An Efficient Non-Interactive OT-MP-PSI Protocol for Multi Institution Attack Detection

by

Onur Eren Arpaci

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2024

© Onur Eren Arpaci 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The Over Threshold Multiparty Private Set Intersection (OT-MP-PSI) problem is a cryptographic problem where N participants aim to identify elements that appear in at least t sets without revealing information about elements that do not meet this threshold. This protocol is particularly relevant for detecting multi-institution attacks, which require private and efficient information sharing. In order to detect the attackers, the institutions want to identify the common external IPs among other institutions without leaking the rest of the connection data. Existing solutions either require numerous communication rounds or have impractical computational complexities for large datasets. We introduce a non-interactive OT-MP-PSI protocol optimized by a novel hashing scheme with a computational complexity of $\mathcal{O}(t^2 M \binom{N}{t})$ as compared to the $\mathcal{O}(M(N \log M/t)^{2t})$ computational complexity of previous existing solutions.

Experimental benchmarks using real-world network data demonstrate the protocol's practicality, showing significantly faster reconstruction times compared to existing methods and effective scalability with the number of participants and set sizes.

Our protocol also efficiently solves the two-party private set intersection ($N = t = 2$) and the multi-party private set intersection ($N > 2, t = N$), offering a versatile solution for various applications.

Acknowledgments

First and foremost, I would like to thank my supervisors, Florian Kerschbaum and Raouf Boutaba who made this thesis possible. I learned and grew so much under your guidance, and I am grateful for your support, diligence, and encouragement. Your role was not just significant, but crucial in the completion of this thesis. You showed me what it means to be a great professor.

I thank all my friends who made me the happy person I am today. I especially thank Atakan for for being the best friend anyone could ask for. I also want to express my gratitude to all the CrySP lab members who not only guided me in research but also made the process fun and enjoyable for me, especially Rassoul and Nils, who answered every little question I had and shared their valuable experience with me.

I also wish to thank Beryl for her love and support throughout my degree. Your presence in my life has been invaluable. Thank you for being by my side on bad and good days, for looking after me when I was sick, and for all the wonderful memories we shared. You gave me the emotional and mental support I needed to complete this thesis.

Finally, I'm deeply indebted to my family for the immense love and support that I received. I would like to thank my parents, Sevim and İsmail, for always believing in me and being proud of me; my brother Arda, for listening to my boring talks with great enthusiasm; and my uncle Yılmaz, for nudging me into computer science when I was a little kid.

Dedication

To my parents, who love me unconditionally and provided me with the tools and means to pursue my goals even when it was difficult.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgments iv

Dedication v

List of Figures ix

List of Tables x

1 Introduction 1

2 Preliminaries 4

2.1 Security Model 4

2.2 Shamir's Secret Sharing 5

3 Protocol 6

3.1 Using the same polynomial 6

3.2 Reconstruction Complexity 7

3.3 Identifying the Successful Reconstructions 9

3.4 Protocol Description 10

4 The New Hashing Scheme **12**
4.1 Reversing the ordering . . . 14
4.2 Utilizing the empty indexes . . . 15

5 Evaluation **17**
5.1 Security Analysis . . . 17
5.2 Complexity Analysis . . . 18
5.2.1 Computational Complexity . . . 18
5.2.2 Communication Complexity . . . 19
5.3 Correctness Evaluation . . . 19
5.4 Performance Evaluation . . . 19
5.4.1 Setup . . . 19
5.4.2 Reconstruction . . . 20
5.4.3 Share Generation . . . 23

6 Related Work **28**
6.1 Previous Solutions to OT-MP-PSI . . . 28
6.1.1 Kissner and Song . . . 29
6.1.2 Mahdavi et al. . . . 29
6.1.3 Stanek et al. . . . 30
6.2 Related Work Regarding the Hashing Scheme . . . 30
6.2.1 Optimized Binning . . . 30
6.2.2 Cuckoo Hashing with Simple Hashing . . . 31
6.2.3 2D Cuckoo Hashing . . . 31
6.2.4 Phasing: Permutation-based Hashing . . . 33
6.3 Similar Problems . . . 34
6.3.1 Threshold Private Set Intersection (TPSI) . . . 34
6.3.2 Private Set Union (PSU) . . . 35

7 Conclusion **36**

References **38**

List of Figures

1.1 The OT-MP-PSI protocol visualization ($t = 3$) 2

3.1 The binning technique used in [16], light gray numbers represent the dummy shares, P is the polynomial function, and h is the hash function used for binning. 8

3.2 The new hashing scheme. Every participant creates multiple arrays, P is the polynomial function, h is the hash function used for mapping, H is the hash function used for pseudo-random ordering, and light gray numbers represent the dummy shares. 9

5.1 Number of missed Intersection Elements in 10^7 trials ($M = 200, t = 4$) . . . 20

5.2 Reconstruction time comparison with Mahdavi et al. [16] ($N = 10$) 21

5.3 Reconstruction time on CANARIE data [4] 22

5.4 Reconstruction time vs number of participants ($M = 10000$) 23

5.5 Reconstruction time vs threshold ($M = 10000$) 24

5.6 Reconstruction time vs maximum set size M when ($N = t$) 25

5.7 Reconstruction time vs maximum set size M when ($N = t = 2$) 26

5.8 Share generation time of a single participant 27

6.1 The tables T_L and T_R, and the subtables T_{L0},T_{L1} and T_{R0},T_{R1} [21]. 32

6.2 The possible combinations of locations to which Alice and Bob map their inputs [21]. Each dashed box represents a different combination. 33

List of Tables

3.1 Notations 7

6.1 Comparison of OT-MP-PSI Solutions 28

Chapter 1

Introduction

The Over Threshold Multiparty Private Set Intersection (OT-MP-PSI) problem is a cryptographic problem where N participants each hold a set that contains at most M elements. These participants want to know which elements appear in at least t sets for a certain threshold t without revealing any information about elements that do not meet this threshold. A visualization of the problem is shown in Figure 1.1 where $N = 6$, $M = 5$, and $t = 3$. Such a protocol is practical in situations where the data is sensitive and the data aggregation across multiple agents yields useful information. For example, when a malicious actor targets the networks of several institutions, this is classified as a multi-institutional attack. The institutions are willing to share data related to that specific attack, but the rest of the network data is still sensitive.

While OT-MP-PSI has diverse applications such as private file deduplication on cloud [24] or identifying high-risk individuals in the spread of disease [2], this thesis uses the problem of multi-institution attacks as a primary running example. Multi-institution attacks aim to exploit vulnerabilities in numerous nodes such as servers, user computers, and IoT devices located in various institutions, resulting in substantial financial losses and the compromise of sensitive information [25]. The type of attacks can range from exploiting incorrectly configured devices to zero-day software vulnerabilities. Effectively countering these attacks necessitates timely information exchange between institutions due to the decentralized nature of security management for the target nodes, each overseen by numerous independent security teams. Although this poses a technical challenge, it also introduces a privacy concern, as institutions are reluctant to share information about their networks when it is not related to a specific attack.

A large portion of these attacks are characteristically initiated from a small number

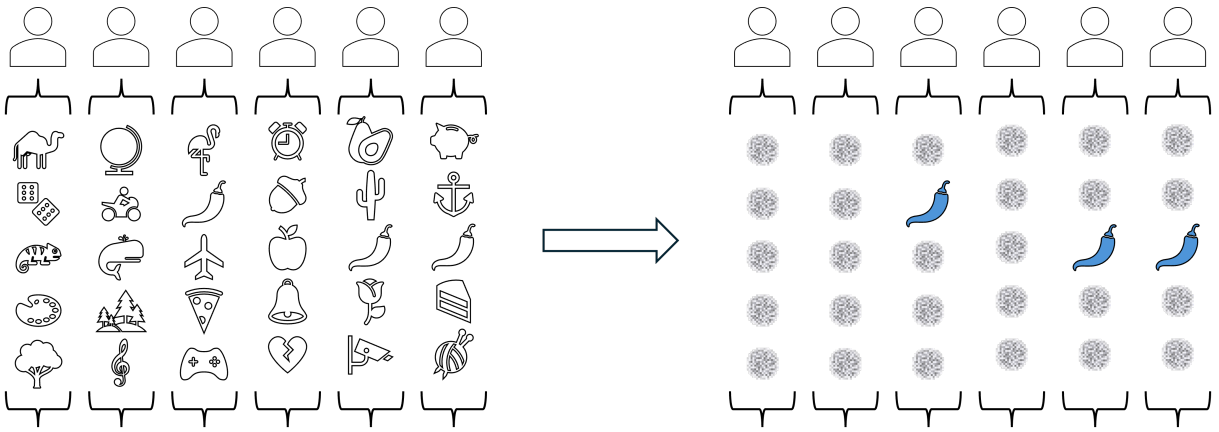


Figure 1.1: The OT-MP-PSI protocol visualization ($t = 3$)

of IP addresses that are external to the institutions. Moreover, an external IP address initiating a connection to the institution's internal nodes is an atypical behavior if this connection is not for a publicly advertised service such as a website. Following these characteristics, Zabarah et al. [25] propose a simple and efficient criterion for detecting multi-institution attacks. If an external IP initiates connections to t or more institutions in a set time interval, it is considered a multi-institution attacker, where t is an empirically decided threshold. This solution is simple and efficient, greatly reducing the collaboration required between institutions because it does not depend on the type of attack and does not require manual analysis. However, it requires institutions to share information about their networks, which is an important privacy concern. An efficient OT-MP-PSI protocol is the missing piece in this solution, as the output of the protocol is exactly the criteria proposed by Zabarah et al.

Kissner and Song proposed the first solution to the OT-MP-PSI problem [13]. However, their solution requires a high number of communication rounds and it is computationally expensive. Specifically, their protocol requires $\mathcal{O}(N)$ communication rounds, $\mathcal{O}(N^3M)$ communication complexity, and $\mathcal{O}(N^3M^3)$ computation complexity. This makes it impractical for real-life applications with large data sets because it requires each participant to have significant computational resources and to be online for the duration of the protocol. Additionally, the sequential nature of the protocol hinders the scalability as adding more compute resources does not directly result in faster execution times.

Mahdavi et al. [16] introduce a solution with a constant number of rounds and better communication complexity. However, their scheme has $\mathcal{O}(M(N \log M/t)^{2t})$ computational

complexity. This is too high for the malicious IP detection problem in institutional networks, which can have thousands of external connections at any given time.

We claim that it is possible to perform non-interactive OT-MP-PSI with linear computation complexity in the data set size. We propose a non-interactive OT-MP-PSI protocol with $\mathcal{O}(t^2 M \binom{N}{t})$ computational complexity. This efficiency improvement, in particular, the set size M not being under the exponentiation, enables the practical usage of the protocol in the aforementioned multi-institution attack detection problem because the nature of the problem requires the processing of very large IP address sets as each institution gets connected to hundreds of thousands of IP addresses every hour. In a setting with 33 participating institutions and at most 144,045 IPs connecting to any institution, the system can detect malicious IPs in 170 seconds, as we show in our experiments. The previous state-of-the-art solution [16] requires multiple days to complete the same computation.

Similarly to Mahdavi et al. [16], we use Shamir's secret sharing [23] to reveal identical items that are over the threshold. Shamir's secret sharing allows a data to be split into many shares such that any size-t subset of the shares can be used to reconstruct the original data. However, subsets smaller than t do not reveal anything about the original data. We make every participant create a single secret share for each element they own, and then combine these secret shares among participants to find the elements that exist in t or more sets. The main challenge with this approach is that all t combinations of secret shares must go through the reconstruction process to find the shares corresponding to the same set element because secret shares do not reveal anything about the underlying set element. A naive approach requires $\binom{N}{t} M^t$ different combinations. Mahdavi et al. [16] uses a binning technique that reduces the necessary combinations to $\mathcal{O}(M(N \log M/t)^{2t})$. As our main contribution, we introduce a new hashing scheme that reduces this number to $\mathcal{O}(tM\binom{N}{t})$.

Moreover, we show that our protocol efficiently extends to the well-studied two-party private set intersection ($N = t = 2$) and the multi-party private set intersection ($N > 2, t = N$) problems. By bridging these two problems and the OT-MP-PSI problem with a single protocol, we provide a flexible solution that can be used in various applications and adapt when the requirements change.

We show the protocol's security under the semi-honest multi-party computation model. The protocol uses a dedicated aggregator server that combines the secret shares without learning anything about the contents of the sets, and all the institutions hold a single symmetric key. The aggregator has to be non-colluding, but participants colluding among themselves does not affect the system's security.

We implement the protocol and evaluate its performance on real-world network connection logs from the CANARIE IDS Program [4] to show its practicality.

Chapter 2

Preliminaries

2.1 Security Model

In general terms, a semi-honest party adheres to the established protocol but records all its intermediate calculations and any messages it receives from another party [18]. We say a protocol is secure under the semi-honest model if each party cannot gain any other information except the intended output of the protocol. More formally, let $f : (\{0,1\}^*)^m \longmapsto (\{0,1\}^*)^m$ be an m-ary functionality, where $f_i(x_1, ..., x_m)$, denotes the i^{th} element of $f(x_1, ..., x_m)$. Let Π be an m-party protocol for computing f. The view of the i^{th} party during an execution of Π on $\overline{x} = (x_1, ..., x_m)$ is defined as $VIEW_i^{\Pi} = (x_i, m_1, ..., m_t)$ where m_i represents the i^{th} message it has received. We say that Π privately computes f with respect to the semi-honest model if there exists a polynomial-time algorithm, denoted S, such that for every $i \in [m]$

$$\{S(x_i, f_i(\overline{x})\}_{\overline{x}\in(\{0,1\}^*)^m} \stackrel{c}{\equiv} \{VIEW_i^{\Pi}(\overline{x})\}_{\overline{x}\in(\{0,1\}^*)^m} \tag{2.1}$$

where $\stackrel{c}{\equiv}$ denotes computationally indistinguishable.

In our case, the function to be computed privately is as follows: Let P denote the set of participants, A denote the aggregator, and S_i denote the i^{th} participant's set.

$$f(S_1, S_2, \ldots, S_N, \{\}) = \begin{cases} I = \{s \in S \mid \sum_{i=1}^{N} \mathbb{I}(s \in S_i) \geq t\} & \text{for all } \boldsymbol{P_i} \in \boldsymbol{P} \\ \{(\mathbb{I}(s \in S_1), \mathbb{I}(s \in S_2), \ldots \mathbb{I}(s \in S_N))\}_{\forall s \in I} & \text{for } \boldsymbol{A} \end{cases} \tag{2.2}$$

I.e., All $\boldsymbol{P_i}$ learns the set elements that exist in t or more sets, and the Aggregator $\boldsymbol{A}$ learns which of the $\boldsymbol{P_i}$ has the elements that are over the threshold t. Note that $\boldsymbol{A}$ does not learn

the plain-text values of the elements; it just learns who owns the elements that are over the threshold.

2.2 Shamir's Secret Sharing

The objective of secret sharing is to divide a secret S into m shares so that it can be reconstructed when a specific subset of these shares is combined. In a (t, m)-threshold scheme, any group of t parties can collaborate to retrieve the secret S using their shares, while any group with fewer than t members cannot gather any information about the secret, regardless of their efforts to collude.

In Shamir's secret sharing [23], the distributing party generates $t-1$ values $\{c_i\}_{i\in[t-1]}$ chosen at random from some finite field F_p of prime order p and forms the polynomial

$$f(x) = c_{t-1}x^{t-1} + c_{t-2}x^{t-2} + ... + c_1x + S \tag{2.3}$$

The distributing party generates m shares by evaluating f at m publicly-known distinct values. For instance, the share for party i (with $i \in F_p$) is $s_i = (i, f(i))$. Since t points uniquely determine a polynomial of degree $t-1$, anyone possessing t shares s_i can recover the secret S using Lagrange interpolation:

$$s = \sum_{i=1}^{t} \left(s_i \cdot \prod_{j=1, j\neq i}^{t} \frac{-j}{i-j} \right) \tag{2.4}$$

In our protocol, we will leverage Shamir's secret sharing to reveal a set element only if t threshold sets include that element.

Chapter 3

Protocol

Here, we describe our OT-MP-PSI protocol. The principal idea is that all participants create secret shares for each element in their set, using the same polynomial, and send these shares to the aggregator. The aggregator can only reconstruct secrets that have t or more shares. The aggregator then broadcasts the elements that it could reconstruct. This approach has a couple of challenges.

3.1 Using the same polynomial

We cannot directly use Shamir's secret sharing as our protocol would be compromised if there was a central party that distributes the shares. This is so because other parties would have to disclose their set elements to this party to get a secret share. Mahdavi et al. [16] use an interactive protocol with a key holder server to create secret shares obliviously. Although this approach provides some resistance to collusion, it is expensive and slow. We decided to trade off this collusion resistance with performance. In our protocol, participants communicate a secret key K among themselves, which is not disclosed to the Aggregator, and they produce the coefficients of the polynomial themselves using a keyed hash function, as described in equation 3.1

$$P_s^K(x) = H_K^{t-1}(s)x^{t-1} + H_K^{t-2}(s)x^{t-2} + \cdots + H_K(s)x + s \tag{3.1}$$

Where s is the secret to be shared, H_K is the hash-based message authentication code (HMAC), K is the secret key, and the superscripts of H_K indicate iteration, i.e.,

Table 3.1: Notations

$\boldsymbol{P_i}$	$\coloneqq$	i^{th} Participant
S	$\coloneqq$	Domain of elements
S_i	$\coloneqq$	Set of elements held by the participant i
$s_{i,j}$	$\coloneqq$	j^{th} element in the set S_i
$\boldsymbol{A}$	$\coloneqq$	Aggregator
$H(\cdot)$	$\coloneqq$	Cryptographic hash function
$H_K(\cdot)$	$\coloneqq$	Hash-based message authentication code (HMAC) with key K
$h()$	$\coloneqq$	The hash function used to map elements to array indexes
$P_s(x)$	$\coloneqq$	Polynomial used for secret share creation
K	$\coloneqq$	Symmetric secret key used by participants
		Encryption Scheme
Enc_K	$\coloneqq$	Encryption with symmetric key K
Dec_K	$\coloneqq$	Decryption with symmetric key K
		Protocol Parameters
N	$\coloneqq$	Number of participants
t	$\coloneqq$	Threshold
M	$\coloneqq$	Maximum number of elements in each set

$H_K^i(s) = H_K(H_K^{i-1}(s))$. With this approach, the Aggregator does not learn more than the intended output of the protocol as long as it is non-colluding. This approach is completely decoupled from our new hashing scheme, and if users would prefer the collusion resistance, they can still use the share creation approach of [16] with our new hashing scheme.

3.2 Reconstruction Complexity

If we naively send all secret shares to the aggregator, the aggregator would need to try $\binom{N}{t}M^t$ different combinations to find the matching shares. This is infeasible if the set sizes are large. We need to give the aggregator some clue about how to combine the shares, but

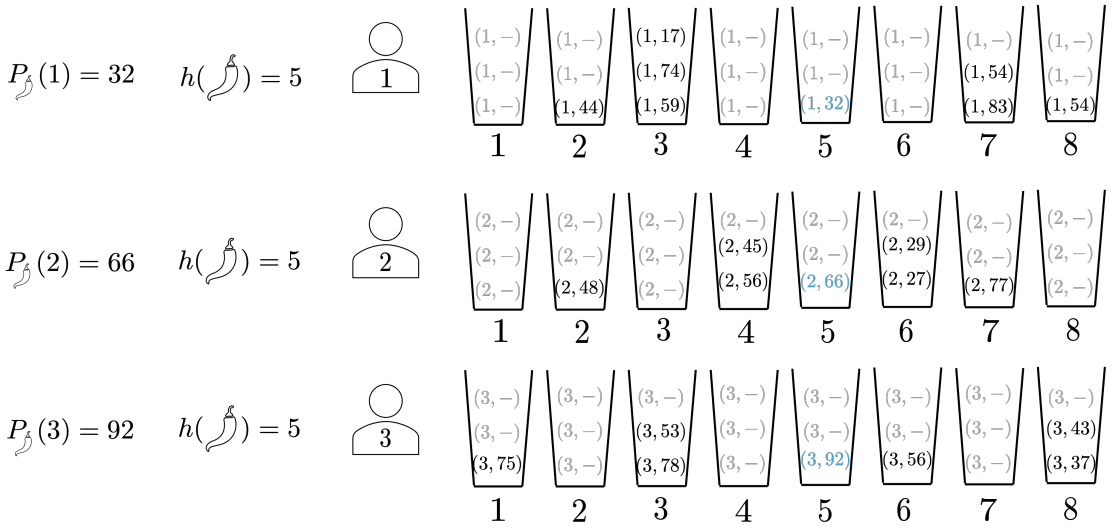


Figure 3.1: The binning technique used in [16], light gray numbers represent the dummy shares, P is the polynomial function, and h is the hash function used for binning.

if we just put an identifier (e.g. keyed hash of the element) alongside the secret share, then the protocol would leak the similarity distribution of the sets so the aggregator could see which sets have how many overlapping elements, even when those elements are under the threshold. This is more information than the intention of our protocol's output.

A common approach to this problem is to put the shares into bins using a hash function. In this way, the aggregator has to try t combinations of shares that are only from the same bins. However, in order to not leak any distribution information, participants have to pad all of the bins with dummy shares so that all of the bins are equal in size. A diagram of this technique is shown in Figure 3.1. This approach reduces the complexity to $O(M(N \log M/t)^{2t})$, as shown in [16]. Experimentally, the maximum bin size, corresponding to the $\log M$ term in the complexity, carries large constants. This prevents the system from scaling for our problem size.

As the main contribution of this thesis, we introduce a new approach. Instead of hashing shares into bins, they are hashed into array indexes. This means that each index can store only a single share. When multiple shares are hashed into the same index, we select one of them using a pseudo-random ordering. Consequently, a single array will be missing some shares. Each participant creates multiple such arrays. The motivation is that if participants create enough arrays, missing values in one array will appear in others, and the probability of failure can be bounded to an acceptably low value with minimal

cost. The main advantage of this approach is that the aggregator does not need to try combinations of shares. It only needs to try combinations of participants. After selecting t participants, the aggregator applies Lagrange interpolation to the shares corresponding to the same array and index (e.g., the first participant's first array's first index and the second participant's first array's first index). An illustration of the new hashing scheme is shown in Figure 3.2. We show that if the size of each array is set to $t \times M$ and each participant produces 20 arrays (with some additional optimizations), the probability of getting an incorrect result is smaller than 2^{-40}. We explain this hashing scheme in detail and show the failure probability in Chapter 4.

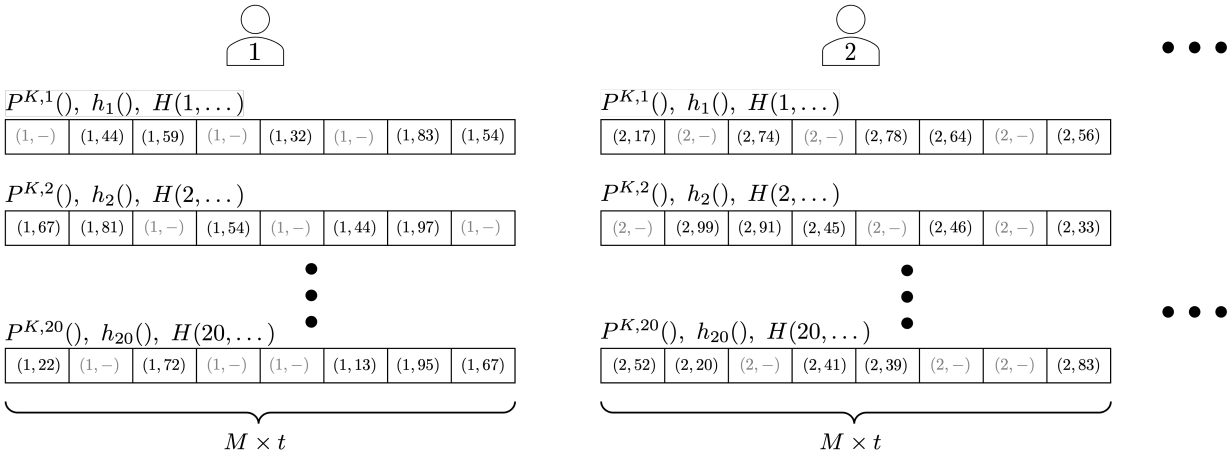


Figure 3.2: The new hashing scheme. Every participant creates multiple arrays, P is the polynomial function, h is the hash function used for mapping, H is the hash function used for pseudo-random ordering, and light gray numbers represent the dummy shares.

3.3 Identifying the Successful Reconstructions

Another challenge we have to solve is identifying successful reconstructions. That is, how does the aggregator know when it reconstructs an actual set element and not a random element from the finite field F_p? We considered using a finite field larger than the size of the S. Even though this is fine for our specific use case of detecting malicious IPs, it limited the protocol's functionality for broader applications.

We tackled this issue by decoupling the data transfer from the private set intersection. Instead of creating the secret share of the set elements, participants create secret shares

of the number 0; the set element is used to create the polynomial coefficients. In this way, if the shares correspond to the same element, they reconstruct the number 0, and if not, they reconstruct a random element of F_p. The actual elements are encrypted with the symmetric key K using a randomized encryption scheme and sent to the aggregator separately. If there are t shares that reconstruct the number 0, the aggregator publishes one of the ciphertexts that correspond to those t shares. The new version of the polynomial is described in equation 3.2

$$P_s^K(x) = H_K^{t-1}(s)x^{t-1} + H_K^{t-2}(s)x^{t-2} + \cdots + H_K(s)x + 0 \tag{3.2}$$

3.4 Protocol Description

Combining the solutions of these three challenges, we describe our protocol as follows:

1. All participants $\boldsymbol{P_i}$ decide on a symmetric secret key K, without involving $\boldsymbol{A}$
2. $\boldsymbol{A}$ broadcasts a random number r for this round.
3. All $\boldsymbol{P_i}$ create two 2D arrays called *Shares* and *EncryptedElements*, both with 20 columns and tM rows. They fill the arrays according to equation 3.3 for each element $s_{i,j}$ in their set S_i and for each $\alpha \in \{1, \ldots, 20\}$.

   if

   $$H(\alpha, s_{i,j}, K, r) \leq H(\alpha, s, K, r)\ \forall s \in \{s \in S_i \colon h_\alpha(s, K, r) = h_\alpha(s_{i,j}, K, r)\}$$

   then (3.3)

   $$Shares[\alpha][h_\alpha(s_{i,j}, K, r)] = P_{s_{i,j}}^{K,r,\alpha}(i)$$
   $$EncryptedElements[\alpha][h_\alpha(s_{i,j}, K, r)] = Enc_K(s_{i,j})$$

4. All $\boldsymbol{P_i}$ fill up the empty indexes in *Shares* and *EncryptedElements* with random bits, then send the arrays to the Aggregator $\boldsymbol{A}$.
5. After receiving N *Shares* arrays, $\boldsymbol{A}$ takes every t combination of $\boldsymbol{P_i}$ and checks if the shares in the same indexes produce a valid secret.

6. $\boldsymbol{A}$ broadcasts the cipher-text values of valid reconstructions

7. All $\boldsymbol{P_i}$ use Dec_K to decrypt the results broadcasted by $\boldsymbol{A}$; they then deduplicate the results and learn the elements that appear in at least t sets

Chapter 4

The New Hashing Scheme

In this chapter, we introduce a novel hashing scheme designed to efficiently address the matching problem mentioned earlier. The challenge lies in the fact that the secret shares, by their nature, do not reveal the underlying element, making it difficult to combine them. In this hashing scheme, each participant places their secret shares into an array using a hash function. This ensures that identical elements are assigned to the same array indices across participants. Consequently, the aggregator only needs to process shares that fall into the same indices.

In cases of hash collisions, where multiple elements map to the same index, we must select a single element from those that collide. Rather than choosing this element randomly, we employ an additional hash function to establish an order among the shares and then select the smallest one according to this order. This method increases the likelihood that different participants will choose the same element for a given index.

Participants repeat this process to create multiple arrays using a distinct mapping hash function and ordering hash function for each array. Our goal is to limit the probability of missing an intersection to an acceptably low level by re-randomizing the mapping and ordering for each array. We set our target failure probability to 2^{-40} as 40 is a common statistical security parameter [6, 3, 15].

We first show that if the size of the array is set to $M \times t$, the probability of missing an intersection is a constant number smaller than 1 for a single array. Let $h : \{0,1\}^* \longmapsto [Mt]$ denote the mapping hash function, H denote the collision-resistant ordering hash function, and $Shares_i$ denote the array of the participant $\boldsymbol{P_i}$.

Given a set element $s_{i,j} \in S_i$ of the participant $\boldsymbol{P_i}$, let p denote the probability of $H(s_{i,j}) < H(s_{i,k})$ for a random $s_{i,k} \in S_i$. Since the output distribution of H is uniform, p

can be calculated with the following equation:

$$p \coloneqq \frac{H(s_{i,j})}{max(H)}$$

The probability that a random set element $s_{i,k}$ maps to the same index as $s_{i,j}$ is given by

$$P(h(s_{i,j}) = h(s_{i,k}) \mid s_{i,j}) = \frac{1}{Mt}$$

The probability that a random set element $s_{i,k}$ blocks $s_{i,j}$, that is, $s_{i,k}$ maps to the same index and is smaller than $s_{i,j}$ in the ordering:

$$P\left(h(s_{i,j}) = h(s_{i,k}) \; and \; H(s_{i,k}) < H(s_{i,j}) \mid s_{i,j}\right) = p \times \frac{1}{Mt} = \frac{p}{Mt}$$

To calculate a lower bound for the probability that $s_{i,j}$ will be placed in the array, we raise the probability that $s_{i,j}$ is not blocked by a random element to the M^{th} power because $|S_i| \leq M$.

$$\begin{aligned}
&P(H(s_{i,j}) < H(s) \; \forall s \in \{s \in S_i \; : \; h(s_{i,j}) = h(s)\} \mid s_{i,j}) \\
&= P(s_{i,j} \in Shares_i) \\
&= \left(1 - \frac{p}{Mt}\right)^{|S_i|-1} \geq \left(1 - \frac{p}{Mt}\right)^{M}
\end{aligned}$$

With large M, the expression $\left(1 - \frac{p}{Mt}\right)^{M}$ approximates $e^{-p/t}$ and for $M > 50$ this approximation does not affect any of our results. We now calculate the probability that t different participants will put $s_{i,j}$ in the array given that $s_{i,j}$ is in their sets. This is so because, for a successful reconstruction, all participants that have the set element have to put the element into the array. Since all participants use the same ordering hash function for the same array, the p values will be the same, so we can raise the $e^{-p/t}$ to the t^{th} power to calculate this probability.

$$P(s_{i,j} \in Shares_k \; \forall_{k \in [t]} \mid s_{i,j} \in S_k \; \forall_{k \in [t]}) \geq \left(e^{-p/t}\right)^t = e^{-p}$$

From this equation, it follows that the probability of failure for a particular set element is $1 - e^{-p}$, given that this element exists in t different sets.

To calculate the probability of failure for any set element, we treat p as a continuous random variable with a uniform distribution between 0 and 1.

$$P(\text{fail} \mid p) \leq 1 - e^{-p}$$
$$P(\text{fail}) \leq \int_0^1 (1 - e^{-p})dp = e^{-1} \approx 0.3678$$

This shows that the probability of failure, more explicitly, the probability of missing any given over-threshold intersection with a single array is at most e^{-1}. Every participant needs to create 28 different arrays so that the probability of failure is at most $(e^{-1})^{28} \approx 2^{-40.4}$

Next, we will explain two optimizations to reduce the number of arrays.

4.1 Reversing the ordering

Instead of using a unique ordering hash function for each array, using the same ordering hash function for every two consecutive arrays and reversing the ordering for even-numbered arrays gives better results. The intuition behind this approach is that we know the "unlucky" elements in the odd-numbered arrays; these are elements with big p values. By reversing the order and making them "lucky" in the next consecutive array, we get better probabilities than re-randomizing for each array.

With this technique, the value of p for a particular element will be equal to $1 - p$ in the next array. Therefore, the probability of missing a particular intersection in two consecutive arrays is given by

$$P(\text{fail with two arrays} \mid p) \leq (1 - e^{-p})(1 - e^{-(1-p)})$$

We then calculate the probability of failure for any intersection similarly to the single array calculation.

$$P(\text{fail with two arrays}) \leq \int_0^1 (1 - e^{-p})(1 - e^{-(1-p)})dp$$
$$= 3e^{-1} - 1 \approx 0.10363$$

This technique reduces the required number of arrays to 26 to obtain $(3e^{-1} - 1)^{13} \approx 2^{-42.5}$ failure probability.

4.2 Utilizing the empty indexes

Normally, after inserting their elements into the arrays, the participants would fill the empty array indexes with dummy shares. However, we can utilize this wasted space to reduce the number of required arrays. With this optimization, participants will do a second insertion after the first insertion for every array. They will use a different mapping hash function for the second insertion and reverse the ordering hash function from the first insertion. While doing the second insertion, elements from the first insertion have priority in the array; in other words, if an element maps to an already occupied position, we do not make any changes to the array.

Therefore, two events must occur for an element to be successfully put into the array in the second insertion: it must map into an empty index and be the smallest element among the elements that map into the same index. We already know the probability of the second event occurring in t different sets; it is $e^{-(1-p)}$ due to reverse ordering. To calculate the probability of the first event, we first calculate the probability of any array index being empty after the first insertion. We do this by calculating the probability of a single element not mapping to that index repeated M times. Again, the approximation does not affect the results for $M > 50$.

$$P(Shares[x] \text{ is empty}) = \left(\frac{tM-1}{tM}\right)^{|S_i|} \geq \left(\frac{tM-1}{tM}\right)^{M} \approx e^{-1/t}$$

We then calculate the probability of $s_{i,j}$ being mapped into empty indexes in all t sets in the second insertion, given that $s_{i,j}$ exists in t different sets.

$$P(Shares_k[h(s_{i,j})] \text{ is empty } \forall_{k\in[t]} \mid s_{i,j} \in S_k \ \forall_{k\in[t]}) \geq \left(e^{-1/t}\right)^t = e^{-1}$$

Now, we multiply these two probabilities to find the second insertion success probability of a particular intersection.

$$P(\text{success in second insertion} \mid p) \geq e^{-(1-p)} \times e^{-1} = e^{p-2}$$

Lastly, the probability of missing a particular intersection in a single array is given by the failure probability of the first insertion multiplied by the failure probability of the second insertion. Afterward, we calculate the failure probability for any intersection.

$$\begin{aligned} P(\text{fail} \mid p) \leq & (1-e^{-p})(1-e^{p-2}) \\ P(\text{fail}) \leq & \int_0^1 (1-e^{-p})(1-e^{p-2})dp \\ = & 2e^{-2} \approx 0.2706 \end{aligned}$$

This approach reduces the required number of arrays to 22 to have $(2e^{-2})^{22} = 2^{-41.5}$ failure probability.

Finally, we can get better results by combining the two optimizations; more explicitly, we will do second insertions for every array and order reversing for every two consecutive arrays. In this case, the failure probability for a particular intersection in two consecutive arrays is given by

$$P(\text{fail with 2 arrays} \mid p) \leq \underbrace{(1-e^{-p})(1-e^{p-2})}_{\text{first array}} \underbrace{(1-e^{-(1-p)})(1-e^{-p-1})}_{\text{second array}}$$

We then calculate the failure probability for any intersection

$$\begin{aligned} & P(\text{fail with 2 arrays}) \\ & \leq \int_0^1 (1-e^{-p})(1-e^{p-2})(1-e^{-(1-p)})(1-e^{-p-1})dp \\ & = 2e^{-1} + 2e^{-2} + 3e^{-4} - 1 \approx 0.06138 \end{aligned}$$

The combination of these two approaches reduces the number of required arrays to 20 to obtain a failure probability of $(0.06138)^{10} = 2^{-40.3}$.

Chapter 5

Evaluation

In this chapter, we evaluate the security and performance of our OT-MP-PSI protocol. First, we provide a security analysis using the semi-honest multi-party computation model. We then provide a theoretical complexity analysis for computational and communication costs. Finally, we present experimental performance benchmarks for our protocol.

5.1 Security Analysis

In this section, we analyze the security of our protocol under the semi-honest model and show that it is secure by constructing the simulators for the participants and the aggregator.

Theorem 1. *The protocol described in section 3.4 is secure in the semi-honest model given that the Aggregator* $\boldsymbol{A}$ *is non-colluding.*

Proof. The construction of the simulator $S_{\boldsymbol{P_i}}((S_i, K, r), f_P(\dots))$ is trivial as the only message that participants receive is the encryption of the protocol output. This can be easily simulated since participants have the encryption key. Similarly, the construction of a simulator for a subset of colluding participants is also trivial, since every participant receives the same message from the aggregator.

We can construct the simulator $S_{\boldsymbol{A}}(r, f_A(\dots))$ for the aggregator as follows:

1. The simulator creates N different *EncryptedElements* arrays and fills them with random bits. Since the original *EncryptedElements* array is encrypted, the simulated array will be indistinguishable from the real one.

2. The simulator creates N different sets and puts random elements that will satisfy the protocol output of the aggregator. For example, if a subset of the participants has a common element, the simulator also puts a common element in the simulated sets. The simulator then creates the *Shares* array for each simulated set, following the protocol description and using a randomly selected K. The simulated *Shares* arrays will have the same probabilistic distribution of successful reconstructions as the real ones; secret shares do not give any information without a successful reconstruction, and since the polynomial coefficients are rerandomized for each sub-array of a single *Shares* array, unintended successful reconstructions by combining shares across different arrays of the same participant will not be possible. Thus, simulated *Shares* arrays will be indistinguishable from the real ones.

□

5.2 Complexity Analysis

In this section, we provide a theoretical complexity analysis for our OT-MP-PSI protocol. We analyze the computational and communication costs for participants and the aggregator.

5.2.1 Computational Complexity

Theorem 2. *The computational complexity of the aggregator for the protocol described in section 3.4 is* $\mathcal{O}(t^2M\binom{N}{t})$.

Proof. The aggregator has to try $\binom{N}{t}$ different combinations of participants. For each combination, the aggregator has to do $\mathcal{O}(tM)$ Lagrange interpolations. For threshold t, a single Lagrange interpolation has a complexity of $\mathcal{O}(t)$. Thus, the total complexity is $\mathcal{O}(t^2M\binom{N}{t})$. □

As a corollary of theorem 2, we can say that the computational complexity for the two-party private set intersection problem ($N = t = 2$) is $\mathcal{O}(M)$, and for the multi-party private set intersection problem ($N = t$) is $\mathcal{O}(N^2M)$.

Theorem 3. *The computational complexity of the participants for the protocol described in section 3.4 is* $\mathcal{O}(t^2M)$.

Proof. As the size of the *Shares* and *EncryptedElements* arrays are $\mathcal{O}(tM)$, the participants have to create $\mathcal{O}(tM)$ secret shares, and encryptions. The cost of creating a secret share is $\mathcal{O}(t)$, and the cost of a single encryption is $\mathcal{O}(1)$. Thus, the total complexity is $\mathcal{O}(t^2M)$. □

5.2.2 Communication Complexity

Theorem 4. *The communication complexity of the protocol described in section 3.4 is* $\mathcal{O}(tMN)$.

Proof. All the participants have to send their *Shares* and *EncryptedElements* arrays to the aggregator. The size of each *Shares* and *EncryptedElements* array is $\mathcal{O}(tM)$. Thus, the total communication complexity is $\mathcal{O}(tMN)$. □

5.3 Correctness Evaluation

We experimentally evaluate the failure rate of our hashing scheme to validate our theoretical failure probability analysis. Figure 5.1 shows the number of missed over-threshold intersection elements out of 10 million trials for different number of arrays. It can be observed that the experimental results are well below the computed upper bounds.

5.4 Performance Evaluation

In this section, we provide performance benchmarks for our OT-MP-PSI protocol. We measure the share generation time for each participant and the reconstruction time of the aggregator. We compare our results with the implementation of Madavi et al. [16] as the only other solution with a public code repository at the time of writing. We also evaluate our system using real-world network data obtained from the CANARIE IDS program [4].

5.4.1 Setup

We implement our protocol using Julia language with 430 lines of code. The cryptographic libraries that we use are SHA.jl and Nettle.jl. We use Julia's threads library for parallelization. We used the 61-bit Mersenne prime for the finite field. This is so that we can take

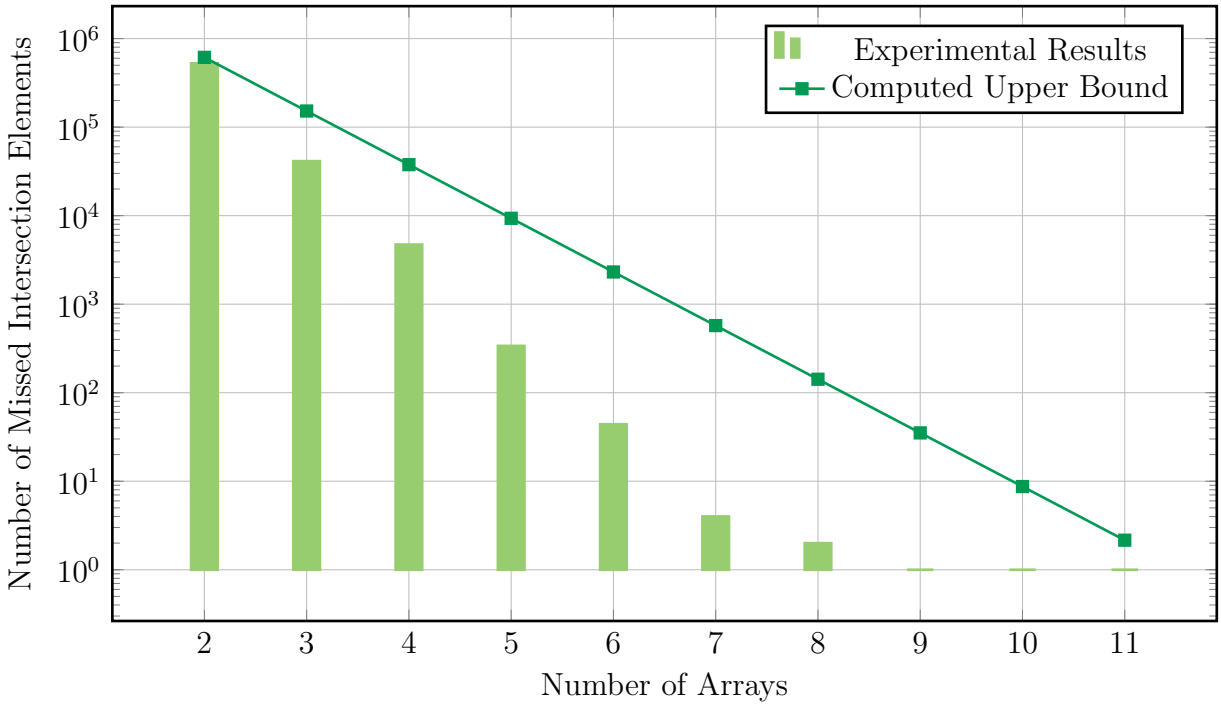


Figure 5.1: Number of missed Intersection Elements in 10^7 trials ($M = 200, t = 4$)

advantage of fast modulo arithmetic with Mersenne primes and use 128-bit integers instead of arbitrary precision integers. We use a server with 8 x Intel Xeon E7-8870 processors for all of our experiments. In total, we have 80 physical cores, each running at 2.4 GHz. The server has 1 TB of memory. However, none of our experiments require more than 30 GB of memory. The experiments for reconstruction runtime utilize all 80 cores. However, when benchmarking the share generation runtime, we use a single core to showcase the practicality of the protocol for lightweight clients.

5.4.2 Reconstruction

We compare the reconstruction time of our protocol with the reconstruction time of Mahdavi et al. [16] in Figure 5.2. We terminated some of the experiments from [16] because they ran for more than an hour. Our protocol is at least two orders of magnitude faster than Mahdavi et al. [16] for all parameters, and the difference increases exponentially with bigger threshold values.

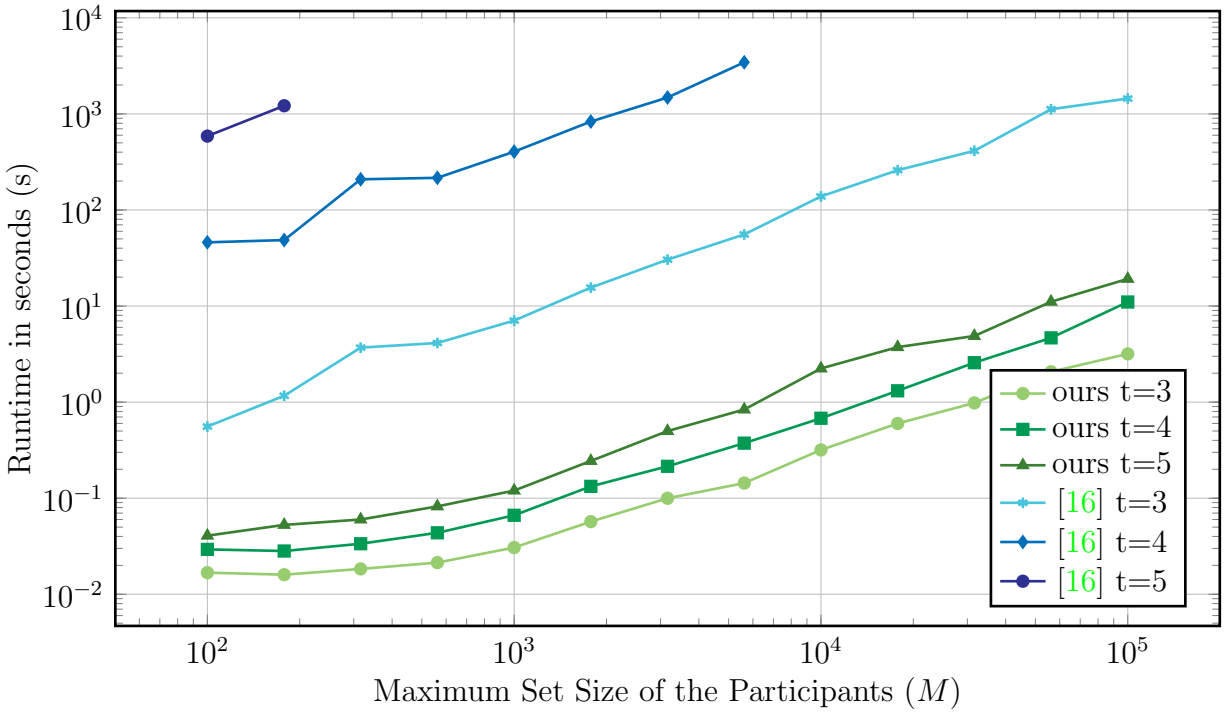


Figure 5.2: Reconstruction time comparison with Mahdavi et al. [16] ($N = 10$)

To show the practicality of our protocol in a real-world setting, we collected network connection logs from 54 participating institutions in the CANARIE IDS program [4] for a 1-week period from November 1st to November 8th. The OT-MP-PSI protocol is run for every one hour of the data. If an institution has no external IPs that start connections in that hour, the institution is not included in the protocol. We set the threshold value to 3 as that is the suggested value by [25]. The reconstruction time during this week is shown in Figure 5.3. The maximum reconstruction time during the week is 438 seconds, where 40 institutions participate in the protocol, and the maximum set size is 220,011. The average reconstruction time is 170 seconds, the average participating institution count is 33, and the average maximum set size is 144,045. Since the protocol runs once every hour, we consider the average reconstruction time acceptable for a real-world application.

We measure the reconstruction time of our protocol for different numbers of participants in Figure 5.4. We see that the reconstruction time increases polynomially with the number of participants. This is because the $\binom{N}{t}$ term in our protocol's complexity has an upper bound of $(\frac{eN}{t})^t > \binom{N}{t}$.

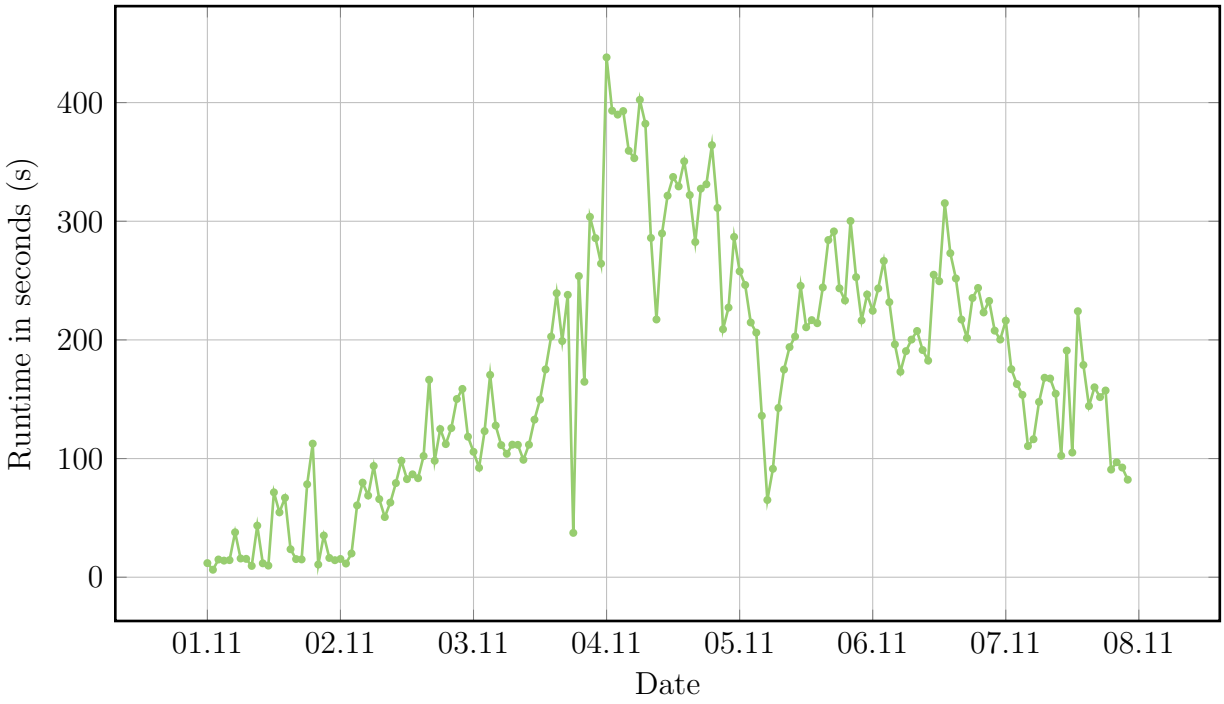


Figure 5.3: Reconstruction time on CANARIE data [4]

We measure the reconstruction time of our protocol for different threshold values in Figure 5.5. We can see that the reconstruction time increases exponentially with the threshold value until $t = N/2$, after which it decreases exponentially. This is because of the $\binom{N}{t}$ term in the complexity of our protocol.

To show the efficiency of our hashing scheme for the MP-PSI problem, we measure the reconstruction time of our protocol for different maximum set sizes while t is set to $t = N$ in Figure 5.6. As expected, we see a linear increase in the reconstruction time with the maximum set size.

We also show the efficiency of our hashing scheme for the 2P-PSI problem by measuring the reconstruction time of our protocol for different maximum set sizes while N and t are set to $t = N = 2$ in Figure 5.7. Similarly, we see a linear increase in the reconstruction time with the maximum set size.

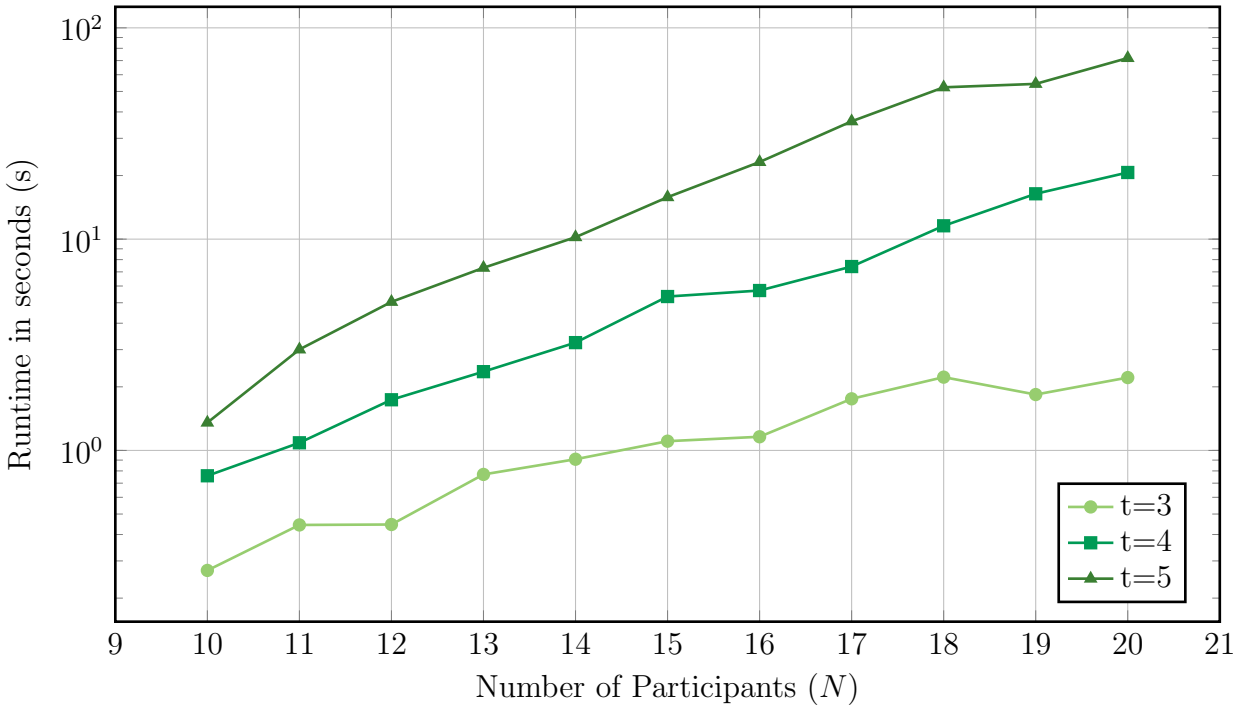


Figure 5.4: Reconstruction time vs number of participants ($M = 10000$)

5.4.3 Share Generation

We measure the share generation time of a single participant for different maximum set sizes in Figure 5.8. We see that the share generation time increases linearly with the maximum set size, which confirms the $\mathcal{O}(t^2 M)$ complexity of the participants.

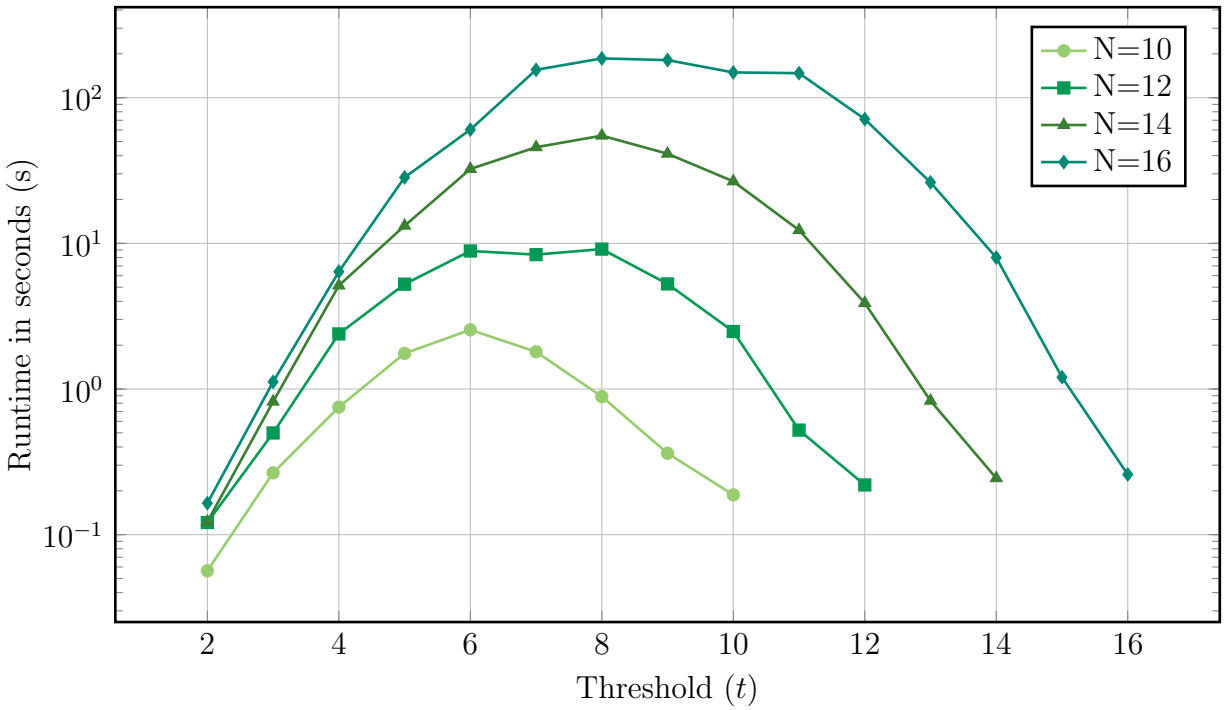


Figure 5.5: Reconstruction time vs threshold ($M = 10000$)

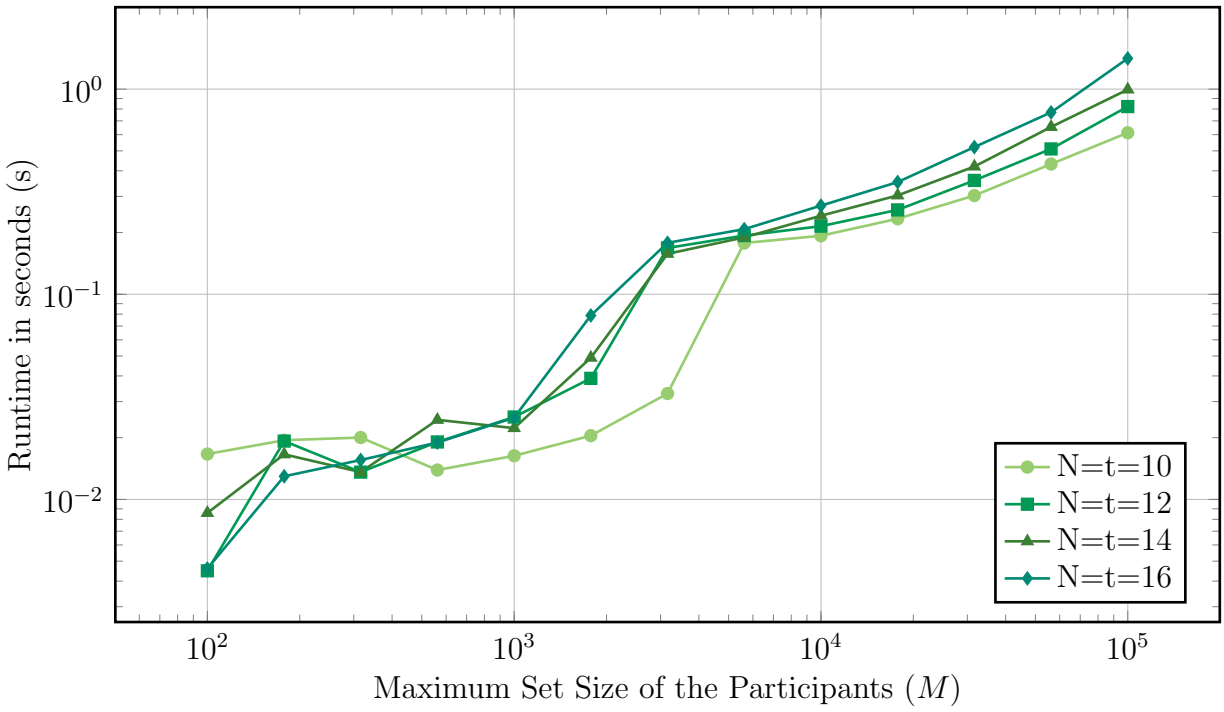


Figure 5.6: Reconstruction time vs maximum set size M when $(N = t)$

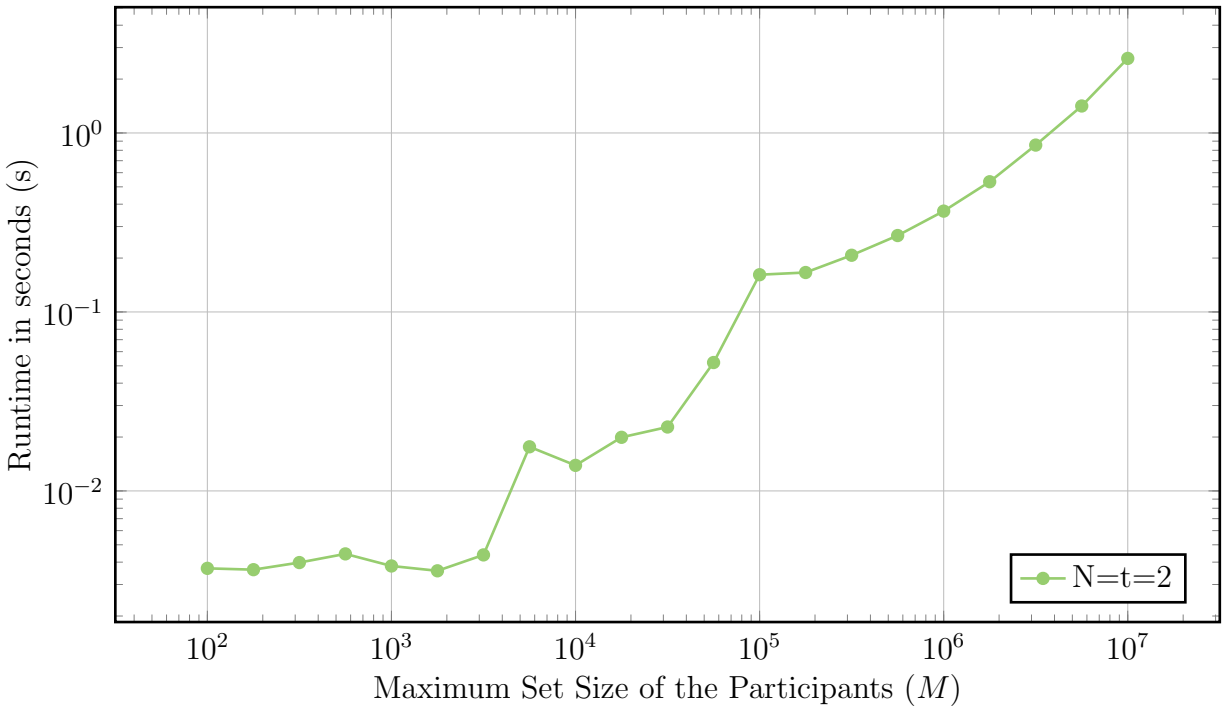


Figure 5.7: Reconstruction time vs maximum set size M when ($N = t = 2$)

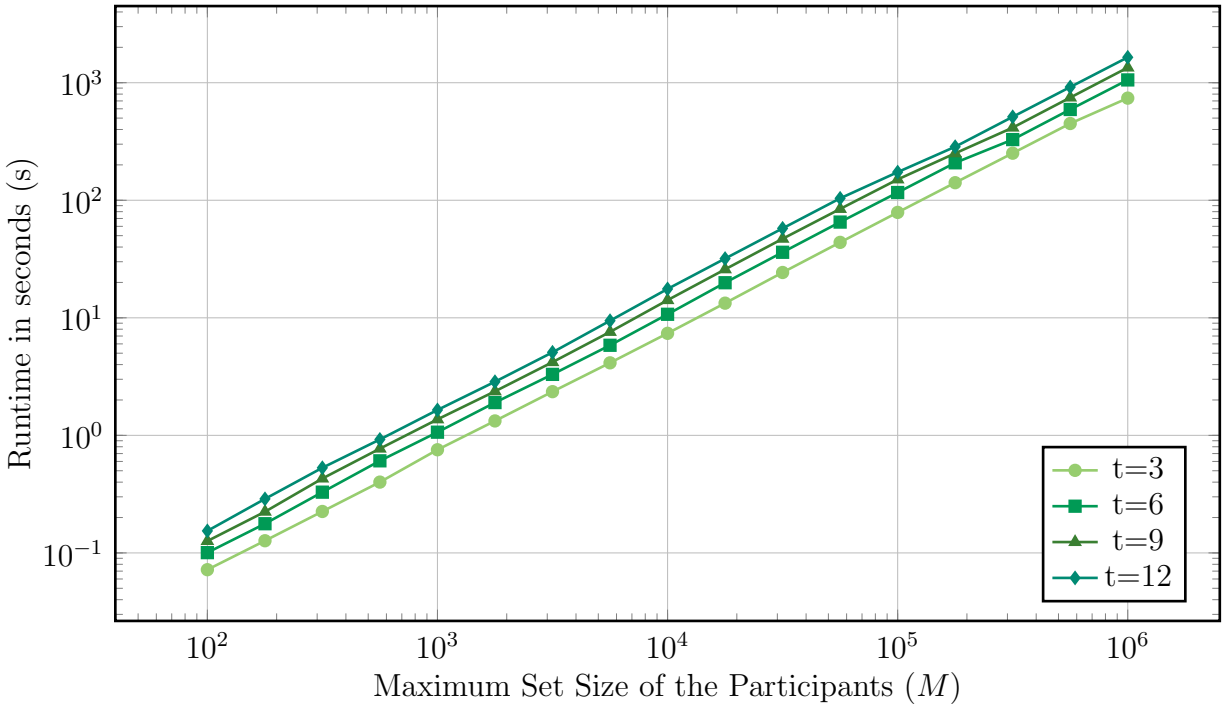


Figure 5.8: Share generation time of a single participant

Chapter 6

Related Work

In this chapter, we discuss the previous solutions to the OT-MP-PSI problem, some other closely related problems, and related work regarding our new hashing scheme. Table 6.1 shows an overview of other OT-MP-PSI solutions.

6.1 Previous Solutions to OT-MP-PSI

Table 6.1: Comparison of OT-MP-PSI Solutions

Solution	**Computational Complexity**	**Comm. Complexity**	**Comm. Rounds**	**Trust Assumptions**
Kissner and Song [13]	$\mathcal{O}(N^3M^3)$	$\mathcal{O}(N^3M)$	$\mathcal{O}(N)$	semi-honest with up to k collusions
Mahdavi et al. [16]	$\mathcal{O}(M(N \log M/t)^{2t})$	$\mathcal{O}(tMNk)$	$\mathcal{O}(1)$	semi-honest with up to k collusions
Stanek et al. [24]	$O(NM)$	$O(NM)$	1	trusted server
Our Solution	$\mathcal{O}(t^2M\binom{N}{t})$	$\mathcal{O}(tMN)$	1	semi-honest with non-colluding server
Our Solution + Mahdavi et al.'s share generation	$\mathcal{O}(t^2M\binom{N}{t})$	$\mathcal{O}(tMNk)$	$\mathcal{O}(1)$	semi-honest with up to k collusions

6.1.1 Kissner and Song

Kissner and Song proposed the first solution to the OT-MP-PSI problem [13]. They use polynomial rings to represent multisets. Given a multiset $S = \{S_j\}_{1 \leq j \leq k}$, they represent it as a polynomial $f(x) = \Pi_{j=1}^{k}(x - S_j)$. With this representation, the union of two sets becomes a product of two polynomials. They use homomorphic encryption to encrypt the polynomials; then, all the participants sequentially multiply their sets to produce a polynomial that represents the union of all sets. Then, they do homomorphic derivative operations on this polynomial to find the elements that exist in at least threshold number of sets. Their solution requires a high number of communication rounds. Specifically, their protocol requires $\mathcal{O}(N)$ communication rounds and $\mathcal{O}(N^3M)$ communication complexity. Moreover, multiplying the polynomials and evaluating them for each set element has a computational complexity of $\mathcal{O}(N^2M^3)$ for each participant, totaling to $\mathcal{O}(N^3M^3)$ computation when serialized. There are three reasons why this is worse than our $\mathcal{O}(t^2M\binom{N}{t})$ complexity. First, every arithmetic operation is performed with homomorphic encryption, which is computationally expensive. Second, each polynomial multiplication depends on the previous participants' multiplication, making it less parallelizable than our reconstruction process. Finally, in most use cases, M is much larger than N, i.e, few users with large datasets, resulting in $t^2\binom{N}{t} \ll N^3M^2$. However, if the threshold t is close to $N/2$ and N is large enough, this assumption might not hold.

6.1.2 Mahdavi et al.

Mahdavi et al. [16] introduce a solution with a constant number of rounds and better communication complexity. They use Shamir's secret sharing scheme to reveal identical items over the threshold, similar to our solution. They introduce a new protocol called oblivious pseudo-random secret sharing (OPR-SS) to allow a single key-holder to obliviously create secret shares for all participants. This approach provides a more relaxed non-collusion rule than our solution. They only require that the aggregator and the key-holder are not colluding with each other, which can be even more relaxed if multiple key-holders and aggregators are used. We intentionally trade this off with a stricter collusion rule in return for a more efficient protocol.

Their scheme has $\mathcal{O}(M(N \log M/t)^{2t})$ computational complexity due to the binning technique, which we substantially improve with our new hashing scheme.

6.1.3 Stanek et al.

When using cloud storage, end-to-end encryption of files prevents the cloud provider from deduplicating the popular files and results in inefficient use of storage resources. Stanek et al. [24] argue that while user-created files are private and should be encrypted, there is a significant amount of data that are not as sensitive, such as operating system files from full device backups. They create a system where if a file is popular enough, it gets revealed and the cloud provider deletes the extra copies. They introduce a threshold cryptosystem using Shamir's secret sharing, where participants can create decryption shares for a file they encrypted, and t decryption shares created for the same file can decrypt the chipertext. Their system requires two fully trusted servers, one identity provider for distributing the shares of the private key, and one indexing server to bypass the matching problem that our hashing scheme solves. A more trustless version of this system can be implemented with our OT-MP-PSI protocol.

6.2 Related Work Regarding the Hashing Scheme

Although there are no previous works that directly address the OT-MP-PSI problem with a hashing scheme, there are some works that address similar problems in the context of two-party private set intersection (2P-PSI) and multi-party private set intersection (MP-PSI) protocols, which are special cases of OT-MP-PSI where $N = t = 2$ for 2P-PSI and $t = N$ for MP-PSI. Our hashing scheme efficiently addresses both problems because our computational complexity $\mathcal{O}(t^2 M \binom{N}{t})$ becomes $\mathcal{O}(M)$ for $N = t = 2$ and $\mathcal{O}(N^2 M)$ for $t = N$ as shown in Section 5.2.1.

6.2.1 Optimized Binning

The binning technique used by Mahdavi et al. [16] can be further optimized with the recursive hashing proposed by Hemenway et al. [11]. The simple binning technique [16] reduces the number of comparisons by hashing elements into bins using a predefined hash function $h : U \to [B]$. Each bin is of size b, and secure comparisons are performed within each bin. This approach requires Bb^2 secure comparisons for two parties. If B is on the order of M ($B = \mathcal{O}(M)$), the maximum bin size is approximately $\log M / \log \log M$ with high probability. Consequently, this would necessitate $M(\log M / \log \log M)^2$ secure comparisons. Hemenway et al. [11] introduced an optimization that reduces the number of secure comparisons to $\mathcal{O}(M \log M)$ by recursively hashing each bin into sub-bins.

6.2.2 Cuckoo Hashing with Simple Hashing

Cuckoo Hashing. In simple terms, Cuckoo Hashing [19] is a technique used to avoid hash collisions where multiple hash functions are used to provide alternative locations for each key-value pair. It was originally designed to construct a worst-case constant lookup time hash table. However, it found many applications in the context of private protocols. While many other variants have been proposed, the original Cuckoo hashing introduced by Pagh and Rodler [19] uses two hash functions h_0, h_1 to map M elements to two tables T_0, T_1, each containing $(1+\varepsilon)M$ bins. Each bin accommodates at most a single element. The scheme avoids collisions by relocating elements when a collision is found using the following procedure: Let $b \in 0, 1$. An element x is inserted into a bin $h_b(x)$ in table T_b. If a prior item y exists in that bin, it is evicted to bin $h_{1-b}(y)$ in T_{1-b}. The pointer b is then assigned the value $1-b$. The procedure is repeated until no more evictions are necessary or until a threshold number of relocations has been performed.

A tempting approach involves having both parties, Alice and Bob, use Cuckoo hashing to assign their items to bins, comparing only the items placed in the same bin by each party. However, this method faces a challenge: Alice might assign an item x to $h_0(x)$, while Bob assigns the same item to $h_1(x)$. A protocol that compares Alice's item in bin $h_0(x)$ to Bob's items in both $h_0(x)$ and $h_1(x)$ would reveal to Bob that Alice has an item corresponding to these two bins, compromising privacy. The solution, as implemented in the studies by [7, 22, 20], is to have Alice use Cuckoo hashing to bin her items, while Bob uses simple hashing, mapping each of his items to both bins $h_0(x)$ and $h_1(x)$. Bob then pads his bins to contain $b = O(\log M/\log\log M)$ items in each bin. This results in a total of $O(M \log M/\log\log M)$ comparisons.

6.2.3 2D Cuckoo Hashing

Pinkas et al. [21] propose a new variant of Cuckoo hashing called 2D Cuckoo Hashing that reduces the number of comparisons to $O(M)$ for 2P-PSI. The construction involves two tables, T_L and T_R, positioned on the left and right sides, respectively. Each table has a size of $4(1+\varepsilon)M$ and consists of two smaller subtables. Specifically, T_L is divided into T_{L0} and T_{L1}, while T_R is divided into T_{R0} and T_{R1}. The hash functions h_{L0}, h_{L1}, h_{R0}, and h_{R1} assign items to the corresponding subtables T_{L0}, T_{L1}, T_{R0}, and T_{R1}. The arrangement of these tables is illustrated in Figure 6.1.

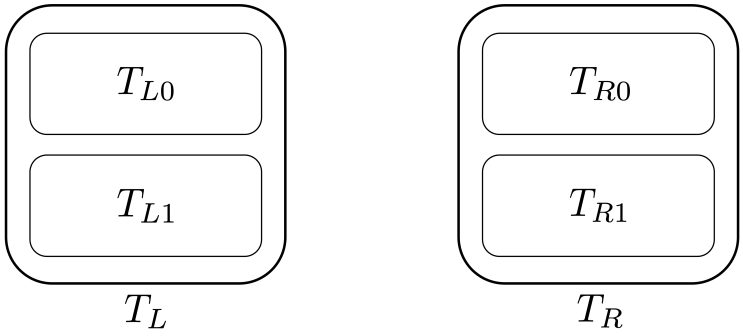


Figure 6.1: The tables T_L and T_R, and the subtables T_{L0},T_{L1} and T_{R0},T_{R1} [21].

Alice and Bob use different strategies to assign their items to the subtables. The strategies are as follows:

Alice maps each of her items to all the subtables on one side. Specifically, each item x belonging to Alice is either assigned to both bins $T_{L0}[h_{L0}(x)]$ and $T_{L1}[h_{L1}(x)]$ on the left side or to the bins $T_{R0}[h_{R0}(x)]$ and $T_{R1}[h_{R1}(x)]$ on the right side. This means that Alice places each item in all the subtables on a single side.

Bob, on the other hand, maps each of his items to one subtable on each side using standard Cuckoo hashing. Each item x from Bob is mapped to either $T_{L0}[h_{L0}(x)]$ or $T_{L1}[h_{L1}(x)]$ on the left side, and similarly to either $T_{R0}[h_{R0}(x)]$ or $T_{R1}[h_{R1}(x)]$ on the right side. In essence, Bob assigns each item to one subtable on both sides.

Figure 6.2 illustrates the possible hashing options for an item x for both parties. It is evident that if both parties possess the same item x, there will be exactly one subtable among T_{L0}, T_{L1}, T_{R0}, or T_{R1} where both Alice and Bob store x.

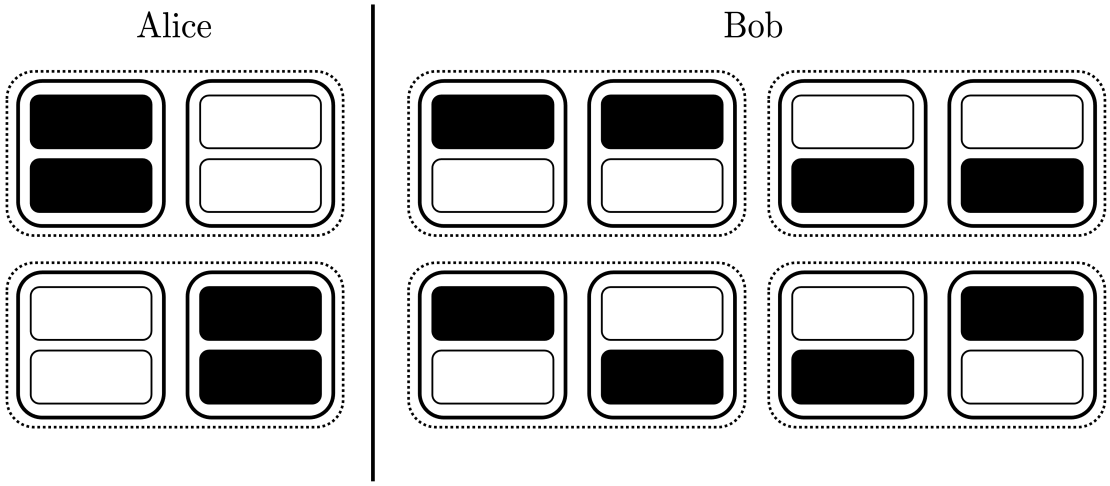


Figure 6.2: The possible combinations of locations to which Alice and Bob map their inputs [21]. Each dashed box represents a different combination.

Although both our hashing scheme and the 2D Cuckoo hashing achieve $O(M)$ complexity for 2P-PSI, our scheme is more general and can be applied to any number of participants and any threshold t. 2D Cuckoo hashing is specifically designed for 2P-PSI and does not generalize to more than two participants or a threshold t due to Alice and Bob's asymmetric insertion strategy. However, being this specific allows 2D Cuckoo hashing to achieve lower constants in concrete implementations.

6.2.4 Phasing: Permutation-based Hashing

Pinkas et al. [20] propose a new hashing technique to reduce the bit representations of the elements. This optimization is targeted toward circuit-based PSI protocols and, thus, is not directly applicable to our OT-MP-PSI protocol. However, if our hashing scheme is to be implemented in a circuit-based PSI protocol, this technique can be used to further reduce the computational complexity.

When using β bins, they map items by assigning the first $\log \beta$ bits of an item's hashed representation to specify the bin. The remaining bits are utilized in a way that effectively prevents collisions, which is done by permutation-based hashing. This technique reduces the bit-length required for values in the PSI protocol by $\log \beta$ bits, which can lead to an overhead decrease of 60%-75% in the settings they investigated.

6.3 Similar Problems

Here, we discuss some cryptographic problems and the existing solutions that are closely related to the OT-MP-PSI problem.

6.3.1 Threshold Private Set Intersection (TPSI)

Although very similarly named, the OT-MP-PSI and TPSI problems are very different. We can observe this similar naming caused some confusion in the existing literature: Mohanty et al. [17] incorrectly classify [13, 16, 2] as TPSI protocols and compare them with other TPSI protocols.

TPSI is a cryptographic problem where two or more participants holding sets of elements want to learn the intersection of their sets if the intersection size is above a certain threshold. Notice the difference between the threshold in TPSI and the threshold in OT-MP-PSI. In TPSI, the threshold is defined for the size of the intersection, while in OT-MP-PSI, the threshold is defined for the number of sets that contain an element.

Zhao and Chow [26] propose the first dedicated solution for TPSI. Before, a TPSI protocol could be achieved by running a circuit-based PSI protocol with a circuit that computes the intersection size. However, this would always leak the size of the intersection even when it is smaller than the threshold. Although Hallgren et al. [10] also introduced a TPSI protocol around the same time as [26], they have a computation complexity of $\mathcal{O}(M^2)$, which is worse than the garbled circuit approach. Zhao and Chow [26] introduce a new cryptographic primitive called encrypted PSI cardinality (ePSI-CA). It makes Alice compute an encrypted Bloom filter of its set with k hash functions and send it to Bob. Bob then sums the encrypted Bloom filter indices corresponding to its set elements using homomorphic addition. Then, for each set element, Bob creates a polynomial that will output 1 if the result of the summation is exactly k; if it is less than k, it will output 0, and he encrypts the polynomial coefficients. Bob sends the summations and the polynomials to Alice. Alice then decrypts the summations, obliviously evaluates the polynomials and sums the results using homomorphic addition. The result will be the intersection size encrypted with Bob's public key. They then use this primitive to construct a TPSI protocol.

Ghosh et al. [9] focus on the communication complexity of the TPSI protocols. They show that the lower bound of the communication complexity of TPSI is $\Omega(t)$, and they propose a protocol that achieves this lower bound. Their main building block is representing sets as monomials of a polynomial; in other words, they encode the set $S = \{a_1, \ldots, a_M\}$ as a polynomial $p(x) = \sum_{i=1}^{M} x^{a_i}$.

Badrinarayanan et al. [1] improve upon the work of [9] and generalize their TPSI protocol to the multi-party setting. They show that the theoretical lower bound of the communication complexity of the multi-party TPSI is $\Omega(tN)$, and they propose a protocol that achieves this lower bound.

6.3.2 Private Set Union (PSU)

Private Set Union (PSU) is a cryptographic problem where two or more parties, each holding a set of elements, want to learn the elements that exist at least in one of the sets without revealing any other information, such as the frequency of the elements or which sets contributed the elements. Frikken [8] proposes a solution for PSU using techniques very similar to Kissner and Song [13]. Roughly, their protocol works as follows: each participant creates polynomials representing their sets by making the set elements roots of the polynomials. Then, each participant sequentially multiplies their polynomial with an encrypted accumulator polynomial and publishes tuples $(\text{Enc}(xf(x)), \text{Enc}(f(x)))$ for each element x in their set where f is the accumulator polynomial. The tuples are shuffled in a way that will not leak their origin. Then they collectively decrypt the tuples; if a participant has an element x that was previously added to the accumulator polynomial, the tuple outputs $(0, 0)$; if not, it outputs $(xf(x), f(x))$, and they can calculate the x by $xf(x) \times f(x)^{-1}$. In this way, every element is revealed only once.

Davidson and Cid [5] introduce a new PSU protocol that is computationally more efficient than [8]. They use a similar idea but replace the polynomial representation with a Bloom filter.

Kolesnikov et al. [14] and Jia et al. [12] focus on two-party PSU. They develop more efficient protocols compared to [5, 8] using symmetric key encryption. Jia et al. [12] show that the binning technique used in [14] is leaking information. Their protocols solve this problem while maintaining the same efficiency.

Chapter 7

Conclusion

In this work, we prove that it is possible to perform non-interactive OT-MP-PSI with linear computation complexity in the data set size by constructing a protocol using our new hashing scheme. Our proposed protocol efficiently identifies elements that appear in at least t sets among N participants without revealing any information about elements that do not meet this threshold. Our primary use case is detecting multi-institution attacks, where timely and private information sharing is crucial to mitigate risks.

Existing solutions, such as those by Kissner & Song [13] and Mahdavi et al. [16], either require a high number of communication rounds or have computational complexities that render them impractical for real-life applications involving large datasets. Our proposed protocol addresses these issues with a computational complexity of $\mathcal{O}(t^2 M \binom{N}{t})$, making it suitable for environments with large and dynamic datasets, such as network security.

The key innovation in our protocol is the novel hashing scheme that significantly optimizes performance. Instead of hashing secret shares into bins, which requires padding and carries large constants that prevent scalability, our approach hashes shares into array indexes, with each index storing a single share. When multiple shares are hashed into the same index, a pseudo-random ordering selects one. This reduces the need for the aggregator to try combinations of shares, instead requiring just combinations of participants. By setting the size of each array to $t \times M$ and having each participant produce multiple arrays, we ensure that the probability of missing values is minimal, with a failure probability smaller than 2^{-40}.

We evaluated the security and performance of our OT-MP-PSI protocol under the semi-honest model, confirming its security through simulator construction for both participants and the aggregator.

Our protocol outperforms Mahdavi et al.'s [16] approach, showing at least two orders of magnitude faster reconstruction times and scaling effectively with the number of participants and set sizes. We also benchmark our protocol using real-world network data from the CANARIE IDS Program to demonstrate our protocol's efficiency and practicality. The results from real-world data tests show that the average reconstruction time is 170 seconds for 33 participants and a maximum set size of 144,045. These benchmarks validate our protocol's suitability for real-world applications with large datasets.

We also show that our protocol efficiently addresses the two-party private set intersection ($N = t = 2$) and the multi-party private set intersection ($N > 2, t = N$) problems by both theoretical analysis and experimental results. By bridging these two problems and the OT-MP-PSI problem with a single protocol, we offer a versatile solution that can be utilized across various applications and adjusted as requirements evolve.

In general, our protocol offers a robust, scalable, and privacy-preserving solution for the OT-MP-PSI problem. For future work, we aim to explore optimizations in choosing combinations of participants, as it is the main bottleneck in our protocol.

References

[1] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. In Juan A. Garay, editor, *Public-Key Cryptography – PKC 2021*, pages 349–379, Cham, 2021. Springer International Publishing.

[2] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. Practical multi-party private set intersection protocols. *IEEE Transactions on Information Forensics and Security*, 17:1–15, 2022.

[3] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of cryptology*, 14:101–119, 2001.

[4] Canarie. New funding to deliver intrusion detection systems to Canada's research and education sector, Jul 2021.

[5] Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy*, pages 261–278, Cham, 2017. Springer International Publishing.

[6] Alfonso De Gregorio. Cryptographic key reliable lifetimes: Bounding the risk of key exposure in the presence of faults. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 144–158. Springer, 2006.

[7] Michael J Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1):115–155, 2016.

[8] Keith Frikken. Privacy-preserving set union. In Jonathan Katz and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 237–252, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[9] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 3–29, Cham, 2019. Springer International Publishing.

[10] Per Hallgren, Claudio Orlandi, and Andrei Sabelfeld. PrivatePool: Privacy-preserving ridesharing. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 276–291, 2017.

[11] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 14–25, 2019.

[12] Yanxue Jia, Shi-Feng Sun, Hong-Sheng Zhou, Jiajun Du, and Dawu Gu. Shuffle-based private set union: Faster and more secure. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2947–2964, Boston, MA, August 2022. USENIX Association.

[13] Lea Kissner and Dawn Song. *Private and threshold set-intersection*. School of Computer Science, Carnegie Mellon University, 2004.

[14] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 636–666, Cham, 2019. Springer International Publishing.

[15] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[16] Rasoul Akhavan Mahdavi, Thomas Humphries, Bailey Kacsmar, Simeon Krastnikov, Nils Lukas, John A Premkumar, Masoumeh Shafieinejad, Simon Oya, Florian Kerschbaum, and Erik-Oliver Blass. Practical over-threshold multi-party private set intersection. In *Annual Computer Security Applications Conference*, pages 772–783, 2020.

[17] Tapaswini Mohanty, Vikas Srivastava, Sumit Kumar Debnath, Ashok Kumar Das, and Biplab Sikdar. Quantum secure threshold private set intersection protocol for IoT-enabled privacy-preserving ride-sharing application. *IEEE Internet of Things Journal*, 11(1):1761–1772, 2024.

[18] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009.

[19] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[20] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, Washington, D.C., August 2015. USENIX Association.

[21] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via Cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 125–157, Cham, 2018. Springer International Publishing.

[22] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 797–812, USA, 2014. USENIX Association.

[23] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[24] Jan Stanek, Alessandro Sorniotti, Elli Androulaki, and Lukas Kencl. A secure data deduplication scheme for cloud storage. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, pages 99–118, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[25] Saif Zabarah, Omar Naman, Mohammad A Salahuddin, Raouf Boutaba, and Samer Al-Kiswany. An approach for detecting multi-institution attacks. *Annals of Telecommunications*, pages 1–14, 2023.

[26] 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'18, page 54–65, New York, NY, USA, 2018. Association for Computing Machinery.