Practical Multi-Party Private Set Intersection Protocols

Bay, Asli; Erkin, Zekeriya; Hoepman, Jaap Henk; Samardjiska, Simona; Vos, Jelle

DOI
10.1109/TIFS.2021.3118879

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Information Forensics and Security

Practical Multi-Party Private Set Intersection Protocols

Aslı Bay, Zekeriya Erkin, *Senior Member, IEEE*, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos

***Abstract*— Privacy-preserving techniques for processing sets of information have attracted the research community's attention in recent years due to society's increasing dependency on the availability of data at any time. One of the fundamental problems in set operations is known as *Private Set Intersection* (PSI). The problem requires two parties to compute the intersection between their sets while preserving correctness and privacy. Although several efficient two-party PSI protocols already exist, protocols for PSI in the multi-party setting (MPSI) currently scale poorly with a growing number of parties, even though this applies to many real-life scenarios. This paper fills this gap by proposing two multi-party protocols based on Bloom filters and threshold homomorphic PKEs, which are secure in the semi-honest model. The first protocol is a multi-party PSI, whereas the second provides a more subtle functionality - *threshold* multi-party PSI (T-MPSI) - which outputs items of the server that appear in at least some number of other private sets. The protocols are inspired by the Davidson-Cid protocol based on Bloom filters. We compare our MPSI protocol against Kolesnikov *et al.*, which is among the fastest known MPSI protocols. Our MPSI protocol performs better than Kolesnikov *et al.* in terms of run time, given that the sets are small and there is a large number of parties. Our T-MPSI protocol performs better than other existing works: the computational and communication complexities are linear in the number of elements in the largest set given a fixed number of colluding parties. We conclude that our MPSI and T-MPSI protocols are practical solutions suitable for emerging use-case scenarios with many parties, where previous solutions did not scale well.**

***Index Terms*— Privacy-preserving protocols, PSI, MPSI, threshold MPSI, threshold PKE, Bloom filters.**

I. Introduction

MULTI-PARTY computation (MPC) has been an active research field for several decades, and it enables us to design functions with secret inputs from two or more parties. The research challenge is to design such functions efficiently in terms of several aspects such as run-time, bandwidth, and storage needed to perform such functions. Recently, MPC has received more attention from the industry due to the General Data Protection Regulation (GDPR), which was introduced in the European Union in May 2018.

In this paper, we focus on a specific MPC problem, namely Private Set Intersection (PSI), which aims at calculating the intersection of two or more sets without revealing the set items of the involved parties or in certain settings, even the set sizes. The problem in the two-party setting has attracted significant attention from the research community in the last decades [1]–[9]. Some of these protocols have been extended to the multi-party setting, commonly referred to as Multi-party Private Set Intersection (MPSI). A much less common type of protocol is Threshold MPSI (T-MPSI), which returns all elements that are in at least a threshold amount of sets. This type of protocol is also sometimes referred to as over-threshold or d-and-over intersection.

So far, MPSI solutions have been designed towards numerous applications: Online recommendation systems including dating sites [10], confidential data sharing such as security incident information [11], border protection against criminal attempts [12], comparison of no-fly lists [7], network security operations such as botnet detection and detecting intrusions by finding the sets' suspicious IPs [13]. Of course, these methods reach much further than only these examples: MPSI protocols can be used among several commercial companies to find the intersection of customer lists where each list, except the intersection, is protected. The list of common customers can be used to plan promotions for such customers [14]; MPSI can be used among the community of medical professionals to find out the patients of a hospital who has participated in the medical tests of different research labs [15]; MPSI can also be used in multi-party access control, where several co-owners of a common content each specify a set of users who are permitted to access data. The ones in the intersection are allowed to access the content [16]; MPSI can be employed among several enterprises which have private audit logs of connections to their corporate networks and are interested in identifying similar activities in all networks [17].

Most of these multi-party applications feature only a few parties t and generally large numbers of items n. Many works have therefore reasonably focused on time and bandwidth-efficient protocols in the case where $n \gg t$. However, these protocols typically translate poorly to a situation with many parties and few elements. Two exemplary and increasingly relevant applications are the following.

Identifying High-Risk Individuals in the Spread of Disease: If a disease spreads in an organization, it is important to identify those who form a high risk of exposure, particularly

Manuscript received April 2, 2021; revised July 31, 2021 and September 20, 2021; accepted September 21, 2021. Date of publication October 8, 2021; date of current version December 16, 2021. This work was supported by the H2020 EU Project Product Security for Cross Domain Reliable Dependable Automated Systems (SECREDAS) under Grant GA 783119. The associate editor coordinating the review of this manuscript and approving it for publication was Mr. Frederik Armknecht. *(Corresponding author: Aslı Bay.)*

Aslı Bay is with the Computer Engineering Department, Antalya Bilim University, 07190 Antalya, Turkey (e-mail: aslidarbuka@gmail.com).

Zekeriya Erkin and Jelle Vos are with the Department of Intelligent Systems, Delft University of Technology, 2628 CD Delft, The Netherlands.

Jaap-Henk Hoepman is with the Institute for Computing and Information Sciences, Radboud University, 6525 XZ Nijmegen, The Netherlands, and also with IT Law, University of Groningen, 9712 CP Groningen, The Netherlands.

Simona Samardjiska is with the Institute for Computing and Information Sciences, Radboud University, 6525 XZ Nijmegen, The Netherlands.

Digital Object Identifier 10.1109/TIFS.2021.3118879

if the disease is hard to detect otherwise. Through contact tracing, we can identify those at risk who have been in contact with someone infected. This process requires those at risk to share an exhaustive list of recent encounters. While this is an important practice, this list might reveal additional sensitive information such as visited locations and other specific behaviors. In situations where the collection of this information is impermissible, a (threshold) MPSI can be used to identify individuals that form a high risk while preserving the privacy of others. For example, by performing an MPSI with threshold $\mathcal{T}$ on a group of people's contact details, we can identify those individuals that have had contact with at least $\mathcal{T}$ other people, which means that these people run a higher risk of infection. The protocol only reveals those individuals that met many others, and it does not reveal any other information.

Criminal Activities on Smart Roads: In our EU project SECREDAS,[1] we aim at enabling autonomous vehicles on smart roads. Achieving this goal will have an enormous impact on changing transportation in general, including but not limited to reducing traffic jams and thus CO2 emission, better planning for building roads, and most importantly, improving safety. Given that smart roads and autonomous vehicles will heavily rely on several types of sensory data and computation, it is also mandatory to provide security and privacy while preserving functional safety and operational performance.

While security and privacy are essential requirements in the project, it is also essential to provide auditing functionality in case of urgency where criminal activities are involved. One such use case is detecting theft that happens at high-way parking areas (this use case was first addressed in [18]). Within a network of smart roads with parking areas, it is necessary to identify a particular vehicle or vehicles that appear in a number of parking areas. The ideal privacy-preserving solution for a vehicle tracking system would be hiding the identifiers of each car, thus providing privacy for the vehicle owners, but in case of detecting criminal activity, it should be possible to identify the vehicle that visits certain parking areas. More precisely, it is possible to use cameras and other sensors to detect vehicles in parking areas. What we are interested in is the vehicles visiting more than a threshold number of parking areas. Thus, our problem can be formalized as a T-MPSI problem.

Anonymous Voting and Consensus: In voting, we often encounter the situation where there are many voters t and few candidates n. Consider an approval voting scheme, where we let voters vote on multiple candidates to find which candidates are approved in consensus or which candidates pass at least $\mathcal{T}$ votes. In the simplest form, such a vote works by counting up all votes and releasing the final counts. However, a candidate might reason about voting behavior from these results and trouble those from whom they had expected a vote. Another example of a bad consequence of releasing all counts is that voters might be urged to change their behavior in future votes, dropping support for candidates close to surpassing the threshold, while they might not necessarily disapprove. Thus it is prudent that no other information than the set of approved candidates is revealed, but at the same time, the voters can be sure that the count was done securely. This can be achieved using an MPSI protocol: Every voter submits the set of candidates they approve of, and the final intersection represents the vote results. In the same way, a (threshold) MPSI protocol can be used for determining threshold-approval.

[1] https://www.ecsel.eu/projects/secredas

A. Related Work

PSI protocols have been thoroughly studied, and some works are already fast enough to compute the intersection of large sets in the order of seconds [1], [2]. There are also many other works that compute the private intersection of multiple sets (MPSI) [13], [17], [19]–[24]. However, related to our problem, namely the threshold MPSI (T-MPSI), existing work is limited [19], [21].

Several different techniques are used to design PSIs and MPSIs, such as oblivious transfer, permutation-based hashing, circuit-based computations, Bloom filters, cuckoo hashing, and oblivious programmable hashing.

Among those methods, the fastest PSI protocols are by Pinkas *et al.* [8], [9], where the former uses oblivious transfer and permutation-based hashing and the latter is based on a generic circuit-based multi-party computation.

One of the earliest works proposing an MPSI protocol is by Lai *et al.* [20]. However, this protocol leaks information to the involved parties about the contents of each party's set, as it sends part of each party's set encoded as a Bloom filter to other parties in plain text. Like a hash map, a Bloom filter consists of bins, although these are Booleans for Bloom filters. The parties compute the Bloom filter representing the intersection using a logical AND operation between the bins of each filter.

Miyaji and Nishida in [21], [22] propose a multi-party PSI protocol based on Bloom filters and an additively homomorphic encryption scheme. They also introduce a threshold PSI [21] but do not provide a security analysis. Their computational complexity is linear in the number of items in the largest data set and linear in the size of the Bloom filter.

Another work proposed by Many and Dimitropoulos [25] is based on *Counting Bloom Filters*. Similar work is done by Karapiperis *et al.* [26], in which they use a Count-Min sketch data structure. In this work, by using homomorphic operations and symmetric noise addition techniques, sketches are used to represent the elements of local data sets which are then intersected to provide a global synopsis [27], [28]. However, in both [25] and [26], the T-MPSI definition is different from ours. Namely, our protocol outputs the set of elements that appear in at least a threshold number of parties' data sets (including the server's), while their protocols also output the number of times such an element appeared in all data sets. For this reason, we do not compare them against our protocol.

Several PSI protocols use Cuckoo filters. Like Bloom filters, they are an approximate set representation, but in the context of PSI, they serve a fundamentally different purpose. While Bloom filters contain bits that can be combined using a logical `AND` operation to compute the Bloom filter representing the intersection, a Cuckoo filter cannot be combined in such a way.

Instead, such a filter contains bins that can contain multiple entries, and they are merely used as a binning technique to limit the number of pairwise comparisons. There exist other such binning techniques as well [29], [30]. The protocol by Pinkas *et al.* [30], for example, performs a private set inclusion protocol for each element in one party's set with regard to the others set to find the intersection. By first distributing the elements over bins, the problem is reduced to set inclusions between respective bins rather than interactions between all elements. Kolesnikov *et al.* [31] use a similar trick to reduce the number of oblivious PRF interactions.

In addition to [21], Kissner and Song in [19] also address the threshold PSI problem, using additively homomorphic encryption, equivocal commitments, mix-net, and a shuffling protocol. The proposed protocols, which are in the semi-honest and malicious models, have cubic and quartic complexity in the number of sets.

A radically different approach was presented by Kolesnikov *et al.* [17], which uses a novel primitive based on oblivious transfer to evaluate whether all parties have a specific item in their sets. The authors also implemented this scheme, and they showed that it is highly time-efficient for large amounts of items. However, the time complexity scales quadratically in the number of parties. Another scheme by Inbar *et al.* [13] is also based on Oblivious Transfer and aims to evaluate a Garbled Bloom Filter. This scheme was also implemented, but the implementation is not readily available. Interestingly, the authors also propose a scheme proven secure in the *augmented* semi-honest model where communication scales logarithmically in a hyper-cube optimization.

There is also a recent paper by Badrinarayanan *et al.* [32] which provides threshold MPSIs which have sublinear communication complexities with a threshold number T that can be sublinear in the number of elements in data sets (namely, n). However, their threshold MPSI functionalities are completely different than ours. Namely, for t parties P_1, P_2, ..., P_t each of which holds a private data set S_i of size n, their first threshold functionality allows the parties to learn the intersection $S = \cap_{i=1}^{t} S_i$ only if $|S| \geq n - T$, while the second functionality provides intersection if $|\cup_{i=1}^{t} S_i - S| \leq T$. Neither of the two matches our threshold definition so their work is not comparable to ours.

Finally, a line of interesting research, although not directly comparable to the other works we discussed, uses ε-differential privacy [33] to achieve various operations while providing some level of privacy. In general, differential privacy cannot offer a provable guarantee of privacy and is always accompanied by a trade-off between accuracy and security. Still, it is widely used in practice, for example, by both Google and Apple to aggregate and analyze user's data [34], [35]. While usually not a strictly PSI technique, it can be used for finding (approximate) intersections.[2] One dedicated PSI algorithm using differential privacy is the one by Xue *et al.* [36]. It is an MPSI protocol where each client locally perturbs their data set using a randomized mechanism that satisfies ε-differential privacy and sends their data set in this new form to the server. Then the server computes the set intersection privately. While the protocol has linear complexity in the number of parties, it also scales with the size of the universal set, which is typically significantly larger than the size of a party's set.

[2]We have included Google's general RAPPOR algorithm [34] in Table I, but note that it is not optimized for PSI.

B. Our Contribution

We consider a scenario where many (t) parties - each with their own small set of n items - want to compute the intersection of all sets through communication with a single leader – that we call *server* (as opposed to pairwise communication). Secondly, we consider a scenario where many parties want to compute the collection of items that a leader shares with at least ℓ parties. In both cases, no party may learn the other parties' sets.

In particular, we propose two protocols in the semi-honest security model – namely an MPSI protocol inspired by the PSI of Davidson and Cid [3] that functions in a star topology and solves the problem from the first scenario; and an efficient threshold MPSI (T-MPSI) that solves the problem from the second scenario. We use Bloom filters, which are efficient probabilistic data structures for representing sets in a finite space. Note that Bloom filters have been used before for PSI in [3]–[7] and also for computing MPSIs in [13], [21], [22]. For the construction of our T-MPSI, we further use a secure comparison protocol (SCP) as a sub-protocol. Here we describe and use in our implementation a modification of the Kerschbaum *et al.* SCP [37]. The modification is necessary to prevent leakage of private data. Note that it is possible to use a different SCP as long as it satisfies our security requirements.

Our MPSI scheme bears some similarities with the MPSI from [21], [22]: However, their protocol is not secure against collusion that involves the dealer. Our proposed MPSI protocol circumvents this issue by letting each party randomize individually. Furthermore, our scheme is significantly more efficient by at least a factor of λ - the statistical security parameter - and has the advantage that the functioning of the protocol does not depend on an outside party.[3]

Our contributions can be summarized as follows:

- An MPSI protocol that scales linearly with the number of parties. The protocol improves the state of the art significantly for large numbers of (corruptible) parties, for instance, when there are 10 or more parties with sets of around 64 elements.
- A T-MPSI protocol that scales quadratically with the number of parties. The protocol stays within 1 minute of run-time when 50 parties have 4 elements in their sets.
- To the best of our knowledge, among the MPSIs based on public-key techniques, our MPSI protocol requires the least effort in both computation and communication. Furthermore, our complexity analysis shows that the T-MPSI we propose is the most efficient among existing

[3]On the other hand, our T-MPSI is quite different from the one in [21]: in our construction, we use a secure comparison protocol as a sub-protocol which is not the case of the T-MPSI in [21], where again the outsourced dealer must not collude. Our efficiency is again better both in computation and communication (see Table I), and we provide a full simulation-based proof, while the T-MPSI in [21] completely lacks a security analysis.

TABLE I

THE COMPARISON OF PREVIOUS DESIGNS WITH OURS IN THE SEMI-HONEST SETTING. n IS THE NUMBER OF ELEMENTS IN A DATA SET; t IS NUMBER OF PARTIES; u IS THE SIZE OF THE UNIVERSAL SET; ℓ IS THE *threshold* OF HOMOMORPHIC PKE; κ IS THE COMPUTATIONAL SECURITY PARAMETER; λ IS THE STATISTICAL SECURITY PARAMETER; AND $\log|X|$ IS THE SIZE OF THE CIPHERTEXT X IN BITS. THE MERGED COLUMNS REFER TO A PROTOCOL'S TOTAL COMPLEXITY, WHILE SEPARATE SERVER AND CLIENT COLUMNS EXPRESS THE ISOLATED COMPLEXITIES OF THE SERVER AND A CLIENT. *, **–THESE SOLUTIONS ARE BASED ON DIFFERENTIAL PRIVACY, WHILE ALL THE OTHER SOLUTIONS ARE CRYPTOGRAPHIC. ** IS NOT OPTIMIZED FOR PSI BUT IS A GENERAL PRIVACY PRESERVING CROWDSOURCING STATISTICS TECHNOLOGY

Protocol	Communication (bits)		Computation (operations)		Number of Rounds	MPSI	T-MPSI
	Server	Client	Server	Client			
[19]	$\mathcal{O}(\lambda\ell tn\log\|X\|)$		$\mathcal{O}(\ell tn^2)$		$\mathcal{O}(t)$	✓	–
[19]	$\mathcal{O}(\lambda t^3 n)$		Not given		$\mathcal{O}(t)$	–	✓
[23]	$\mathcal{O}(\lambda t^2 n)$		$\mathcal{O}(tn)$		$\mathcal{O}(1)$	✓	–
[36]*	$\mathcal{O}(tu)$		$\mathcal{O}(tu)$		$\mathcal{O}(1)$	✓	–
[34]**	$\mathcal{O}(\lambda tn^2)$	$\mathcal{O}(\lambda n^2)$	–	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(1)$	–	–
[21],[22]	$\mathcal{O}(\lambda tn\log\|X\|)$	$\mathcal{O}(\lambda tn\log\|X\|)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(1)$	✓	–
[21]	$\mathcal{O}(\lambda t^2 n\log\|X\|)$	$\mathcal{O}(t^2 n\log\|X\|)$	$\mathcal{O}(\lambda tn\log\|X\|)$	$\mathcal{O}(\lambda tn)$	$\mathcal{O}(1)$	–	✓
[24]	$\mathcal{O}(tn\lambda)$	$\mathcal{O}(n\lambda)$	$\mathcal{O}(tn\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	✓	–
[17]	$\mathcal{O}(tn\lambda)$	$\mathcal{O}(n\ell\lambda)$	$\mathcal{O}(t\kappa)$	$\mathcal{O}(\ell\kappa)$	$\mathcal{O}(1)$	✓	–
[13]	$\mathcal{O}(tn\kappa k)$	$\mathcal{O}(tn\kappa k)$	$\mathcal{O}(tn\kappa k)$	$\mathcal{O}(tn\kappa k)$	$\mathcal{O}(\log t)$	✓	–
Sect. III	$\mathcal{O}(n\ell\log\|X\|)$	$\mathcal{O}(\lambda n\log\|X\|)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(1)$	✓	–
Sect. IV	$\mathcal{O}(nt\ell\log\|X\|)$	$\mathcal{O}(\max(\lambda,t)n\log\|X\|)$	$\mathcal{O}(nt)$	$\mathcal{O}(\max(\lambda,t)n)$	$\mathcal{O}(t)$	–	✓

ones: the computational and communication complexities are linear in the number of items in the largest data set. For the comparison of the existing MPSI and T-MPSI, we refer the reader to Table I, where we give separate complexities for the server and clients.[4]

- An implementation of both of our protocols in C++.[5] It is to our knowledge the first T-MPSI protocol to be implemented and open-sourced.
- Both protocols require no trusted dealer but only a server that can be corrupted and can be freely chosen from the involved parties.
- Both protocols are secure in the semi-honest model, meaning that no party learns anything about the other parties' sets beyond their size. We provide a formal simulation-based security proof for both of the protocols to justify our security and privacy claims.

The rest of the paper is organized as follows: In Section II, we give the necessary preliminaries and notations. In Section III, our new MPSI protocol is introduced with its security proof and complexity analysis. Later in Section IV, we describe our T-MPSI and provide its security proof and complexity analysis. Afterwards, we compare our implementation to the state of the art in Section V, and we conclude our paper in Section VI.

II. PRELIMINARIES AND NOTATIONS

Throughout the paper we use the following notations:

$\mathbb{Z}_N$: the ring of residue classes modulo N.

When N is a prime p, it is denoted by $\mathbb{Z}_p$.

$\mathbb{Z}_p[X]$: the ring of polynomials with coefficients from $\mathbb{Z}_p$.

t : the number of parties.

u : the cardinality of the Universal set.

P_i : i-th party – a client for $i \in \{1,\ldots,t-1\}$ and a server for $i = t$.

S_i: a private data set of i-th party.

n_i: the size of S_i.

n: the size of the biggest data set among all, $n = \max\{n_1, n_2, \ldots, n_t\}$.

ℓ: a *threshold* of the homomorphic PKE.

PSI: private set intersection.

MPSI: multi-party private set intersection.

T-MPSI: threshold multi-party private set intersection.

κ: the computational security parameter.

λ: the statistical security parameter.

$\log|X|$: the size of the ciphertext X in bits.

$\mathcal{S}$: the intersection of S_i's, $\mathcal{S} = \cap_{i=1}^{t} S_i$.

$\mathcal{S}_{\mathcal{T}}$: $\mathcal{T}$-threshold intersection of S_t with the S_i's, $S_{\mathcal{T}} = \{x \in S_t | \exists S_{i_1}, \ldots, S_{i_{\mathcal{T}}}, x \in \bigcap_{j=1}^{\mathcal{T}} S_{i_j}\}$

h_i : hash function $h_i : \{0,1\}^* \to \{0,1\}^m$, $1 \le i \le k$.

$(\mathsf{pk}, \mathsf{sk})$: a pair of public and private key.

sk_i: a secret share of sk among t participants.

$\mathsf{Enc}(\mathsf{pk}, M)$ and $\mathsf{Dec}(\mathsf{sk}, C)$ (shortly $\mathsf{Enc}(M)$ and $\mathsf{Dec}(C)$): generic public-key encryption of M and decryption of C, respectively.

$\mathbf{BF_i}$: a Bloom filter of size m on the set S_i.

$\mathbf{IBF}_i$: an inverted Bloom filter of size m on the set S_i.

EBF and **EIBF**: entry-wise encryption of a Bloom filter and an inverted Bloom filter, respectively.

'$+_H$': homomorphic operation over ciphertexts.

$\mathcal{I}$: a set $\mathcal{I} = \{i_1, \ldots, i_\eta\} \subset \{1, 2, \ldots, t-1\}$.

$\bar{\mathcal{I}}$: the complement of $\mathcal{I}$ with respect to $\{1, 2, \ldots, t-1\}$.

Inp_i : the inputs of the clients where $\mathsf{Inp}_i = (S_i, |S_j|_{j=1}^t, \mathsf{pk}, \mathsf{sk}_i)$, $i \in \{1, \ldots, t-1\}$.

Inp_t: the input of server, where $\mathsf{Inp}_t = (S_t, |S_j|_{j=1}^t, \mathsf{pk})$.

A. Bloom Filters

Bloom filters were introduced by Bloom [38] as an efficient representation of data sets.

[4]For [21], the given complexities of the server are for the outsourced dealer.

[5]The source code for the implementation can be found at *https://github.com/jellevos/threshold-multiparty-psi*

A Bloom Filter, $\mathbf{BF} = (\mathbf{BF}[0], \ldots, \mathbf{BF}[j], \ldots, \mathbf{BF}[m-1])$ of length m encodes a set S of length at most n into m-bit string by means of randomly chosen k hash function, $(h_1, h_2, \ldots, h_k)$, where $h_i : \{0,1\}^* \rightarrow [0, 1, \ldots, m-1]$. In order to insert S into a Bloom filter, this is first done by initializing all indices to 0, and then for every $x \in S$, we set the indices $h_1(x), h_2(x), \ldots, h_k(x)$ to 1. If the index is already 1, we do nothing. Any party can verify whether an element is stored in a Bloom filter or not by a simple checking procedure. Bloom filters do not have a false negative error as they are deterministic and when an element is represented in a Bloom filter, in the query phase, their indices are all one. However, they yield a false positive error as the values $\mathbf{BF}_i[h_i(x)]$ might be all one for every $i \in \{1, \ldots, k\}$. As shown in [39], the probability that a particular bit in the Bloom filter is 1 is $p = 1-(1-1/m)^{kn}$. The upper bound of the false positive probability is $\varepsilon = p^k \times \left(1+\mathcal{O}\left(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}\right)\right)$, which is negligible in the number of hash functions k. In order to cap ε in a specific rate, the lower bound of m is $m \geq n \log_2 e \cdot \log_2 \frac{1}{\varepsilon}$, where e is the base of natural algorithm and the number of hash functions is $k = \frac{m}{n} \ln 2$. When substituting the minimal m, we get the number of hash functions as $k = \log_2 1/\varepsilon$. For example, when $\varepsilon = 2^{-50}$, then we need $k = 50$ hash functions. We make use of inverted and encrypted Bloom filters defined in [3] as follows:

Inverted Bloom Filter: For a given Bloom filter $\mathbf{BF}_i$ of S_i of a client i, the corresponding inverted Bloom filter $\mathbf{IBF}_i$ is defined as $\mathbf{IBF}_i[j] = \mathbf{BF}_i[j] + 1 \mod 2$.

Encrypted Bloom Filter: Let $\mathbf{BF}_i$ be the Bloom filter of a data set S_i of a client i, the corresponding encrypted Bloom filter is $\mathbf{EBF}_i[j] = \mathsf{Enc}_{\mathsf{pk}}(\mathbf{BF}_i[j])$, where pk is a public key of a secret key sk.

B. Security Definitions

We will need the following standard security definitions.

A function μ is called negligible (in k) if for every positive polynomial p, and sufficiently large k it holds that $\mu(k) < 1/p(k)$. We denote negligible functions by $negl(k)$.

Definition 1 (Computational Indistinguishability): We say that two distribution ensembles $X = \{X_k\}_{k \in \mathbf{N}}$ and $Y = \{Y_k\}_{k \in \mathbf{N}}$ indexed by a security parameter k are computationally indistinguishable if for any probabilistic polynomial time algorithm $\mathcal{A}$:

$$|Pr\,[1 \leftarrow \mathcal{A}\,(X_k)] - Pr\,[1 \leftarrow \mathcal{A}\,(Y_k)]| = negl(k)\,.$$

We will denote this by $X \approx_c Y$.

We prove the security of our protocols in the *semi-honest model* with *static adversaries* where all protocol participants are assumed to run in probabilistic polynomial time. We have:

- *Adversaries* - can be any subset of the protocol participants. We will typically refer to them as the *corrupted parties*. As usual, we do not consider outside adversaries, because they can be mitigated by standard network security techniques.
- *Static adversaries* - the set of corrupted parties is determined before the execution of the protocol and does not change during the execution. This means that the honest parties cannot become corrupted and reveal their secret values during the execution of the protocol.
- *Semi-honest model* - the corrupted parties follow the protocol honestly: they cooperate collude) to infer information about the honest parties, but do not exhibit active malicious behavior and do not deviate from the protocol. This is in contrast to the malicious model where the adversaries can behave in unpredictable ways and modify and suppress messages or leave the protocol execution altogether. While the semi-honest model offers weaker security than in the malicious model, it is appropriate in our privacy-preserving setting, since it guarantees that there is no unintentional leakage of private information. We emphasize that with appropriate extensions, the protocols can be turned into ones secure in the malicious model, but much less efficient.

We use the following definition to define security against semi-honest adversaries.

Definition 2 (Semi-Honest Security for Deterministic Functionalities [40]):

Let $f : (\{0,1\}^*)^t \rightarrow (\{0,1\}^*)^t$ be a deterministic functionality of a t-party protocol Π, where $f_i(x_1, \ldots, x_t)$ is the i-th component of f. Let $f_{\mathcal{I}}(x_1, \ldots, x_t)$ denote the sub-sequence of $f_{i_1}(x_1, \ldots, x_t), \ldots, f_{i_t}(x_1, \ldots, x_t)$, for $\mathcal{I} = \{i_1, \ldots, i_\ell\} \subset \{1, 2, \ldots, t\}$,. We assume that t-party protocol computes f. Then, the view of i-th party during an execution of the protocol Π on input $x = (x_1, \ldots, x_t)$ is denoted by $\mathrm{VIEW}_I^{\Pi}(x) = (I, \mathrm{VIEW}_{i_1}^{\Pi}, \ldots, \mathrm{VIEW}_{i_t}^{\Pi})$. We say that Π *privately computes* f, if there exists a polynomial-time algorithm denoted by a simulator $\mathcal{S}$ such that for every $\mathcal{I}$:

$$\{\mathcal{S}(\mathcal{I}, (x_{i_1}, \ldots, x_{i_\ell}), f_{\mathcal{I}}(x))\}_{x \in (\{0,1\}^*)^t} \approx_c \{\mathrm{VIEW}_{\mathcal{I}}^{\Pi}(x)\}_{x \in (\{0,1\}^*)^t}.$$

This security definition states that the view of each party in $\mathcal{I}$ can be simulated by only looking at their inputs and outputs.

Remark 1: Note that if we are dealing with probabilistic functionalities, the output of the protocol during the execution corresponding to $\mathrm{VIEW}_{\mathcal{I}}^{\Pi}(x)$ need not be the same as $f(x)$. Therefore, these need to be appended to the view of the semi-honest coalition of parties in $\mathcal{I}$ and to the simulated view, respectively.

Definition 3 (Public Key Encryption (PKE)): A public key encryption scheme $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ consists of three algorithms that for security parameter $k \in \mathbb{N}$ and two finite sets $\mathcal{M}, \mathcal{R} \subseteq \{0,1\}^*$ are defined as follows:

- The key-generation algorithm KGen is a probabilistic algorithm that on input 1^k and a random string outputs a public and private key pair $(\mathsf{pk}, \mathsf{sk})$.
- The encryption algorithm Enc is a probabilistic algorithm that on input public key pk, message $M \in \mathcal{M}$ and random $r \in \mathcal{R}$ outputs $C = \mathsf{Enc}(\mathsf{pk}, M, r)$ as the ciphertext.
- The decryption algorithm Dec is a deterministic algorithm that takes as input a secret key sk and ciphertext C, and outputs either a message $M' = \mathsf{Dec}(\mathsf{sk}, C) \in \mathcal{M}$ or a symbol $\perp \notin \mathcal{M}$ to indicate an invalid ciphertext.

Note that it is usually assumed that there is a trusted third party who sets up all public parameters A standard security notion for PKE is indistinguishability under chosen plaintext attack captured in the following definition.

Definition 4 (IND-CPA Security): A PKE scheme Π is called IND-CPA-secure if any PPT algorithm $\mathcal{A}$ has only negligible advantage in the following $\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{\Pi(1^k)}(\mathcal{A})$ experiment:

Experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{\Pi(1^k)}(\mathcal{A})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}()$
$(M_0^*, M_1^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Dec}}}(\mathsf{pk})$
$b \xleftarrow{\$} \{0, 1\}$
$C \leftarrow \mathsf{Enc}(\mathsf{pk}, M_b^*)$
Return 1 iff $b = b'$ otherwise 0, i.e.

$$Adv = Pr\left[\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{\Pi(1^k)}(\mathcal{A}) = 1\right] - 1/2 = negl(k)\,.$$

Informally, a public key encryption scheme is additively homomorphic, if given two ciphertexts $c_1 = \mathsf{Enc}(\mathsf{pk}, M_1)$ and $c_2 = \mathsf{Enc}(\mathsf{pk}, M_2)$ one can efficiently compute $\mathsf{Enc}(\mathsf{pk}, M_1 + M_2)$ without the knowledge of the secret key. More formally, we have the following definition.

Definition 5 (Additive Homomorphism): A PKE scheme Π with message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ is additively homomorphic if for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}()$, all $M_1, M_2 \in \mathcal{M}$ and arbitrary scalar α, there exists an efficient homomorphic operation $+_H$ over $\mathcal{C}$, such that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, M_1) +_H \mathsf{Enc}(\mathsf{pk}, M_2)) = M_1 + M_2 \text{ and}$$
$$\mathsf{Dec}(\mathsf{sk}, \alpha\mathsf{Enc}(\mathsf{pk}, M_1)) = \alpha M_1.$$

A useful algorithm for an additively homomorphic scheme is a ReRand algorithm, that allows anyone knowing the public key to rerandomize the ciphertext.

Definition 6: Given pk and a ciphertext $C = \mathsf{Enc}(\mathsf{pk}, M)$, encrypted using additively homomorphic PKE scheme, the algorithm ReRand is defined as $\mathsf{ReRand}(C) = C +_H \mathsf{Enc}(\mathsf{pk}, 0)$, i.e. an encryption of 0 is added to the ciphertext.

Note that rerandomization does not change the plaintext of C. On the other hand, rerandomization refreshes the ciphertext, and this is necessary in order to assure that the randomness in the final ciphertext is not related to the randomness used to produce the initial ciphertext. Typically, one needs to refresh/rerandomize the ciphertext after every homomorphic operation.

Threshold Public Key Encryption Often it is desirable that the secret key sk of a PKE is distributed among several parties where each party holds a share of the key sk_i, and decryption is only possible if enough parties (ℓ out of t) combine their decrypted shares of the message. In this setting, the KGen algorithm generates several secret keys sk_i instead of one; the encryption algorithm Enc works the same as in a regular PKE; The decryption is realized through two separate algorithms - a share decryption algorithm ShDec with which ℓ involved parties holding a secret key sk_i produce their decryption share C_i, and a combining algorithm Comb which using only the public key and the ℓ decryption shares outputs a message M' (or $\perp$ to indicate invalid). We will denote such a scheme as (ℓ, t) TPKE. Note that parties in such a scheme can either generate these keys using a suitable distributed key generation protocol [41], or they can trust a third party to do so.

Following the approach of [24] we require an extension of the decryption algorithm of an additively homomorphic threshold PKE that allows the involved parties to learn whether a ciphertext is encryption of zero or not, but nothing else. We will call such decryption algorithm "Decryption-to-zero" and denote it by $\mathsf{ShDec0}$. This property can typically be achieved by randomizing the ciphertext by each of the involved parties, combining the results in a new value, and jointly decrypting this obtained value. The result is that the decrypted value is randomized if it was different from 0 and nothing can be learned from it, apart from not being zero.

C. Additivelly Homomorphic Threshold PKE Schemes

Our protocols require the existence of an additively homomorphic threshold PKE that additionally has a decryption-to-zero variant defined. The threshold ElGamal PKE [42] and the threshold Pailler PKE [43] are two well-known examples that satisfy these properties. Here we briefly explain the latter.

Threshold Paillier PKE [43]. A (ℓ, t)-threshold version of the Paillier's scheme is briefly described as follows:

Key Generation: Generate two primes p and q such that $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are two other primes and different from p and q, and $\gcd(N, \phi(N))$. Then set $N = pq$ and $M = p'q'$. Let β be an element randomly chosen from $\mathbb{Z}_N^*$. Also, pick $(a, b) \in \mathbb{Z}_N^* \times \mathbb{Z}_N^*$ and set $g = (1+N)^a b^N \mod N^2$. The secret key $\mathsf{sk} = (\beta \times M)$ is shared as follows: Let $a_0 = \beta M$ and generate the polynomial $f(X) = \sum_{i=0}^{\ell} a_i X^i$ where $a_i \in_R \{0, \ldots, NM - 1\}$. The private key share of the i-th party is $\mathsf{sk}_i = f(i) \mod NM$ for t number of parties. The public key is $\mathsf{pk} = (g, N, \theta = aM\beta \mod N)$.

Encryption: Pick a random $r \in \mathbb{Z}_N^*$ and encrypt the message X by $C = g^X r^N \mod N^2$.

Share Decryption: The i-th party computes $C_i = C^{2\Delta \mathsf{sk}_i} \mod N^2$, where C is the ciphertext and $\Delta = t!$.

Combining: Let S be a subset of ℓ different C_i-s (of the ℓ involved parties). Combine the elements of S as follows: $X = L(\prod_{i \in S} C_i^{2\lambda_{0,i}^S} \mod N^2) \times \frac{1}{4\Delta^2\theta} \mod N$, where $\lambda_{0,i}^S = \Delta \prod_{i' \in S \setminus \{i\}} \frac{i'}{i'-i} \in \mathbb{Z}$ and $L(u) = \frac{u-1}{N}$.

A **Decryption-to-zero** variant can be realized as follows: Each party raises the ciphertext C to a nonzero random power as C^{r_i} and the results of all parties are multiplied together to obtain $C^{\Sigma r_i} = g^{X(\Sigma r_i)}(r^{\Sigma r_i})^N \mod N^2$. The obtained result is then jointly decrypted by the involved parties. If the result of the decryption is 0, they conclude that $X = 0$. Otherwise, they see a random value coming from the randomization of the ciphertext, from which nothing can be inferred about the plaintext.

D. Davidson-Cid Two-Party Private Set Intersection Protocol

Our work builds upon Davidson-Cid's two-party Private Set Intersection (PSI) protocol proposed in [3]. In the protocol both parties P_1 and P_2 are given k hash functions $h_1, \ldots, h_k$, P_1 has a pair of public pk (also available to P_2) and private

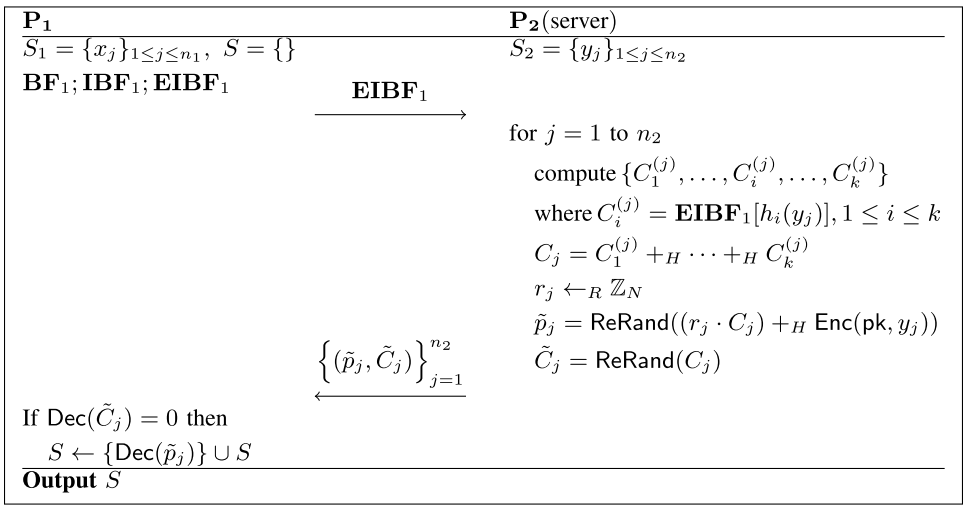

Fig. 1. Davidson-Cid two-party PSI [3].

key sk of an IND-CPA additively homomorphic scheme. They want to jointly find the intersection of their sets S_1 and S_2, in such a way that P_1 learns the output, but no other information is leaked to any of the parties apart from the sizes of the sets, that is given as an input to both parties.

In essence, in the protocol P_1 encrypts the inverted Bloom filter $\mathbf{IBF}_1$ that represents the set S_1 and sends it to P_2. The other party P_2, based on it's own set computes cumulative randomized values over the ciphertexts, without learning anything. This is made possible by the additive homomorphism property of the encryption scheme. Finally P_1 decrypts these values which decrypt to 0 if the corresponding element of P_2's set is in the intersection. Otherwise, the result is a random value. We outline the protocol in Fig. 1.

Clearly, when $y_j \in S_1$, C_j is encrypted to zero, so P_1 will correctly identify that y_j needs to be added to the intersection. Otherwise (when $y_j \notin S_1$), the decryption of $\tilde{p}_j$ will be a random number and will not disclose y_j. This correctness is probabilistic due to the false positive rate of Bloom filters, however it is negligible in k as explained in Section II-A.

E. Multi-Party Secure Comparison

In our threshold MPSI we make use of a multi-party secure comparison protocol (SCP). Such a protocol compares two values x_0 and x_1 in a multi-party setting, with the objective that no party learns anything about x_0 and x_1 during the protocol execution. For many applications, it is desirable that the output of the protocol is also private as is in our case as well.

To the best of our knowledge, the problem of secure comparison is actually the first ever MPC problem considered in the literature known under the name of the millionaire's problem [44]. But despite a broad range of applications of the two-party setting and numerous solutions proposed, we are not aware of any multi-party solution in the literature but the one of Kerschbaum *et al.* [37] that we describe shortly.

We use a slight modification of this protocol that addresses a minor security issue, and we evaluate the performance of our protocol including it as a sub-protocol. Notably, this means that our T-MPSI inherits a linear round complexity, which is not an artifact of the rest of our protocol. Since our protocol can use any secure multi-party SCP, this potentially allows an improvement in the round complexity if a constant round SCP is designed in the future.

Kerschbaum *et al.* Secure Comparison Protocol [37]. The protocol securely compares two values x_0 and x_1 given only their encrypted values $\mathsf{Enc}(x_0)$ and $\mathsf{Enc}(x_1)$ as input. The output is a single encrypted bit $\mathsf{Enc}(b)$ that determines whether $x_0 \le x_1$. In their protocol, $\mathbb{Z}_p$ (where p is prime) is represented by the upper half of the range $[0, p-1]$ as negative, that is $[\lceil \frac{p}{2} \rceil, p-1] \equiv [\lfloor -\frac{p}{2} \rfloor, -1]$. The objective is to compute whether $d = x_0 - x_1$ is positive or not by hiding d through multiplication with a large random number r, i.e., $d \ll r$. Also, in order to avoid obtaining a factor of d, they compute $rd - r'$ where r' is random number which is smaller than r. Let there be t parties such that they receive $\mathsf{Enc}(x_0)$ and $\mathsf{Enc}(x_1)$ as input and run the following protocol:

- P_1 computes $(a_1^1, a_2^1, a_3^1) = (\mathsf{Enc}(1), \mathsf{Enc}(0), \mathsf{Enc}(c))$ where,

$$\begin{aligned}\mathsf{Enc}(c) &= (\mathsf{Enc}(x_0)\mathsf{Enc}(-x_1))^{r_1}\mathsf{Enc}(-r_1') \\ &= \mathsf{Enc}(r_1(x_0 - x_1) - r_1'),\end{aligned}$$

  by picking two prime numbers r_1 and r_1' as explained above. To compute $\mathsf{Enc}(-x_1)$, P_1 needs to compute the multiplicative inverse of $\mathsf{Enc}(x_1)$. P_1 sends (a_1^1, a_2^1, a_3^1) to P_2.
- For every party P_i, $2 \le i \le t$, P_i selects $r_i' < r_i$ and flips a coin $b_i \in \{0, 1\}$, then computes (a_1^i, a_2^i, a_3^i) in the following way:
  If $b_i = 0$, then

  $$a_j^i = a_j^{i-1}\mathsf{Enc}(0),\ j \in \{1, 2\};\quad a_3^i = (a_3^{i-1})^{r_i}\mathsf{Enc}(-r_i')$$

  If $b_i = 1$, then

  $$a_j^i = a_{3-j}^{i-1}\mathsf{Enc}(0),\ j \in \{1, 2\};\quad a_3^i = (a_3^{i-1})^{-r_i}\mathsf{Enc}(r_i')$$

  P_i then sends (a_1^i, a_2^i, a_3^i) to $P_{i+1 \pmod t}$ (P_t sends to P_1).

- All parties P_i, $1 \leq i \leq t$, jointly decrypt a_3^t to decide the result. If $\mathsf{Dec}(a_3^t) < 0$, then a_1^t is the ciphertext of 1 (i.e. $\mathsf{Enc}(1)$), that is $[x_0 \leq x_1] = 1$, else $a_1^t = \mathsf{Enc}(0)$.

For the correctness of the protocol, we refer the reader to [37].

We would like to emphasize that this protocol suffers from information leakage. In fact, the parameters r_1 and r_2 in $r_1(x_0 - x_1) + r_1'$ cannot securely protect the relation between x_0 and x_1 from an adversary. The reason is that r_1 is not large enough for multiplicative masking. To address this security issue, we can switch to additive masking by choosing a large r_1' at the computation cost of performing a homomorphic exponentiation. The bit length of r_1' should be not less than $\ell + \kappa + 1$, where ℓ is the bit length of the x_0 and x_1, and κ is a security parameter. The security proof for this can be found in [45].

For t parties, the asymptotic round complexity of this protocol is $\mathcal{O}(t)$. To compute c for all the parties, there are t rounds and one extra round to decrypt c (a_3^t). Hence, the communicational and computational complexities are $\mathcal{O}(t)$ ciphertexts and $\mathcal{O}(t)$ homomorphic encryptions respectively.

III. Our Multi-Party Private Set Intersection Protocol

Our new multi-party private set intersection protocol is inspired by the Davidson-Cid two-party PSI (see Section II-D). In order to generalize their technique to the multi-party case, we need to slightly change the setting. Let $P_1, \ldots, P_t$ be the parties involved in the protocol, where $t \geq 3$. Each party P_i holds a private data set S_i of n_i elements. The goal of the protocol is to securely compute the intersection $S = \bigcap_{j=1}^{t} S_j$, while no party learns anything else. Parties P_1 to P_{t-1} will be called *clients*, and P_t *server*. Unlike in the Davidson-Cid two-party PSI, it is the server that computes the intersection.

For the sake of simplicity, we use the same notations with Davidson-Cid's paper. The formal description of the protocol is given Fig. 2. The detailed steps are as follows:

Input: Each of the clients $P_i, i \in \{1 \ldots, t-1\}$ holds a shared decryption key sk_i of an additively homomorphic threshold encryption scheme with decryption to zero algorithm $\mathsf{ShDec0}$. The public key pk is available to all parties.

The size of the private sets $|S_i|, i \in \{1 \ldots, t\}$ is also available to all parties.

Initialization: P_t randomly selects a set of hash functions $\{h_1, h_2, \ldots, h_k\} \in \mathcal{H}$ and sends them to the clients P_i, $1 \leq i \leq t-1$.

Local EIBF generation: Each client P_i, where $1 \leq i \leq t-1$,

1) Computes their corresponding Bloom filter $\mathbf{BF}_i$ of their data set S_i.
2) Inverts $\mathbf{BF}_i$ to obtain $\mathbf{IBF}_i$.
3) Computes their encrypted inverted Bloom filter $\mathbf{EIBF}_i$ by encrypting each element $\mathbf{IBF}_i[j]$ in the inverted Bloom filter with the public key pk.
4) Sends $\mathbf{EIBF}_i$ to P_t (server).

Set Intersection computation: The server P_t:

1) Computes k hash values of each element $y_j \in S_t$, and for each i, computes $\{C_1^{i,j}, \ldots, C_k^{i,j}\}$, as $C_d^{i,j} = \mathbf{EIBF}_i[h_d(y_j)]$ for all $j \in \{1, \ldots, n_t\}$.
2) Computes $C_j^i = C_1^{i,j} +_H \cdots +_H C_k^{i,j}$. Then, he computes $C_j = \mathsf{ReRand}(C_j^1 +_H \cdots +_H C_j^{t-1})$. The rerandomization is necessary because of the deterministic nature of the homomorphic operation.
3) Sends $(C_1, C_2, \ldots, C_{n_t})$ to ℓ parties among $t-1$ parties and asks them to mutually decrypt-to-zero each C_j. We allow $\ell \leq t-1$ for more flexibility, although often $\ell = t-1$.
4) Each client P_i computes their decryption share $sh_{i,j}$ for all $j \in \{1, \ldots, n_t\}$ as follows: They first send a randomized $C_j^{r_{i,j}}$ to the server; the server combines them into $\bar{C}_j = C_j^{\sum r_{i,j}}$ and sends $\bar{C}_j$ to each party. Now each party decrypts to obtain their shares $sh_{i,j}$ for all $j \in \{1, \ldots, n_t\}$. Finally the shares are sent to the server. Note that in Figure 2 we have abused notation to improve the readability, and this whole step of producing shares with the $\mathsf{ShDec0}$ algorithm is denoted as $sh_{i,j} = \mathsf{ShDec0}(\mathsf{sk}_i, C_j)$ for all $j \in \{1, \ldots, n_t\}$.
5) For each $j \in \{1, \ldots, n_t\}$ the server runs the combining algorithm on the obtained shares and computes $\mathsf{Dec}(C_j) \leftarrow \mathsf{Comb}(sh_{1,j}, \ldots, sh_{t-1,j})$.
6) For each $j \in \{1, \ldots, n_t\}$, if $\mathsf{Dec}(C_j) = 0$ the server adds the corresponding y_j to the intersection $S = \{y_j\} \cup S$, otherwise he discards y_j.

Output: P_t outputs S.

A. Protocol Correctness

P_t can compute the intersection if the corresponding plaintexts of all C_j^i's are zero. This happens only when y_j appears in every S_i, for $i \in \{1, \ldots, t-1\}$ simultaneously which results in $y_j \in S_1 \cap \cdots \cap S_{t-1}$. The correctness is not perfect as in [3], because of the false positive probability of Bloom filters. This probability is however can be made negligible in number of hash functions k (see Sec. II-A).

B. Security Against Semi-Honest Adversaries

We will prove the security of the protocol under the assumption of the existence of a IND-CPA-secure additively homomorphic threshold PKE scheme Π with threshold $\ell < t$. Here, we consider the protocol participants as corrupted by the *adversary* where they are curious but honest. Outside adversaries are not considered in this context as they can be mitigated by standard network security techniques.

We remind that the inputs of the clients are $\mathsf{Inp}_i = (S_i, |S_j|_{j=1}^t, \mathsf{pk}, \mathsf{sk}_i)$, $i \in \{1, \ldots, t-1\}$ whereas of the server it is $\mathsf{Inp}_t = (S_t, |S_j|_{j=1}^t, \mathsf{pk})$. The output of the clients is $\emptyset$, and of the server $S = \bigcap_{j=1}^{t} S_j$. We will consider two scenarios in both of which we assume $\eta < \ell$:

1) The server P_t is honest, and a subset $\mathsf{P}_{\mathcal{I}}$ of the clients is corrupted.
2) The server P_t is corrupted, and a subset $\mathsf{P}_{\mathcal{I}}$ of the clients is corrupted.

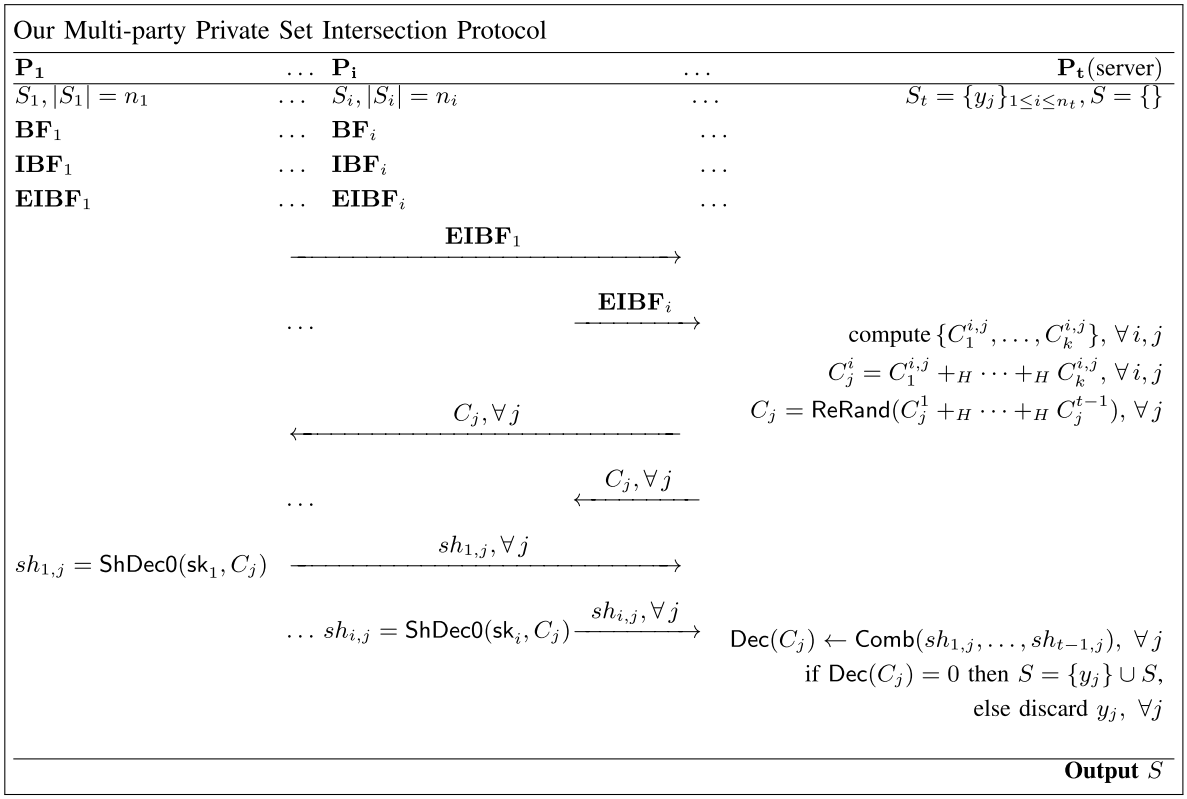

Fig. 2. Our multi-party private set intersection protocol based on Bloom filters.

In the first case, the simulator is given $\mathsf{Inp}_{\mathcal{I}}$ and no output and he needs to simulate the honest server towards the η corrupted parties. He needs to produce simulated $(\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_{n_t})$ indistinguishable from the real $(C_1, C_2, \ldots, C_{n_t})$, because this is the only message that appears in the view of the corrupted parties. The simulator $\mathcal{S}$ starts by choosing a random input for the honest parties that comply with what the corrupted parties know - the public hash functions $h_1, \ldots, h_k$, and the sizes of the sets of the honest parties. Then $\mathcal{S}$ follows the protocol and produces $(\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_{n_t})$ from the EIBFs of the corrupted parties and the ones produced from random input of the honest parties. Suppose there is an adversary $\mathcal{A}$ that can distinguish between $(C_1, C_2, \ldots, C_{n_t})$ and $(\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_{n_t})$ with non-negligible probability. Since each of C_i and $\tilde{C}_1$ are rerandomized homomorphic sums, they both look like fresh encryptions of a IND-CPA secure scheme Π. Therefore, it is straightforward to construct an adversary $\mathcal{A}_\Pi$ from $\mathcal{A}$ that breaks the IND-CPA-security of Π.

If the server P_t is also corrupted, but the number of corrupted clients η is still smaller than the threshold ℓ, the corrupted parties learn also the output of the protocol, i.e. the intersection $S = \bigcap_{j=1}^{t} S_j$. In this case, the simulator $\mathcal{S}$ needs to simulate the EIBFs of the honest parties, but also to be able to produce decryption shares without the knowledge of a secret share of the key. The simulated decryption shares must be such that, after combining all of the decryption shares, the result agrees with the output of P_t. The simulator starts by choosing sets $\tilde{S}_i$, $i \in \bar{\mathcal{I}}$, such that $\bigcap_{j \in \mathcal{I}} S_j \cap \bigcap_{j \in \bar{\mathcal{I}}} \tilde{S}_j = S$. He then follows the protocol, forms the EIBFs corresponding to the sets $\tilde{S}_i$ and sends them to the corrupted server. The corrupted clients also send their randomized EIBFs to the server. The server then computes the $(\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_{n_t})$ as in the protocol and sends them to a subset of ℓ parties. Since the number of corrupted parties η is smaller than ℓ, there must be at least one honest party that receives $(\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_{n_t})$, in particular, there are $\ell - \eta$ honest parties, The simulator receives these, and simulates the shares of the honest parties as follows: It invokes the share decryption algorithm $\mathsf{ShDec0}$ on $\tilde{C}_i$ and random element from the secret key space, $\tilde{\mathsf{sk}}_k$ for each of $\ell - \eta - 1$ honest parties. For the last remaining honest party, if $y_j \in S$, the simulator computes the decryption share from the decryption shares of the corrupted parties and the simulated honest parties, such that the combining algorithm Comb outputs 0. The simulator achieves this by calling a simulator for $\mathsf{ShDec0}$. Note that such a simulator exists both for the Elgamal threshold encryption and Pailler generalized threshold encryption schemes. If $y_j \notin S$, it forces the output to a random value. Now clearly the simulated EIBFs are indistinguishable from the real ones, otherwise the IND $-$ CPA security of the TPKE breaks. Also, the decryption shares are produced by a simulator for $\mathsf{ShDec0}$, so the adversary cannot distinguish them from the real execution.

C. Complexity Analysis

1) Communicational Complexity: Our protocol consists of five rounds. The amount of communicated data of the clients is dominated by the first round in which the clients send to the server their **EIBF**s each of which contains m ciphertexts. For an optimal Bloom filter size, m must be at least $kn \log e$ bits

long (see Section II-A). Therefore, to define the complexity in terms of the size of the data sets n, one can safely say that each client sends $\mathcal{O}(\lambda n)$ ciphertexts to the server, where $\lambda = k$ is a statistical security parameter. To compare the complexity to that of previous designs, we set the server's communication cost to $\mathcal{O}(n\ell)$ ciphertexts. Note that this cost is incurred only during the shared decryption (rounds $2-5$),

2) Computational Complexity: The computational complexity is dominated by the first round in which each client performs $\mathcal{O}(m)$ (or equivalently $\mathcal{O}(\lambda n)$) encryptions. Here we focus on the concrete instantiation from the Elgamal or Paillier scheme, where the homomorphic addition $+_H$ is multiplication in $\mathbb{Z}_N$. The details are as follows.

In the first round, each client P_i computes its Bloom filter of size m by computing k hash values of each element requiring $\mathcal{O}(kn)$ hash computations. Then, each client makes $\mathcal{O}(\lambda n)$ encryptions for encrypting its Bloom filter. In the second round, for each element of his private set S_t, the server P_t makes k hash evaluations to retrieve the corresponding values in the encrypted inverted Bloom filters. This requires $\mathcal{O}(kn)$ hash computations. When considering all elements in its private set, P_t makes $\mathcal{O}(n(t+k))$ homomorphic additions in total. Afterwards, the server initiates shared decryption-to-zero for all C_j's. Each involved client runs the $\mathsf{ShDec0}$ algorithm which essentially consists of two parts. First, each client randomizes the ciphertext by raising it to the power of a random value and sends back to the server. The server multiplies all randomized ciphertexts performing $\mathcal{O}(n\ell)$ multiplications and resends to the clients. Then, each client calculates his decryption share. This requires $\mathcal{O}(n)$ exponentiation for each client. The server P_t makes $\mathcal{O}(n)$ decryptions in total in the last round.

IV. Our Threshold Multi-Party PSI Protocol

A common requirement in real-world applications is to compute a set of elements that appear in the private sets of most of the involved parties, say in at least $\mathcal{T}$ private sets. More formally, we would like to compute the *threshold set intersection* $S_{\mathcal{T}} = \{x \in S_t | \exists S_{i_1}, \ldots, S_{i_{\mathcal{T}}}, x \in \bigcap_{j=1}^{\mathcal{T}} S_{i_j}\}$. Based on the protocol described in Section III, we here propose a new protocol for achieving this goal. Our Threshold Multi-Party PSI Protocol makes use of a slightly modified version of the Secure Comparison Protocol of Kerschbaum *et al.* [37] (see Section II-E). The difference to this protocol is that the server that initiates the protocol does not actually take active part in it. Everything else is exactly the same, so all security arguments are naturally inherited.

The formal description of our T-MPSI protocol is given Fig. 3. The detailed steps are as follows:

Input: As in our Multi-party PSI protocol from Section III we have $t-1$ clients $P_1, \ldots, P_{t-1}$ and a server P_t, each of which holds a private data set S_i of size n_i. Let pk be the public key of a secret key sk shared among all $t-1$ clients of an additively homomorphic threshold encryption scheme. The parties compute the *threshold intersection* $S_{\mathcal{T}}$ as follows.

Initialization: P_t randomly selects a set of hash functions $\{h_1, h_2, \ldots, h_k\} \in \mathcal{H}$ and sends them to the clients P_i, where $1 \le i \le t-1$.

Local EBFs generation: Each client P_i, where $1 \le i \le t-1$:

1) Computes their Bloom filter of their private data set S_i, where $1 \le i \le t-1$.
2) Computes their encrypted Bloom filter $\mathbf{EBF}_i$ by encrypting each element of $\mathbf{BF}_i[j]$ using pk.
3) Forward their $\mathbf{EBF}_i$'s to the server P_t.

We note that in this protocol, inverted Bloom filters are not required.

Set Intersection generation by the server: The server P_t:

1) Computes k hash values of each element $y_j \in S_t$, and for each party P_i.
2) Extracts $\{C_1^{i,j}, \ldots, C_k^{i,j}\}$, where $C_d^{i,j} = \mathbf{EBF}_i[h_d(y_j)]$ and $j \in \{1, \ldots, n_t\}$.
3) Computes $C_j^i = \mathsf{ReRand}(C_1^{i,j} +_H \cdots +_H C_k^{i,j})$ for each $y_j \in S_t$ and for each party P_i $(1 \le i \le t-1)$. We emphasize that if $y_j \in S_i$ where $i \ne t$, then the corresponding plaintext of C_j^i must be k which corresponds to the number of entries of y_j in the Bloom filter as all one.
4) For each C_j^i computes a fresh encryption $\mathsf{Enc}(\mathsf{pk}, k)$.
5) Runs $t \cdot n_t$ SCP protocols ([37]) in parallel with any ℓ clients P_i to compare C_j^i to $\mathsf{Enc}(\mathsf{pk}, k)$.
6) Gets the results $\mathsf{Enc}(\mathsf{pk}, \alpha_{i,j})$ from the SCP. The output $\mathsf{Enc}(\mathsf{pk}, \alpha_{i,j})$ will be the encryption of 1 if the decrypted value of C_j^i is greater than or equal to k and will be the encryption of 0 if the decrypted value of C_j^i is less than k.
7) Computes, $\mathsf{Enc}(\alpha_j) = \mathsf{ReRand}(\mathsf{Enc}(\alpha_{1,j}) +_H \cdots +_H \mathsf{Enc}(\alpha_{t-2,j}) +_H \mathsf{Enc}(\alpha_{t-1,j}))$.
8) Runs n_t SCPs in parallel with any ℓ clients to compare $\mathsf{Enc}(\alpha_j)$ to a fresh encryption $\mathsf{Enc}(\mathcal{T})$.
9) Gets the results from the SCP $(\mathsf{Enc}(\beta_1), \ldots, \mathsf{Enc}(\beta_{n_t}))$, randomizes each one of them and obtains $C_j = \mathsf{ReRand}(\mathsf{Enc}(\beta_j))$.
10) Asks ℓ clients P_i to perform joint decryption of $C_j, \forall j$ (in a random order).
11) After combining the shares from the involved clients obtains $(\mathsf{Dec}(C_1), \ldots, \mathsf{Dec}(C_{n_t}))$.
12) For all j, if $\mathsf{Dec}(C_j) = 1$ then adds y_j to $S_{\mathcal{T}}$, else discards y_j.

Output: P_t outputs $S_{\mathcal{T}}$.

A. Protocol Correctness

The server P_t computes the threshold intersection of his data set if β_j is greater than or equal to the predetermined threshold $\mathcal{T}$. The reason is that β_j counts the number of appearances of $y_j \in S_t$ in clients' private data sets. As previously, the correctness is not perfect.

B. Security

As in our MPSI protocol from Section III, we will prove the security of the protocol under the assumption of existence of a IND-CPA-secure additively homomorphic threshold PKE scheme Π with threshold $\ell < t$. We will use the notations from Section III. The difference is that the output

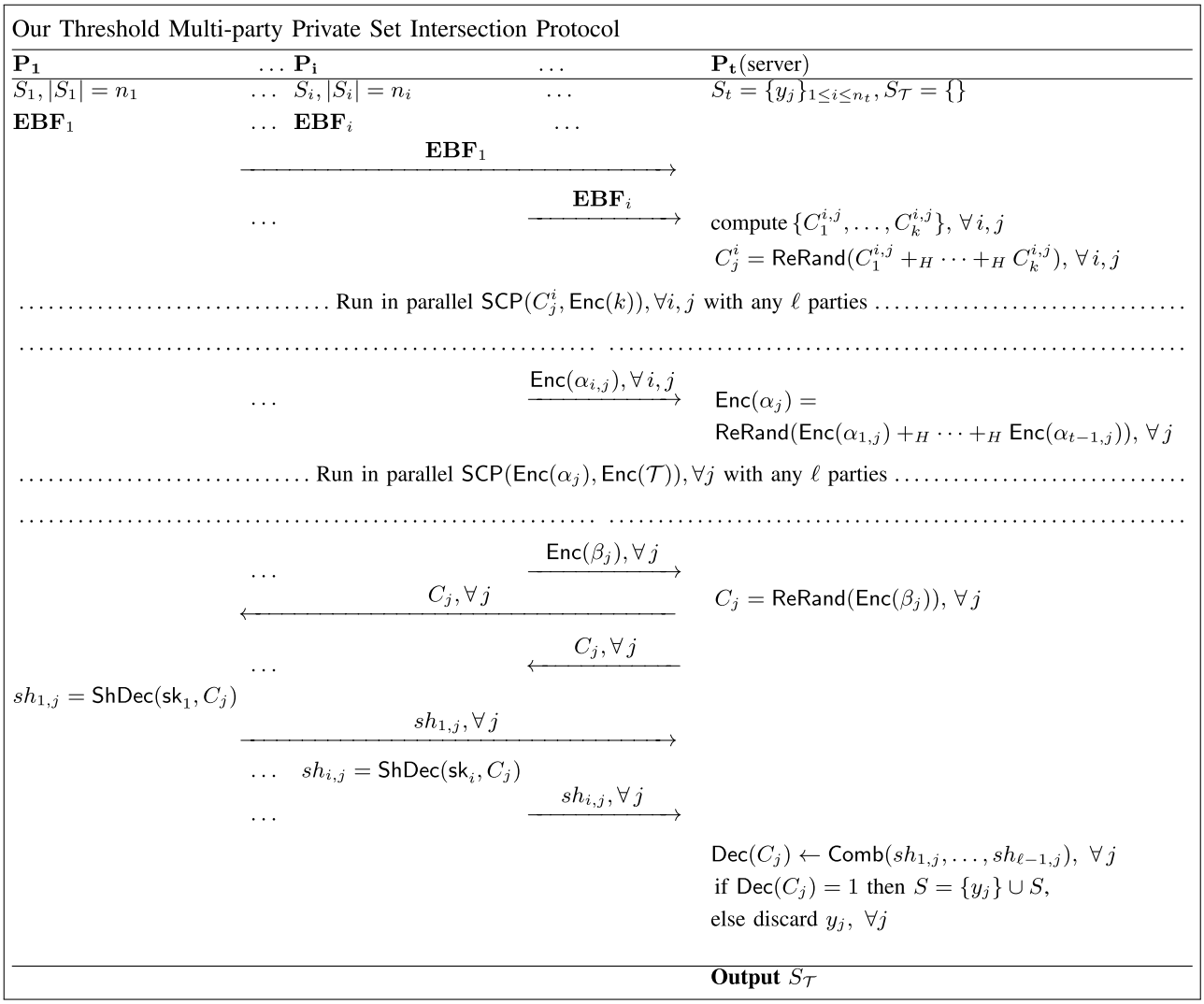

Fig. 3. Our threshold multi-party private set intersection protocol based on Bloom filters.

of the server is now the threshold intersection $S_\mathcal{T} = \{x \in S_t | \exists S_{i_1}, \ldots, S_{i_\mathcal{T}}, x \in \bigcap_{j=1}^{\mathcal{T}} S_{i_j}\}$.

Again, we will consider two scenarios in both of which $\eta < \ell$:

1) The server P_t is honest, and a subset $\mathsf{P}_\mathcal{I}$ of η clients is corrupted.
2) The server P_t is corrupted, and a subset $\mathsf{P}_\mathcal{I}$ of η clients is corrupted.

In the first case, given $\mathsf{Inp}_\mathcal{I}$ and no output, the simulator first needs to produce simulated $(\tilde{C}_1^i, \tilde{C}_2^i, \ldots, \tilde{C}_{n_t}^i), i \in \{1, \ldots, t-1\}$ that the corrupted parties receive as input in the SCP protocol. Similarly as in the MPSI protocol, $\mathcal{S}$ starts by choosing random inputs of the correct size for the honest parties, and follows the protocol to produce $(\tilde{C}_1^i, \tilde{C}_2^i, \ldots, \tilde{C}_{n_t}^i), i \in \{1, \ldots, t-1\}$. Then $\mathcal{S}$ gets involved in the first run of the SCP protocol, simulating the actions of the honest parties. Basically, for each C_j^i there is one comparison to a fresh encryption of $E(k)$ carried out using an atomic SCP. In particular, to simulate the output of a honest party, for received $(a_{1,i,j}^r, a_{2,i,j}^r, a_{3,i,j}^r)$ from a corrupt party, the simulator produces $(a_{1,i,j}^{r+1}, a_{2,i,j}^{r+1}, a_{3,i,j}^{r+1})$ following exactly the SCP protocol and gives it to a corrupt party. The last $a_{3,i,j}^\ell$ is then jointly decrypted by the ℓ involved parties. In the joint decryption, the corrupt parties send their decryption shares to the simulator. The simulator encrypts 0 n_t times and randomizes $\mathsf{Enc}(\alpha_j)$ for each j. Then it gets involved in the SCP once again, and follows the protocol: for received $(a_{1,i,j}^r, a_{2,i,j}^r, a_{3,i,j}^r)$ from a corrupt party, the simulator produces $(a_{1,i,j}^{r+1}, a_{2,i,j}^{r+1}, a_{3,i,j}^{r+1})$ and sends to the next corrupt party. In the end, the simulator collects the decrypted shares, and rerandomizes the results, exactly as in the previous run. Once again, it sends the obtained C_j to the corrupt parties for joint decryption, and they return their decryption shares.

Since everything that the simulator sends to the corrupted parties is rerandomized, it is indistinguishable from fresh encryption. Thus, an adversary that can distinguish the simulated run from the real one, can break the IND-CPA-security of the underlying Π.

If the server P_t is also corrupted, the corrupted parties learn also the output of the protocol, i.e. the threshold intersection $S_\mathcal{T}$. Now, the simulator needs to simulate everything as in

the previous case, plus the shares of the honest parties in the last joint decryption. This can be done by forcing the outcome of the algorithm Comb to 1, for all $y \in S_{\mathcal{T}}$, similarly as in the MPSI protocol from Section III where the result was forced to 0 by the simulator. The security argument is then essentially the same.

C. Complexity Analysis

1) Communicational Complexity: The number of rounds of the protocol is $\mathcal{O}(t)$, which is dominated by the Π_{SCP} protocol. The communication complexity of the server is dominated by the first execution of Π_{SCP} in which the server sends $\mathcal{O}(nt\ell)$ ciphertexts. As in our first protocol, the communicational complexity of each client in the first round is $\mathcal{O}(\lambda n)$ ciphertexts. In the rest of the rounds, the communicational complexity of each (active) client is due to the execution of the Π_{SCP} protocols which is $\mathcal{O}(nt)$ ciphertexts. Therefore, the dominating communicational complexity of each client is $\mathcal{O}(\max(\lambda, t)n)$ ciphertexts. For the details of the analysis of the Π_{SCP} protocol, we refer to Section II-E.

2) Computational Complexity: One of the dominating steps in terms of computational complexity of each of the clients is the construction of their encrypted Bloom filters which takes $\mathcal{O}(\lambda n)$ operations. The second dominating computational step of the protocol is the execution of the Π_{SCP}'s. We remind that each execution of Π_{SCP} does $\mathcal{O}(1)$ homomorphic encryptions for each client and the server. As the Π_{SCP} protocol is executed nt times, for the server and each (active) client, the computational complexity is for each of them $\mathcal{O}(nt)$ homomorphic encryptions. Therefore, the dominating complexity for each client is $\mathcal{O}(\max(\lambda, t)n)$, and it is $\mathcal{O}(nt)$ for the server.

V. Implementation

We have developed a reference implementation of both protocols in C++. The implementation depends on the GMP [46] and NTL [47] libraries, as well as code for MurmurHash3 [48], which is used as a fast hash function for Bloom filters. The implementation for the MPSI protocol runs on one machine and spawns one thread for each client for concurrency, while the main thread represents the server. The T-MPSI protocol runs on a single thread so it demonstrates an upper bound of the run time, which we believe can still be reduced significantly using multi-threading.

A. Set-up

We evaluate the run time performance of the protocols by performing 10 set intersections for each combination of parameters, where all sets contain n random elements. We provide both the mean and standard deviation of these measurements. Our benchmarks were executed on a 64-bit Unix machine with an Intel Core i7-1065G7 processor at 8 × 1.30GHz and a memory capacity of 16GB. For our work, we choose security parameter $\kappa = 1024$ as is common for public-key encryption.

B. MPSI Protocol

We compare the run time of our private set intersection protocol with the protocol by Kolesnikov *et al.* [17] (with $\kappa = 128$ as is common for Oblivious Transfers) because their implementation is readily available and the empirical run time appears to be generally superior to that of similar works. While their protocol is time-efficient for large numbers of elements, our proposed protocol is efficient for larger numbers of parties t. Our bandwidth is also drastically lower for an increasing number of (corruptible) parties. To ensure an overwhelming probability of correctness we fixed the accuracy to a false positive rate of less than 1%, by choosing $k = 7$ hashes and $m = \lfloor \frac{7n}{\log 2} \rfloor$ bits. We note that Kolesnikov *et al.* [17] achieve a far higher probability of correctness (the probability of incorrectness is 2^{-40}) since an increase in their number of bits achieves an exponential reduction in the false positive rate. Applications that require such a low false positive rate can choose a larger m and k but they will incur longer run times and bandwidth.

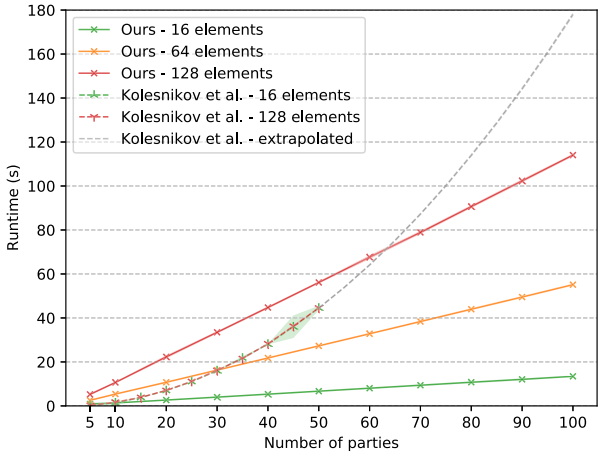

Fig. 4. Run time comparison for the MPSI protocol at $\ell = \frac{1}{2}t$ averaged over 10 runs. The gray line indicates the extrapolated results for Kolesnikov *et al.* The filled in areas represent the 99% confidence intervals ($\pm 3\sigma$).

1) Run Time Results: We present the results of our run time comparison in Figure 4 for a threshold $\ell = \lfloor \frac{1}{2}t \rfloor$. We also evaluate the run time for $\ell = t - 1$, the result of which is relatively similar and can be found in Table II. We capped the number of parties t at 50 for Kolesnikov *et al.* [17] but this was only due to memory constraints when running the protocol. Our protocol can be easily run with many more parties, and there are no theoretical reasons why the results could not be extended to a larger number of parties for Kolesnikov *et al.* as well. For this reason we extrapolated the results to $t = 100$ in Figure 4 using a quadratic fit.

2) Discussion: For every number of elements n there is a lower bound on the number of parties t where our protocol becomes more efficient than [17]. For small n such as $n = 16$ our protocol already becomes more efficient at $t \geq 10$ parties, and for a larger $n = 64$ our protocol is more efficient from $t \geq 30$ parties onwards. Notably, even for 100 parties our runtime stays within 60 seconds for $n = 64$ and within 120 seconds for $n = 128$. As highlighted by the filled in areas in our plot that represent three standard deviations from the mean, our measurements are statistically consistent. For a different collusion threshold ℓ runtime changes only slightly as demonstrated in Table II.

TABLE II

MEAN RUN TIME RESULTS IN SECONDS FOR THE MPSI PROTOCOL AVERAGED OVER 10 RUNS, WHERE ± REPRESENTS THE STANDARD DEVIATION σ. *FOR THESE RESULTS WE HAD TO CHOOSE $\ell = 30$ TO STAY WITHIN MEMORY

		Ours			Kolesnikov et al.	
		$n = 2^4$	$n = 2^6$	$n = 2^8$	$n = 2^4$	$n = 2^8$
$t = 10$	$\ell = 5$	1.05 ± 0.04	5.55 ± 0.03	23.19 ± 0.44	1.12 ± 0.03	1.61 ± 0.25
	$\ell = 9$	1.20 ± 0.20	5.69 ± 0.15	23.75 ± 0.65	1.65 ± 0.01	1.71 ± 0.01
$t = 20$	$\ell = 10$	2.74 ± 0.02	10.86 ± 0.21	46.12 ± 0.14	7.17 ± 0.05	7.22 ± 0.02
	$\ell = 19$	2.84 ± 0.10	11.20 ± 0.30	47.47 ± 1.38	7.06 ± 0.02	7.24 ± 0.01
$t = 30$	$\ell = 15$	3.99 ± 0.03	16.39 ± 0.26	70.46 ± 0.65	16.17 ± 0.18	15.90 ± 0.02
	$\ell = 29$	4.13 ± 0.14	17.13 ± 0.78	72.36 ± 2.01	16.08 ± 1.73	16.49 ± 1.83
$t = 40$	$\ell = 20$	5.39 ± 0.04	21.91 ± 0.03	96.46 ± 1.00	28.10 ± 0.02	28.48 ± 0.02
	$\ell = 39$	5.55 ± 0.17	22.53 ± 0.64	98.07 ± 1.82	29.83 ± 0.04	30.59 ± 0.07
$t = 50$	$\ell = 25$	6.71 ± 0.04	27.31 ± 0.02	117.98 ± 0.09	44.05 ± 0.03	44.63 ± 0.04
	$\ell = 49$	6.92 ± 0.22	28.16 ± 0.88	121.35 ± 3.46	45.63* ± 0.06	46.34* ± 0.06

TABLE III

MEAN RUN TIME RESULTS IN SECONDS FOR THE T-MPSI PROTOCOL AVERAGED OVER 10 RUNS, WHERE ± REPRESENTS THE STANDARD DEVIATION σ

		$n = 2^2$	$n = 2^4$	$n = 2^6$
$t = 3$	$\ell = 1$	0.50 ± 0.02	1.92 ± 0.01	7.62 ± 0.02
	$\ell = 2$	0.57 ± 0.07	2.19 ± 0.28	8.73 ± 1.14
$t = 4$	$\ell = 2$	0.82 ± 0.00	3.28 ± 0.01	13.12 ± 0.02
	$\ell = 3$	0.91 ± 0.10	3.66 ± 0.39	14.60 ± 1.51
$t = 6$	$\ell = 3$	1.52 ± 0.03	5.83 ± 0.02	23.34 ± 0.03
	$\ell = 5$	1.75 ± 0.24	6.86 ± 1.06	27.47 ± 4.24
$t = 8$	$\ell = 4$	2.30 ± 0.01	9.17 ± 0.02	37.07 ± 0.51
	$\ell = 7$	2.81 ± 0.52	11.20 ± 2.08	44.94 ± 8.08

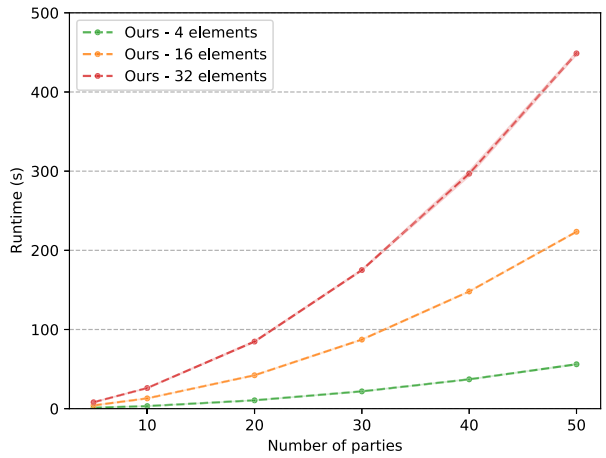

Fig. 5. Run time comparison for the MPSI protocol at $\ell = \frac{1}{2}t$ averaged over 10 runs. The filled in areas represent the 99% confidence intervals ($\pm 3\sigma$).

C. T-MPSI Protocol

We also present an implementation of our T-MPSI protocol. Even though the runtime seems promising already, these results can be considered an upper bound as we believe a significant gain in efficiency can still be achieved through multi-threading. We evaluate the run time for T-MPSI on smaller instances of the problem than the MPSI protocol, with fewer parties and lower set sizes to perform the experiments in reasonable time.

1) Run Time Results: We present the results of our run time experiment in Figure 5, and the results for smaller instances can be found in Table III. We do not compare our protocol against another as there were no other threshold-intersection implementations available. As a set-intersection threshold we chose half of the number of parties: $\mathcal{T} = \frac{t}{2}$.

2) Discussion: For smaller set sizes the experiments show that our protocol proves to scale efficiently. Specifically, when run with $n = 4$ elements, as might be the case in a voting scenario, the protocol takes roughly 1 minute for $t = 50$ parties. Even for $n = 32$ elements and $t = 50$ parties runtime stays within 8 minutes. While the protocol scales quadratically in the number of parties t these results demonstrate that for a small number of elements n the protocol can still be considered practical. Note that when the collusion threshold ℓ is chosen to be a fixed number, the protocol instead scales linearly with a growing number of parties t.

VI. CONCLUSION

In this paper, we describe two efficient multi-party private set intersection protocols for both regular and threshold intersections, which we prove to be secure in the semi-honest model. For threshold intersections, only those elements of the server that appear in at least a pre-determined threshold number of private sets are revealed. Our protocols are based on Bloom filters as an efficient set representation, and threshold homomorphic PKEs. To the best of our knowledge, our proposed protocols are the fastest tools for computing private set intersections and threshold set intersections in real-world problems where many parties are involved in the computation.

ACKNOWLEDGMENT

The authors would like to thank Dr. Majid Nateghizad for helping them in analysing the security of the Secure Comparison Protocol used in this paper.

REFERENCES

[1] A. Cerulli, E. D. Cristofaro, and C. Soriente, "Nothing refreshes like a repsi: Reactive private set intersection," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 10892, B. Preneel and F. Vercauteren, Eds. Leuven, Belgium: Springer, 2018, pp. 280–300, doi: 10.1007/978-3-319-93387-0_15.

[2] E. D. Cristofaro, P. Gasti, and G. Tsudik, "Fast and private computation of cardinality of set intersection and union," in *Cryptology and Network Security*, vol. 7712, J. Pieprzyk, A. Sadeghi, and M. Manulis, Eds. Darmstadt, Germany: Springer, 2012, pp. 218–231, doi: 10.1007/978-3-642-35404-5_17.

[3] A. Davidson and C. Cid, "An efficient toolkit for computing private set operations," in *Information Security and Privacy* (Lecture Notes in Computer Science), vol. 10343, J. Pieprzyk and S. Suriadi, Eds. Auckland, New Zealand: Springer, 2017, pp. 261–278, doi: 10.1007/978-3-319-59870-3_15.

[4] S. K. Debnath and R. Dutta, "Secure and efficient private set intersection cardinality using Bloom filter," in *Information Security* (Lecture Notes in Computer Science), vol. 9290, J. López and C. J. Mitchell, Eds. Trondheim, Norway: Springer, 2015, pp. 209–226, doi: 10.1007/978-3-319-23318-5_12.

[5] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: An efficient and scalable protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, A. Sadeghi, V. D. Gligor, and M. Yung, Eds. Berlin, Germany, Nov. 2013, pp. 789–800, doi: 10.1145/2508859.2516701.

[6] K. B. Frikken, "Privacy-preserving set union," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 4521, J. Katz and M. Yung, Eds. Zhuhai, China: Springer, Jun. 2007, pp. 237–252, doi: 10.1007/978-3-540-72738-5_16.

[7] F. Kerschbaum, "Outsourced private set intersection using homomorphic encryption," in *Proc. 7th ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, H. Y. Youm and Y. Won, Eds. Seoul, South Korea, May 2012, pp. 85–86, doi: 10.1145/2414456.2414506.

[8] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *Proc. 24th USENIX Secur. Symp., (USENIX Security)*, J. Jung and T. Holz, Eds. Washington, DC, USA: USENIX Association, Aug. 2015, pp. 515–530. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas

[9] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 11478, Y. Ishai and V. Rijmen, Eds. Darmstadt, Germany: Springer, May 2019, pp. 122–153, doi: 10.1007/978-3-030-17659-4_5.

[10] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 3027, C. Cachin and J. Camenisch, Eds. Interlaken, Switzerland: Springer, May 2004, pp. 1–19, doi: 10.1007/978-3-540-24676-3_1.

[11] B. Pinkas, T. Schneider, and M. Zohner. (2016). *Scalable Private Set Intersection Based on OT Extension*. Cryptology ePrint Archive, Report 2016/930. [Online]. Available: https://eprint.iacr.org/2016/930

[12] Z. Wang, K. Banawan, and S. Ulukus, "Multi-party private set intersection: An information-theoretic approach," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 366–379, Mar. 2021, doi: 10.1109/JSAIT.2021.3057597.

[13] R. Inbar, E. Omri, and B. Pinkas, "Efficient scalable multiparty private set-intersection via garbled Bloom filters," in *Security and Cryptography for Networks* (Lecture Notes in Computer Science), vol. 11035, D. Catalano and R. D. Prisco, Eds. Amalfi, Italy: Springer, Sep. 2018, pp. 235–252, doi: 10.1007/978-3-319-98113-0_13.

[14] J. H. Cheon, S. Jarecki, and J. H. Seo, "Multi-party privacy-preserving set intersection with quasi-linear complexity," *IEICE Trans. Fundamentals Electron., Commun. Comput. Sci.*, vol. 95, no. 8, pp. 1366–1378, Aug. 2012.

[15] X. Cao, H. Li, L. Dang, and Y. Lin, "A two-party privacy preserving set intersection protocol against malicious users in cloud computing," *Comput. Standards Interfaces*, vol. 54, pp. 41–45, Nov. 2017.

[16] M. Sheikhalishahi, G. Tillem, Z. Erkin, and N. Zannone, "Privacy-preserving multi-party access control," in *Proc. 18th ACM Workshop Privacy Electron. Soc. (WPES)*, 2019, pp. 1–13.

[17] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. Dallas, TX, USA, Nov. 2017, pp. 1257–1272, doi: 10.1145/3133956.3134065.

[18] W. Lueks, J. Hoepman, and K. Kursawe, "Forward-secure distributed encryption," in *Privacy Enhancing Technologies* (Lecture Notes in Computer Science), vol. 8555, E. D. Cristofaro and S. J. Murdoch, Eds. Amsterdam, The Netherlands: Springer, Jul. 2014, pp. 123–142, doi: 10.1007/978-3-319-08506-7_7.

[19] L. Kissner and D. X. Song, "Privacy-preserving set operations," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 3621, V. Shoup, Ed. Santa Barbara, CA, USA: Springer, Aug. 2005, pp. 241–257, doi: 10.1007/11535218_15.

[20] P. K. Y. Lai, S. Yiu, K. P. Chow, C. F. Chong, and L. C. K. Hui, "An efficient Bloom filter based solution for multiparty private matching," in *Proc. Int. Conf. Secur. Manage., (SAM)*, H. R. Arabnia and S. Aissi, Eds. Las Vegas, NV, USA: CSREA Press, Jun. 2006, pp. 286–292.

[21] A. Miyaji and S. Nishida, "A scalable multiparty private set intersection," in *Network and System Security* (Lecture Notes in Computer Science), vol. 9408, M. Qiu, S. Xu, M. Yung, and H. Zhang, Eds. New York, NY, USA: Springer, Nov. 2015, pp. 376–385, doi: 10.1007/978-3-319-25645-0_26.

[22] A. Miyaji, K. Nakasho, and S. Nishida, "Privacy-preserving integration of medical data: A practical multiparty private set intersection," *J. Med. Syst.*, vol. 41, no. 3, p. 37, Mar. 2017, doi: 10.1007/s10916-016-0657-4.

[23] J. H. Cheon, S. Jarecki, and J. H. Seo, "Multi-party privacy-preserving set intersection with quasi-linear complexity," *IACR Cryptol. ePrint Archive*, vol. 2010, p. 512, Oct. 2010. [Online]. Available: https://eprint.iacr.org/2010/512

[24] C. Hazay and M. Venkitasubramaniam, "Scalable multi-party private set-intersection," in *Public-Key Cryptography* (Lecture Notes in Computer Science), vol. 10174, S. Fehr, Ed. Amsterdam, The Netherlands: Springer, Mar. 2017, pp. 175–203, doi: 10.1007/978-3-662-54365-8_8.

[25] D. Many, M. Burkhart, and X. Dimitropoulos, "Fast private set operations with sepia," ETH Zurich, Zürich, Switzerland, Tech. Rep. 345, 2012.

[26] D. Karapiperis, D. Vatsalan, V. S. Verykios, and P. Christen, "Large-scale multi-party counting set intersection using a space efficient global synopsis," in *Database Systems for Advanced Applications* (Lecture Notes in Computer Science), vol. 9050, M. Renz, C. Shahabi, X. Zhou, and M. A. Cheema, Eds. Hanoi, Vietnam: Springer, Apr. 2015, pp. 329–345, doi: 10.1007/978-3-319-18123-3_20.

[27] Y. Duan and J. F. Canny, "Practical private computation and zero-knowledge tools for privacy-preserving distributed data mining," in *Proc. SIAM Int. Conf. Data Mining (SDM)*. Atlanta, GA, USA: SIAM, Apr. 2008, pp. 265–276, doi: 10.1137/1.9781611972788.24.

[28] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *J. Privacy Confidentiality*, vol. 1, no. 1, Apr. 2009, doi: 10.29012/jpc.v1i1.566.

[29] B. H. Falk, D. Noble, and R. Ostrovsky, "Private set intersection with linear communication from general assumptions," in *Proc. 18th ACM Workshop Privacy Electron. Soc. (WPES@CCS)*, L. Cavallaro, J. Kinder, and J. Domingo-Ferrer, Eds. London, U.K., 2019, pp. 14–25, doi: 10.1145/3338498.3358645.

[30] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *Proc. 23rd USENIX Secur. Symp.*, K. Fu and J. Jung, Eds. San Diego, CA, USA: USENIX Association, Aug. 2014, pp. 797–812. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas

[31] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious PRF with applications to private set intersection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. Vienna, Austria, Oct. 2016, pp. 818–829, doi: 10.1145/2976749.2978381.

[32] S. Badrinarayanan, P. Miao, and P. Rindal. (2020). *Multi-Party Threshold Private Set Intersection With Sublinear Communication*. Cryptology ePrint Archive, Report 2020/600. [Online]. Available: https://eprint.iacr.org/2020/600

[33] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 3876, S. Halevi and T. Rabin, Eds. New York, NY, USA: Springer, Mar. 2006, pp. 265–284, doi: 10.1007/11681878_14.

[34] U. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized aggregatable privacy-preserving ordinal response," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, G. Ahn, M. Yung, and N. Li, Eds. Scottsdale, AZ, USA, Nov. 2014, pp. 1054–1067, doi: 10.1145/2660267.2660348.

[35] *Apple's Overview of Their Use of Differential Privacy*. Accessed: Feb. 2021. [Online]. Available: https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf

[36] Q. Xue, Y. Zhu, J. Wang, and X. Li, "Distributed set intersection and union with local differential privacy," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2017, pp. 198–205.

[37] F. Kerschbaum, D. Biswas, and S. de Hoogh, "Performance comparison of secure comparison protocols," in *Database and Expert Systems Applications*, A. M. Tjoa and R. R. Wagner, Eds. Linz, Austria: IEEE Computer Society, Sep. 2009, pp. 133–136, doi: 10.1109/DEXA.2009.37.

[38] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, Jul. 1970.

[39] P. Bose *et al.*, "On the false-positive rate of Bloom filters," *Inf. Process. Lett.*, vol. 108, no. 4, pp. 210–213, 2008, doi: 10.1016/j.ipl.2008.05.018.

[40] O. Goldreich, "Secure multi-party computation," Dept. Comput. Sci. Appl. Math., Weizmann Inst. Sci., Izrael, Tech. Rep., 1998.

[41] T. Nishide and K. Sakurai, "Distributed Paillier cryptosystem without trusted dealer," in *Information Security Applications* (Lecture Notes in Computer Science), vol. 6513, Y. Chung and M. Yung, Eds. Jeju Island, South Korea: Springer, Aug. 2010, pp. 44–60, doi: 10.1007/978-3-642-17955-6_4.
[42] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.
[43] P. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting or lotteries," in *Financial Cryptography* (Lecture Notes in Computer Science), vol. 1962, Y. Frankel, Ed. Anguilla, British West Indies: Springer, Feb. 2000, pp. 90–104, doi: 10.1007/3-540-45472-1_7.
[44] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci. (SFCs)*, Oct. 1986, pp. 162–167.
[45] T. Veugen, "Encrypted integer division," in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Seattle, WA, USA, Dec. 2010, pp. 1–6, doi: 10.1109/WIFS.2010.5711448.
[46] F. S. Foundation. (2020). *The GNU MP bignum Library*. [Online]. Available: https://gmplib.org/
[47] V. Shoup. (2020). *NTL: A Library for Doing Number Theory*. [Online]. Available: https://www.shoup.net/ntl/
[48] *Murmurhash3*. Accessed: 2020. [Online]. Available: https://github.com/aappleby/smhasher/wiki/MurmurHash3

Aslı Bay is currently an Assistant Professor with the Computer Engineering Department, Antalya Bilim University (ABU), Antalya, Turkey. She is also the Coordinator of the Cyber Security Master Program, ABU. She has broad interest in cryptography and security, with special interest in privacy preserving computations.

Zekeriya Erkin (Senior Member, IEEE) is currently an Associate Professor with the Cyber Security Group, Delft University of Technology. His research interests are on privacy enhancing technologies, particularly secure data sharing and processing: protecting sensitive data from malicious entities and service providers using cryptographic tools. He is serving at numerous committees, including IEEE IFS Technical Committee. He is an Associate Editor of IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY (T-IFS), *Eurasip Journal on Information Security*, and *Image processing* (Elsevier).

Jaap-Henk Hoepman is currently an Associate Professor with Radboud University Nijmegen. He works at iHub, the interdisciplinary research hub on digitalization and society. He is also an Associate Professor in IT law section with the Faculty of Law of the University of Groningen. Moreover, he is the Principal Scientist of the Privacy Identity Laboratory. His area of research is privacy by design, privacy enhancing technologies, and privacy friendly identity management, cryptography, and information security in general. He has been the PC chair and a steering committee member positions on various conferences and workshops.

Simona Samardjiska is currently an Assistant Professor with the Digital Security Group, Radboud University, The Netherlands. Her expertise and research interests are in the mathematics of post-quantum cryptography, especially multivariate, and code-based cryptography. She has been actively involved in the current NIST standardization process for post-quantum cryptography as a principal submitter of the Second round candidate MQDSS. She has published on several IACR conferences and IEEE symposiums and has been a program committee member of various cryptography related conferences and workshops.

Jelle Vos is currently pursuing the Ph.D. degree with Delft University of Technology. He works at the Computational Privacy Laboratory under the supervision of Zekeriya Erkin. His academic interests include the design, analysis, and implementation of cryptography-based protocols.