4. Each party takes part in a joint decryption with scales linearly with n and k.
5. The parties take part in the MPSU protocol, which takes $O(nk\lambda)$ bits of communication per party.

The final communication complexity is $O(nk\lambda)$ for each party.

Computation complexity of Frikken [51]

Since the protocol relies on their MPSU protocol, overall computation complexity is $O(nk^2\kappa)$ for each party, see Section A.3.

Communication complexity Miyaji and Nishida [6]

1. Each assistant sends an encrypted Bloom filter, which takes $O(m\lambda)$ bits.
2. The leader sends nm subtracted aggregated bins to each party, which takes $O(n^2m\lambda)$ bits.
3. Some assistants take part in decryption and send back $O(nm\lambda)$ bits.

The leader sends out at most $O(n^2m\lambda)$ bits, while an assistant sends out at most $O(nm\lambda)$ bits.

Computation complexity Miyaji and Nishida [6]

1. Each party generates an encrypted Bloom filter, which takes $O(k + m\kappa)$.
2. The leader aggregates the results using homomorphic addition, which takes $O(nm\kappa)$.
3. The leader creates at most nm ciphertexts by subtracting and re-randomizing, which takes $O(nm\kappa)$.
4. Some parties perform a decryption per subtracted aggregated bin, which takes $O(nm\kappa)$.
5. The leader extracts the elements, which takes $O(kh)$.

This results in a computation complexity of $O(nm\kappa)$ for the leader and an assistant alike, assuming that this complexity dominates $O(kh)$.

A.3 MPSU protocols

Communication complexity general MPC

We use the general MPC as reported by Kissner and Song [18] for threshold set unions, which are a generalization of set unions, where we add a sta-

tistical security parameter λ. This results in $O(nk \operatorname{polylog}(nk) \log(|U|))\lambda$ for each party. We choose to denote it using a version of the big-oh notation that omits logarithmic terms: $\widetilde{O}(nk\lambda)$.

Communication complexity of Frikken [51]

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 $O(nk\lambda)$ 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.

Computation complexity of Frikken [51]

1. Each party encodes their set as encrypted polynomial coefficients, which takes $O(k\kappa)$.
2. Each party $\mathcal{P}_i$ homomorphically multiplies i encrypted polynomials together, which takes at most $O(nk^2\kappa)$.
3. Each party $\mathcal{P}_i$ encrypts $2k$ values and homomorphically multiplies $i-1$ encrypted polynomials together, which takes at most $O(nk^2\kappa)$.
4. Each party $\mathcal{P}_i$ homomorphically multiplies i ciphertexts together, which takes at most $O(nk\kappa)$.
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 computation complexity is $O(nk^2\kappa)$ for each party.

Complexities of Seo, Cheon, and Katz [19]

In this work, the parties choose a value p that forms the modulus of the secret shares. We replace p with the statistical security parameter λ. Additionally, we add this security parameters to each term in the complexities. We omit the logarithmic term $n^2k^2 \log \lambda$ as it is dominated by the term $n^4k^2\lambda$.

A.4 MPSO-CA protocols

Communication complexity general MPC

This is the same as in Section A.1: $\widetilde{O}(nk\lambda)$.

Communication complexity Vaidya and Clifton [21]

1. Each party sends k hashed elements to their neighbor, which takes $O(k\lambda)$ bits.
2. Each party sends a set to almost all other parties, which takes $O(nk\lambda)$ bits.
3. Each party sends at most k hashed elements to almost all other parties, which takes $O(nk\lambda)$ bits.

This results in a communication complexity of $O(nk\lambda)$ for every single party.

Computation complexity Vaidya and Clifton [21]

The encryptions outweigh in the first stage outweigh the simple comparison operations in the remainder of the protocol. Each party hashes every element of all parties. This results in a computation complexity of at most $O(nk\kappa)$ for every party.

Communication complexity Kissner and Song [18]

Kissner and Song [18] already give a computation and communication complexity but we want to know the complexity per party (as opposed to the total complexity).

1. Each party sends their encrypted polynomial to t other parties, which takes $O(tk\lambda)$ bits.
2. Each party sends another encrypted polynomial to one other party, which takes $O(k\lambda)$ bits.
3. The leader sends the final encrypted polynomial to all other parties, which takes $O(nk\lambda)$ bits.
4. Each party takes part in the shuffle protocol, which takes $O(n^2k\lambda)$ bits.
5. Each party participates in a group decryption by sending the their decrypted shares to t other parties, which takes $O(ntk\lambda)$ bits.

The final communication complexity is $O(n^2k\lambda)$ for each party.

Computation complexity Kissner and Song [18]

1. Each party generates an encrypted polynomial, which takes $O(k\kappa)$.
2. Each party homomorphically multiplies $t+1$ polynomials, which takes $O(tk^2\kappa)$.
3. Each assistant adds two encrypted polynomials together, which takes $O(k\kappa)$.
4. Each party evaluates and randomizes k elements in the encrypted polynomial, which takes $O(k^2\kappa)$.
5. Each party takes part in the shuffle protocol, which we believe does not require expensive cryptographic operations.
6. Each party participates in a group decryption of all nk elements, which takes $O(ntk\kappa)$.

This leads to a computation complexity of $O(ntk\kappa + k^2\kappa)$ for both the leader and an assistant.

Complexities of Burkhart et al. [53]

The authors mention that the protocol requires $(n-1)k$ multiplications. The secure multiplication protocol by Gennaro, Rabin, and Rabin [74], requires each party to perform $O(tk\lambda)$ operations and send $O(tk\lambda)$ bits. As such, the computation complexity of the protocol would be $O(ntk^2\lambda)$ operations, and the communication complexity would be $O(ntk^2\lambda)$ bits.

Communication complexity Debnath et al. [37]

1. Each assistant sends an encrypted Bloom filter, which takes $O(m\lambda)$ bits.
2. The leader sends the first assistant at most k aggregated bins, which takes $O(k\lambda)$ bits.
3. Each assistant forwards at most k aggregated bins, which takes $O(k\lambda)$ bits.
4. The leader sends all assistants at most k aggregated bins, which takes $O(nk\lambda)$ bits.
5. The assistants each send back at most $O(k\lambda)$ bits.

This results in a communication complexity of $O(nk\lambda)$ for the leader and $O(k\lambda)$ for an assistant.

Computation complexity of Debnath et al. [37]

1. Each assistant generates an encrypted Bloom filter, which takes $O(k + m\kappa)$.
2. The leader aggregates the results using homomorphic addition for the bins corresponding to its set elements, which takes $O(nkh\kappa)$.
3. Each party shuffles the ciphertexts and randomizes them, which takes $O(k\kappa)$.
4. Each party perform a decryption per ciphertext, which takes $O(k\kappa)$.
5. The leader extracts the elements, which takes $O(kh)$.

As a result, the leader performs $O(nkh\kappa)$ operations, while the assistants perform $(k\kappa)$ operations.

A.5 MPMO protocols

Computation complexity of Shishido and Miyaji [39]

1. Each party encodes their set as two encrypted Bloom filters, which takes $O(m\kappa)$.
2. The leader (or outsourced partner) homomorphically adds the Bloom filters together, which takes $O(nm\kappa)$.
3. Each assistant helps in the decryption of the Bloom filters, which takes $O(m\kappa)$.
4. The leader combines the resulting shares to finish the decryptions, which takes $O(m\kappa)$.
5. The leader attempts to extract each element in the universe, which takes $O(|\mathcal{U}|)$.

Assuming that the domain size $|\mathcal{U}|$ is limited, the leader's computation complexity is $O(nm\kappa)$. The assistants' complexity is $O(m\kappa)$.

Communication complexity of Shishido and Miyaji [39]

1. Each assistant sends their Bloom filters to the leader (or outsourced partner), which requires $O(m\lambda)$ bits.
2. The leader (or outsourced partner) sends the aggregated ciphertexts to the assistants, which takes $O(nm\lambda)$ bits.
3. Each assistant sends the decryption shares back, which takes $O(m\lambda)$ bits.

This results in a communication complexity of $O(m\lambda)$ bits for an assistant and $O(nm\lambda)$ bits for the leader.

Complexities of Hong et al. [7]

The per-party communication of this paper is the same as for Kissner and Song [18] although there is a constant time improvement: It is $O(nk\lambda)$ bits. The asymptotic computation cost is dominated by the encryption according to the authors, which takes requires each party to perform two exponentiations in a special field. According to the authors it takes approximately $O(n^{1.6}k^{1.6}\kappa)$ per party after adding the computational security parameter κ, since $\log_2(3) \approx 1.6$.

Bibliography

[1] Yehuda Lindell. 'Secure multiparty computation'. In: *Commun. ACM* 64.1 (2021), pp. 86–96. DOI: 10.1145/3387108 (cited on page 1).

[2] Thai Duong, Duong Hieu Phan, and Ni Trieu. 'Catalic: Delegated PSI Cardinality with Applications to Contact Tracing'. In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12493. Lecture Notes in Computer Science. Springer, 2020, pp. 870–899. DOI: 10.1007/978-3-030-64840-4_29 (cited on pages 1, 2).

[3] Brian D. Ondov et al. 'Mash: fast genome and metagenome distance estimation using MinHash'. In: *Genome Biology* 17.1 (June 2016), p. 132. DOI: 10.1186/s13059-016-0997-x (cited on page 1).

[4] Nick Angelou et al. 'Asymmetric Private Set Intersection with Applications to Contact Tracing and Private Vertical Federated Machine Learning'. In: *CoRR* abs/2011.09350 (2020) (cited on page 2).

[5] 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 M. Thuraisingham et al. ACM, 2017, pp. 1257–1272. DOI: 10.1145/3133956.3134065 (cited on pages 3, 29, 34, 42, 46, 47, 72, 104).

[6] 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 (cited on pages 4, 24, 31, 42, 45, 48–50, 60, 96, 109, 111).

[7] Jeongdae Hong et al. 'Constant-Round Privacy Preserving Multiset Union'. In: *Korean Mathematical Society* 50.6 (Nov. 30, 2013), pp. 1799–1816. DOI: 10.4134/BKMS.2013.50.6.1799. (Visited on 02/23/2021) (cited on pages 4, 13, 40, 57, 58, 104, 115).

[8] Meishan Huang and Bogang Lin. 'Privacy-preserving multi-set operations'. In: *2012 IEEE 14th International Conference on Communication Technology*. 2012, pp. 713–719. DOI: 10.1109/ICCT.2012.6511298 (cited on page 13).

[9] 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 (cited on pages 13, 58).

[10] 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 (cited on pages 16, 68).

[11] Andrea Cerulli, Emiliano De Cristofaro, and Claudio Soriente. 'Nothing Refreshes Like a RePSI: Reactive Private Set Intersection'. In: *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*. Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. Lecture Notes in Computer Science. Springer, 2018, pp. 280–300. DOI: 10.1007/978-3-319-93387-0_15 (cited on page 17).

[12] Tatiana Bradley, Sky Faber, and Gene Tsudik. 'Bounded Size-Hiding Private Set Intersection'. In: *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings*. Ed. by Vassilis Zikas and Roberto De Prisco. Vol. 9841. Lecture Notes in Computer Science. Springer, 2016, pp. 449–467. DOI: 10.1007/978-3-319-44618-9_24 (cited on page 17).

[13] 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 (cited on page 23).

[14] *MurmurHash3*. URL: https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp (cited on pages 24, 73).

[15] Yann Collet. *xxHash*. URL: https://cyan4973.github.io/xxHash/ (cited on pages 24, 73).

[16] 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 (cited on pages 24, 55).

[17] 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 (cited on page 24).

[18] 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 (cited on pages 27, 42, 43, 48, 53, 57, 60, 96, 104, 106, 107, 110, 111, 113, 115).

[19] 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 (cited on pages 28, 51, 52, 112).

[20] David G. Cantor and Hans Zassenhaus. 'A New Algorithm for Factoring Polynomials Over Finite Fields'. In: *Mathematics of Computation* 36.154 (1981), pp. 587–592 (cited on pages 28, 57).

[21] Jaideep Vaidya and Chris Clifton. 'Secure set intersection cardinality with application to association rule mining'. In: *J. Comput. Secur.* 13.4 (2005), pp. 593–622 (cited on pages 29, 40, 53, 55, 113).

[22] 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 (cited on pages 29, 46).

[23] Changyu Dong and Grigorios Loukides. 'Approximating Private Set Union/Intersection Cardinality With Logarithmic Complexity'. In: *IEEE Trans. Inf. Forensics Secur.* 12.11 (2017), pp. 2792–2806. DOI: 10.1109/TIFS.2017.2721360 (cited on pages 29, 55).

[24] Jianqiu Ji et al. 'Min-Max Hash for Jaccard Similarity'. In: *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*. Ed. by Hui Xiong et al. IEEE Computer Society, 2013, pp. 301–309. DOI: 10.1109/ICDM.2013.119 (cited on page 29).

[25] 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 (cited on pages 31, 42, 43, 108).

[26] Elaine Barker. *Recommendation for Key Management: Part 1 - General*. en. May 2020. DOI: https://doi.org/10.6028/NIST.SP.800-57pt1r5 (cited on page 31).

[27] Daniel J. Bernstein and Tanja Lange. *Genus-1 curves over large-characteristic fields*. 2021. URL: https://hyperelliptic.org/EFD/g1p/index.html (cited on page 32).

[28] 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 (cited on page 32).

[29] 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 (cited on page 32).

[30] Henry de Valence, Isis Lovecruft, and Tony Arcieri. *The Ristretto Group*. URL: https://ristretto.group/why_ristretto.html (cited on page 32).

[31] Ran Canetti, Shai Halevi, and Jonathan Katz. 'A Forward-Secure Public-Key Encryption Scheme'. In: *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Springer, 2003, pp. 255–271. DOI: 10.1007/3-540-39200-9_16 (cited on page 33).

[32] Paulo S. L. M. Barreto and Michael Naehrig. 'Pairing-Friendly Elliptic Curves of Prime Order'. In: *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*. Ed. by Bart Preneel and Stafford E. Tavares. Vol. 3897. Lecture Notes in Computer Science. Springer, 2005, pp. 319–331. DOI: 10.1007/11693383_22 (cited on page 33).

[33] Pierre-Alain Fouque and Mehdi Tibouchi. 'Indifferentiable Hashing to Barreto-Naehrig Curves'. In: *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*. Ed. by Alejandro Hevia and Gregory Neven. Vol. 7533. Lecture Notes in Computer Science. Springer, 2012, pp. 1–17. DOI: 10.1007/978-3-642-33481-8_1 (cited on page 33).

[34] Adi Shamir. 'How to Share a Secret'. In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176 (cited on page 35).

[35] Jonathan Katz and Moti Yung. 'Threshold Cryptosystems Based on Factoring'. In: *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, 2002, pp. 192–205. DOI: 10.1007/3-540-36178-2_12 (cited on pages 35, 37, 45).

[36] 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 (cited on pages 36, 93).

[37] 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 (cited on pages 37, 42, 45, 53, 54, 56, 91, 105, 110, 114).

[38] 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 (cited on page 38).

[39] 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 (cited on pages 38, 57, 114, 115).

[40] Aydin Abadi, Sotirios Terzis, and Changyu Dong. 'Feather: Lightweight Multi-party Updatable Delegated Private Set Intersection'. In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 407 (cited on pages 40, 42, 72, 110).

[41] 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 (cited on pages 42, 43, 106).

[42] 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 (cited on pages 42, 43, 107, 108).

[43] 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 (cited on pages 42, 45, 108).

[44] 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 (cited on pages 42, 43, 109).

[45] 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 (cited on pages 42, 46, 109).

[46] 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 (cited on pages 44, 49).

[47] Tomas Sander, Adam L. Young, and Moti Yung. 'Non-Interactive CryptoComputing For NC^1'. In: *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. IEEE Computer Society, 1999, pp. 554–567. DOI: 10.1109/SFFCS.1999.814630 (cited on page 45).

[48] Yongjun Zhao and Sherman S. M. Chow. 'Can You Find The One for Me?' In: *Proceedings of the 2018 Workshop on Privacy in the Electronic Society, WPES@CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, and Aaron Johnson. ACM, 2018, pp. 54–65. DOI: 10.1145/3267323.3268965 (cited on page 48).

[49] 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 (cited on page 48).

[50] Saikrishna Badrinarayanan et al. 'Multi-party Threshold Private Set Intersection with Sublinear Communication'. In: *Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II*. Ed. by Juan A. Garay. Vol. 12711. Lecture Notes in Computer Science. Springer, 2021, pp. 349–379. DOI: 10.1007/978-3-030-75248-4_13 (cited on page 48).

[51] 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 (cited on pages 48, 49, 51–53, 104, 111, 112).

[52] Rasoul Akhavan Mahdavi et al. 'Practical Over-Threshold Multi-Party Private Set Intersection'. In: *ACSAC '20: Annual Computer Security Applications Conference, Virtual Event / Austin, TX, USA, 7-11 December, 2020*. ACM, 2020, pp. 772–783. DOI: 10.1145/3427228.3427267 (cited on page 50).

[53] 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 (cited on pages 53, 54, 114).

[54] Jeongdae Hong et al. 'Privacy Preserving Multiset Union with ElGamal Encryption'. In: *IACR Cryptol. ePrint Arch.* 2008 (2008), p. 523 (cited on page 57).

[55] Bharath K. Samanthula and Wei Jiang. 'Secure Multiset Intersection Cardinality and its Application to Jaccard Coefficient'. In: *IEEE Trans. Dependable Secur. Comput.* 13.5 (2016), pp. 591–604. DOI: 10.1109/TDSC.2015.2415482 (cited on page 58).

[56] Harmanjeet Kaur, Neeraj Kumar, and Joel J. P. C. Rodrigues. 'An Efficient Privacy Preserving Computation of Multiset Intersection Cardinality'. In: *2019 IEEE Global Communications Conference, GLOBECOM 2019, Waikoloa, HI, USA, December 9-13, 2019*. IEEE, 2019, pp. 1–5. DOI: 10.1109/GLOBECOM38437.2019.9013167 (cited on page 58).

[57] Zekeriya Erkin and Gene Tsudik. 'Private Computation of Spatial and Temporal Power Consumption with Smart Meters'. In: *Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings*. Ed. by Feng Bao, Pierangela Samarati, and Jianying Zhou. Vol. 7341. Lecture Notes in Computer Science. Springer, 2012, pp. 561–577. DOI: 10.1007/978-3-642-31284-7_33 (cited on page 61).

[58] Morris Dworkin. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. en. Aug. 2015. DOI: https://doi.org/10.6028/NIST.FIPS.202 (cited on page 62).

[59] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. 'Privacy-Friendly Aggregation for the Smart-Grid'. In: *Privacy Enhancing Technologies - 11th International Symposium, PETS 2011, Waterloo, ON, Canada, July 27-29, 2011. Proceedings*. Ed. by Simone Fischer-Hübner and Nicholas Hopper. Vol. 6794. Lecture Notes in Computer Science. Springer, 2011, pp. 175–191. DOI: 10.1007/978-3-642-22263-4_10 (cited on pages 62, 64, 78).

[60] Constantinos Patsakis, Michael Clear, and Paul Laird. 'Private Aggregation with Custom Collusion Tolerance'. In: *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*. Ed. by Dongdai Lin, Moti Yung, and Jianying Zhou. Vol. 8957. Lecture Notes in Computer Science. Springer, 2014, pp. 72–89. DOI: 10.1007/978-3-319-16745-9_5 (cited on page 62).

[61] William G. Cochran. 'The χ^2 Test of Goodness of Fit'. In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 315–345. DOI: 10.1214/aoms/1177729380 (cited on page 73).

[62] Martin Koster. 'Multi-Functional Privacy-Preserving Data Aggregation'. MA thesis. Delft University of Technology, 2021 (cited on page 79).

[63] J. Barkley Rosser and Lowell Schoenfeld. 'Approximate formulas for some functions of prime numbers'. In: *Illinois Journal of Mathematics* 6.1 (1962), pp. 64–94. DOI: 10.1215/ijm/1255631807 (cited on page 80).

[64] Melissa E. O'Neill. 'The Genuine Sieve of Eratosthenes'. In: *J. Funct. Program.* 19.1 (2009), pp. 95–106. DOI: 10.1017/S0956796808007004 (cited on page 80).

[65] H. W. Lenstra. 'Factoring Integers with Elliptic Curves'. In: *Annals of Mathematics* 126.3 (1987), pp. 649–673 (cited on page 80).

[66] Hans Riesel. *Prime numbers and computer methods for factorization, Second Edition*. Vol. 126. Progress in mathematics. 1994 (cited on page 82).

[67] Jean-Guillaume Dumas. 'On Newton-Raphson Iteration for Multiplicative Inverses Modulo Prime Powers'. In: *IEEE Trans. Computers* 63.8 (2014), pp. 2106–2109. DOI: 10.1109/TC.2013.94 (cited on page 82).

[68] Makoto Matsumoto et al. 'CRYPTOGRAPHIC MERSENNE TWISTER AND FUBUKI STREAM/BLOCK CIPHER'. In: *IACR Cryptol. ePrint Arch.* 2005 (2005), p. 165 (cited on page 84).

[69] Luca Caviglione et al. 'Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection'. In: *IEEE Access* 9 (2021), pp. 5371–5396. DOI: 10.1109/ACCESS.2020.3048319 (cited on page 86).

[70] Shahriar Badsha, Iman Vakilinia, and Shamik Sengupta. 'Privacy Preserving Cyber Threat Information Sharing and Learning for Cyber Defense'. In: *IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019, Las Vegas, NV, USA, January 7-9, 2019*. IEEE, 2019, pp. 708–714. DOI: 10.1109/CCWC.2019.8666477 (cited on page 86).

[71] Sarah Brown, Joep Gommers, and Oscar Serrano. 'From Cyber Security Information Sharing to Threat Management'. In: *Proceedings of the 2nd ACM Workshop on Information Sharing and Collaborative Security*. WISCS '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 43–49. DOI: 10.1145/2808128.2808133 (cited on page 86).

[72] Stratosphere. *Stratosphere Laboratory Datasets*. Retrieved February 17, 2021, from https://www.stratosphereips.org/datasets-overview. 2015 (cited on page 86).

[73] Jiongze Chen et al. 'Performance Evaluation of a Queue Fed by a Poisson Lomax Burst Process'. In: *IEEE Communications Letters* 19.3 (Mar. 2015), pp. 367–370. DOI: 10.1109/LCOMM.2014.2385083. (Visited on 02/25/2021) (cited on page 87).

[74] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. 'Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography'. In: *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*. Ed. by Brian A. Coan and Yehuda Afek. ACM, 1998, pp. 101–111. DOI: 10.1145/277697.277716 (cited on page 114).