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

Jelle Vos
Delft University of Technology
Delft, The Netherlands
J.V.Vos@tudelft.nl

Mauro Conti
University of Padua
Padua, Italy
mauro.conti@unipd.it

Zekeriya Erkin
Delft University of Technology
Delft, The Netherlands
Z.Erkin@tudelft.nl

ABSTRACT

Today, our society produces massive amounts of data, part of which are strictly private. So, a long line of research has worked to design protocols that perform functions on such private data without revealing them. One function that has attracted significant interest is a multi-party private set operation, where each party's input is a set. The parties commonly intend to compute these sets' collective intersection (MPSI) or union (MPSU), which finds uses in various applications, including private scheduling and threat intelligence. Most current protocols use integer-based homomorphic encryption, with large elements and expensive operations, or oblivious transfers, which require communicationally-expensive pairwise interactions between all parties. Thus, existing solutions introduce significant overhead that hinders practical use. This paper considers a certain class of previously-proposed MPSI and MPSU protocols. We propose to express them in terms of new private AND or OR operations among all parties and use elliptic curves to realize these operations efficiently. We achieve a significant performance gain: Firstly, our protocols take only three rounds of communication. Secondly, our constant-time open-source implementation is two orders of magnitude faster than the state-of-the-art MPSI for small universes and outperforms the state-of-the-art MPSI for large universes for three parties or more.

KEYWORDS

private set operations, multi-party computation, homomorphic encryption

1 INTRODUCTION

Our increasingly digital society is making a growing amount of data available to computers, networks, and third parties. As a consequence, our sensitive data is in danger of getting exposed. The field of multi-party computation attempts to mitigate this by studying protocols that enable parties to perform their operations digitally without the risk of privacy-violating data leaks. Among those operations are multi-party private set operations. We consider multi-party private set intersections (MPSI) and unions (MPSU): Consider n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, who each have a set $X_1, \ldots, X_n$, respectively. For MPSIs, the task is to privately compute $X_1 \cap \cdots \cap X_n$. For MPSUs, the parties must compute $X_1 \cup \cdots \cup X_n$. Each set contains at most k elements from a finite universe $\mathcal{U}$, so $|X_i| \leq k$. In our setting, we select a leader who receives the result of the operation, but all parties are allowed to learn it. We refer to the other parties as assistants. We denote $\mathcal{P}_1$ as the leader, without loss of generality.

MPSI and MPSU protocols serve many different applications, as set operations form a fundamental building block in day-to-day functionality. For example, by using a private set intersection on possible dates for a meeting, multiple colleagues can select a meeting date at which they are all available without revealing other information about their calendar. A use case of private set unions is the creation of no-fly lists: Several agencies can prevent passengers from flying, but it would leak information if an agency knew which individuals the other agencies were investigating. The result of a private set union reveals the complete set of banned passengers, but without reference to which or how many agencies are investigating them. Other use cases of MPSIs include confidential data sharing on security incident information and botnet detection [4]. At the same time, MPSUs form the basis of other privacy-preserving protocols such as private data mining and graph algorithms [19].

These MPSI and MPSU protocols have been studied for almost two decades now, but the current state of the art still suffers from significant costs when the number of elements k in the set or the number of parties n grows, making these protocols prohibitively expensive in practice. For example, protocols using integer-based homomorphic encryption require $O(k)$ 3072-bit ciphertexts [3] and long run times due to the expensive public-key operations. Oblivious transfer-based protocols offer performance gains by offloading public-key operations to an initial phase, but they require all involved parties to send messages to all other parties.

Interestingly, several homomorphic encryption-based protocols for MPSIs [4, 5, 15, 31] and MPSUs [4] implicitly rely on secure multi-party logic in the form of private AND and OR operation. So, efficient protocols for these building blocks directly lead to efficient MPSI and MPSU protocols. In a private logic protocol, parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ submit input bits $x_1, \ldots, x_n$ to privately compute $x_1 \vee \cdots \vee x_n$. Through DeMorgan's law one can transform the same protocol to compute $x_1 \wedge \cdots \wedge x_n$. In this work we also consider the notion of 'composed' ORs and ANDs, where the parties submit multiple bits at once, and a leader chooses the bits to compute these logical operations over. While private logic protocols have been studied before [22], current solutions either provide weak privacy guarantees or require a high degree of interaction between all parties.

In this paper, we propose efficient protocols for performing these private AND and OR operations. Instead of offloading public key operations to an earlier phase like oblivious-transfer based protocols do, we make the operations significantly cheaper by using elliptic curve cryptography [6]. In this way, we decrease the computational overhead of homomorphic encryption while the parties communicate strictly in a star topology with minimal overhead.

We compare our work against four works that represent the state of the art of MPSI and MPSU protocols [4, 5, 7, 28]. The MPSI protocol by Kolesnikov et al. [28] scales efficiently with the number of elements k, but scales quadratically with the number of parties n as it requires communication between all pairs of parties. This protocol scales largely independent of the size of the universe $|\mathcal{U}|$, and it is suitable for smaller numbers of parties. Bay et al. [4], on the other hand, propose efficient MPSI and MPSU protocols that scale linearly with the number of parties n but also with the size of the universe $|\mathcal{U}|$. Therefore, these protocols are suitable for many parties so long as the universe remains small; they would be unsuitable to represent IP elements, where $|\mathcal{U}| = 2^{32} \approx 4 \times 10^9$. Bay et al. [5] also propose another protocol that would be suitable for many parties and large universes, but the final result can contain false positives, and it only outperforms Kolesnikov et al. when the number of parties is relatively large. For example, when $k = 128$, the number of parties must exceed 65. We consider the current state of the art of MPSU protocols for large universes to be Blanton & Aguiar's [7]. Their secret sharing-based protocol is concretely efficient for small set sizes k, but the round complexity is $O(\log^2 k)$.

We propose to instantiate the protocols by Bay et al. [4, 5] with our secure logic, providing significant improvements in computation and communication over the current state-of-the-art MPSIs and MPSUs. Firstly, we achieve a run time improvement of two orders of magnitude compared to the original integer-based homomorphic MPSI protocol by Bay et al. [4], as we demonstrate in Section 5. At the same time, our protocol consistently outperforms the OT-based MPSI by Kolesnikov et al. [28] for as few as three parties and onwards when we accept a false positive rate of 0.01%, which we demonstrate in Section 6.4. We also provide run time results for our MPSU protocol that are similar to the work by Blanton & Aguiar [7] but with a *constant* round complexity. We claim that with these improvements and the fact that our protocols runs in the star topology, these are the first multi-party private set operation protocols both practically deployable [38] and performant enough in practice. Concretely, our contributions are as follows:

- We present two private OR protocols in the star topology, namely a standard and composed variant, both of which can be transformed into private AND protocols. We prove our protocols to be secure in the semi-honest model using a simulation-based proof and experimentally demonstrate that their run time is constant.
- We use this private logic to instantiate MPSI and MPSU protocols based on previous work [4] that compute the exact set intersection and union efficiently for small universes. The MPSI protocol is two orders of magnitude faster.
- We instantiate an approximate MPSI protocol based on previous work [5] that scales independently from the size of the universe. The protocol is faster than the state-of-the-art protocol by Kolesnikov et al. [28] when the number of parties grows.
- We present a novel efficient MPSU protocol that scales only logarithmically with the size of the universe for a chosen constant number of interactions.[1]

The rest of the paper is organised as follows. We discuss related work in Section 2, and elliptic curve cryptography and Bloom filters in Section 3. In Section 4 we propose our private OR and AND protocols. Then, we use these protocols in Section 5 to construct an MPSI and and MPSU protocol for small universes. After that, we construct an MPSI protocol for large universes in Section 6, and an MPSU protocol for large universes in Section 7. Finally, we state opportunities for future work and conclude our paper in Section 8.

[1]All our implementations are at: https://github.com/jellevos/private-logic-and-mpso

2 RELATED WORK

In this section, we first highlight previous works on multi-party private logic, and continue with multi-party private set operations. For MPSI and MPSU protocols, there are solutions based on Oblivious Transfer (OT) [24, 28], homomorphic encryption [27], and secret sharing [29, 36], among others. For both MPSI and MPSU protocols, we provide an overview of recent works with their characteristics in Table 1, along with our constant-time protocols and the first works on in this field. In the table, t represents the maximum resistance to collusion attacks. In other words, how large can a group of colluding parties be before the protocol's privacy guarantees fail. For a fair comparison, we take $t = n - 1$.

2.1 Multi-party private logic

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

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

One of the first veto voting schemes came in the form of anonymous veto networks (AV-nets) [22], which are closely related to the dining cryptographers problem [12]. In an AV-net, any set of parties can veto some decision without the other parties identifying them. However, this requirement is not sufficient to guarantee a private OR operation. Specifically, a party can locally perform the second round of the protocol on a different input to examine the result had they changed their mind. Essentially, this means that an AV-net securely computes an OR operation between the parties outside of each colluding set, but that makes it unusable for multi-party private set operations. PriVeto [2] fixes these privacy problems using NIZKs, but as a consequence, this requires a full mesh topology.

Another veto voting scheme by Kiayias & Yung [26] computes $x_1' + \cdots + x_n'$, where $x_i' = 0$ if $x = 0$, and otherwise x_i' is some random element. Debnath et al. [15] use a similar approach for an MPSI protocol, where x_i' is either 0 or 1. The problem with the former scheme is that a party can tell if it is the only one who submitted a one [8]. The latter also leaks the number of ones in the output.

The MPSI protocol of Miyaji et al. [31] implicitly performs a private AND operation by computing $r_1(\overline{x_1} + \cdots + \overline{x_n})$ homomorphically, and checking if the result is the identity element. The randomness r_1 is generated by the leader to prevent revealing the number of submitted ones. However, since the leader knows this randomization, it can revert it. A secure version of the protocol comes from Bay et al. [4], which computes $(r_1 + \cdots + r_n)(\overline{x_1} + \cdots + \overline{x_n})$. This scheme is conceptually identical to the veto voting scheme by Brandt [9].

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

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

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

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

In this work, we propose private AND and OR protocols that strictly function in a star topology and run in a constant number of rounds. We also provide *composed* versions. Instead of computing the aforementioned circuit for $\mathbf{r} = r_1 + \cdots + r_n$, we compute $\mathbf{r} = r_1r_2 + r_1r_3 + \cdots + r_1r_n$, allowing for further optimizations.

2.2 Multi-party private set intersections

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

Kissner & Song [27] proposed one of the first MPSI protocols in 2005, along with protocols that perform more complex set operations. Their approach involves encoding set elements as the roots of a polynomial. Then, using a threshold version of the Paillier cryptosystem, they add and randomize encrypted polynomials by passing them around the group of parties. The resulting polynomial only reveals the elements that were in each input set, along with a negligible probability of false positives. In Table 1 we refer to the topology as a 'wheel', because next to a channel between each assistant and the leader, each assistant has a channel to one other assistant, creating the shape of a wheel. After Kissner & Song [27], Li & Wu [29] proposed a similar protocol based on Shamir's secret sharing. Later works used the same set encoding [13, 35].

Later, Miyaji & Nishida [31] proposed a Bloom filter-based MPSI that yields the filter representing the intersection, extending the idea of Kerschbaum et al. [25] to multiple parties. They encrypt the Bloom filters using a threshold version of exponential ElGamal.

In 2017, Hazay et al. [23] proposed a protocol that uses the polynomial set encoding. They evaluate the polynomials obliviously using an additive homomorphic threshold cryptosystem, and provide an extension of the protocol secure in the malicious model.

Kolesnikov et al. [28] propose a protocol that uses an OT-based primitive called oblivious programmable pseudo-random functions (OPPRFs), which return a pre-programmed value when queried on elements in the receiver's set. The authors provide a public implementation with which they set speed records, but the protocol requires each pair of parties to interact with each other.

Inbar et al. [24] propose another OT-based protocol that uses garbled Bloom filters. Their protocol is a multi-party version of a similar 2-party protocol [17]. While in a regular Bloom filter one checks if the selected bins are set to 1, in a garbled Bloom filter one performs an XOR operation between those bins to check if the result is some specific value. The protocol requires all parties to interact.

Since then, Abadi et al. [1] proposed an MPSI protocol in the delegated setting, where the majority of computation is outsourced to a semi-honest third party that *cannot collude* with any of the other parties participating in the protocol. Thus, this setting is different from ours, as we are interested in defending against any collusion. For this reason, we exclude that work.

Bay et al. [4, 5] propose multi-party private set operations based on bitsets and Bloom filters using the threshold Paillier cryptosystem, extending the ideas of Ruan et al. [34] and fixing the security problem of Miyaji & Nishida [31]. The bitset-based protocols scale linearly with the size of the universe, while the Bloom filter-based MPSI scales with the number of elements k in exchange for a chance of false positives. Debnath et al. [16] proposed a similar Bloom filter-based protocol using a threshold version of ElGamal.

Finally, Chandran et al. [11] and Nevo et al. [32] published preprints that propose protocols inspired by Kolesnikov et al. [28], using OPPRFs as a core functionality. The work by Nevo et al. is secure in the malicious model and it outperforms both Chandran et al. and Kolesnikov et al. in their experiments. In the case when the collusion resistance $t = n - 1$, their protocol is equivalent to the protocol by Kolesnikov et al. that is secure in the (augmented) semi-honest model. For this reason, we do not compare their concrete performance, but we list their complexities in Table 1.

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

2.3 Multi-party private set unions

Frikken [19] presents one of the first MPSU protocols. Each party represents its set as an encrypted polynomial. In turn, each party receives an encrypted polynomial, multiplies it with their polynomial, and evaluates it for their elements. The parties shuffle and decrypt the resulting ciphertexts so that for each corresponding element, there is only one ciphertext that does not decrypt to 0. Since parties pass their ciphertexts around in a circular fashion, the number of rounds in the protocol scales with the number of parties.

While the work by Shishido & Miyaji [37] refers to a set union in its title, the actual functionality reflects that of a multiset union as it reveals the multiplicity of each element in the resulting set. For this reason, we omit this work from our comparison. The MPSU protocol by Seo et al. [36] does not have this problem, as the parties compute the least common multiple of the polynomials that represent their sets, removing any multiplicities from the polynomial roots. After this operation, the polynomials must be factored. The authors use reversed Laurent series to speed up this step. The protocol revolves around arithmetic on the rational randomized polynomials, which are shared using Shamir's secret sharing. As

Table 1: Comparison of selected works in terms of communication, computation and security using the notation from Table 2. * We adapted these complexities from the original works, see Appendix A & B.

Work			Communication			Computation		Security	
Ref.	Year	Topology	Leader	Assistant	Rounds	Leader	Assistant	Collusion	Assumption
Multi-party Private Set Intersection (MPSI) protocols									
[27]	2005	Wheel	$O(nk)^*$	$O(tk)^*$	$O(n)$	$O(tk^2)^*$	$O(tk^2)^*$	$n-1$	DCR
[23]	2017	Star	$O(nk)$	$O(k)$	$O(1)$	$O(nk^2)^*$	$O(k)^*$	$n-1$	DCR
[28]	2017	Full mesh	$O(nk)$	$O(tk)$	$O(1)$	$O(n)$	$O(t)$	$n-1$	TDP
[24]	2018	Full mesh	$O(nk)^*$	$O(nk)^*$	$O(1)$	$O(nk)^*$	$O(nk)^*$	$n-1$	TDP
[4]	2021	Star	$O(nk)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(nkh)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	DCR
[16]	2021b	Star	$O(nk)^*$	$O(k)^*$	$O(1)$	$O(nkh)^*$	$O(k)^*$	$n-1$	DDH
[11]	2021	Full mesh	$O(nk\log k)$	$O(k\log k)$	$O(1)$	$O(nk)$	$O(k)$	$n-1$	TDP
[32]	2021	Full mesh	$O(k\max(t,n-t))$	$O(k)$	$O(1)$	$O(k(n-t))$	$O(tk)$	$n-1$	TDP
[5]	2022	Star	$O(nk)^*$	$O(k)^*$	$O(1)$	$O(nkh)^*$	$O(k)^*$	$n-1$	DCR
MPSI small		Star	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	ECDDH
MPSI large		Star	$O(nk)$	$O(k)$	$O(1)$	$O(nkh)$	$O(k)$	$n-1$	ECDDH
Multi-party Private Set Union (MPSU) protocols									
[19]	2007	Wheel	$O(nk)^*$	$O(nk)^*$	$O(n)$	$O(nk^2)^*$	$O(nk^2)^*$	$n-1$	DCR
[36]	2012	Full mesh	$O(n^3k^2)$	$O(n^3k^2)$	$O(1)$	$\tilde{O}(n^4k^2)^*$	$\tilde{O}(n^4k^2)^*$	$\lfloor \frac{n}{2} \rfloor$	-
[7]	2016	Full mesh	$O(nk\log k+n^2)$		$O(\log k)$	$O(nk\log k+n^2)$		$\lfloor \frac{n}{2} \rfloor$	-
MPSU small		Star	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$O(1)$	$O(n\lvert\mathcal{U}\rvert)$	$O(\lvert\mathcal{U}\rvert)$	$n-1$	ECDDH
MPSU large		Star	$O(n^2k\log\lvert\mathcal{U}\rvert)$	$O(nk\log\lvert\mathcal{U}\rvert)$	$O(1)$	$O(n^2k\log\lvert\mathcal{U}\rvert)$	$O(nk\log\lvert\mathcal{U}\rvert)$	$n-1$	ECDDH

a result, the protocol is information-theoretically secure, but the multiplication sub-protocol requires all parties to communicate with each other. Consequently, the protocol scales poorly with the number of parties n, and quadratically with the set size k.

Blanton & Aguiar [7] propose multi-party private set and multi-set operations using general multi-party computation techniques based on secret sharing. Their protocols involve sorting the elements, after which there exist efficient algorithms for computing the set operations. While their MPSI protocol does not reach the same level of performance as other solutions, their MPSU protocol outperforms other solutions, running in the order of seconds for small problem instances. The protocol is dominated by the oblivious sorting protocol, but if the parties already sort their sets, the round complexity is $O(\log k)$. In Table 1 we assume this scenario.

Finally, to our knowledge, the only multi-party private set operation relying on elliptic curve cryptography is the union-cardinality protocol by Vos et al. [41]. Their protocol approximates the cardinality of an aggregated Bloom filter by shuffling it and counting the number of ones. The operation differs from an MPSU protocol.

3 PRELIMINARIES

In this section, we give a short introduction about ElGamal over elliptic curves and Bloom filters. The notation that we use here and in the remainder of this paper can be found in Table 2.

3.1 Elliptic curve ElGamal

The ElGamal cryptosystem allows the use of any group $\mathbb{G}$ in which the DDH assumption holds [18]:

Definition 3.1 (Decisional Diffie-Hellman). Given aG and bG for some random $a, b \in \mathbb{Z}_{|\mathbb{G}|}$, abG is computationally indistinguishable from some $R \in_R \mathbb{G}$, which we write as $\mathcal{R}(\mathbb{G})$.

We use the additive notation, as is common for elliptic curve cryptography. For some elliptic curve groups, DDH is assumed to hold: in this work, we use Curve25519 [6]. This curve has a co-factor of 8, which means that the prime order subgroup that we actually use in cryptographic applications is one eighth of the size of the total group. To prevent issues related to this co-factor, we use a highly-optimized encoding that realizes a true prime-order group [21, 39], eliminating the co-factor. Additionally, this technique allows for faster equality checks [40]. Compressed elements are only 32 bytes in size, so a single ElGamal ciphertext takes 64 bytes.

Table 2: Description of symbols in this work.

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

3.2 Bloom filters

A Bloom filter is an approximate data structure for representing sets. It consists of m bins initially set to 0. When inserting an element into the Bloom filter, the values of several bins selected by h hash functions are changed to 1. We denote such a hash function by $\mathcal{H}_i$, where i is the seed. The function maps elements uniformly to $\{0, \ldots, m-1\}$. In our implementation, we use the xxh3 hash function [14], which is a fast statistical hash function. We map the results to the correct range using a modulo operation. Note that the hash function does not have to be cryptographically secure, as the security of Bloom filter-based private set operations does not rely on the security of the hash function. Algorithm 1 describes how to create the Bloom filter of a set X.

Algorithm 1 Creates a Bloom filter for set X

```
1: procedure CreateBF($X, m, h$)
2:     $\hat{X} \leftarrow [0, \ldots, 0]$                ▷ Bit vector of length $m$
3:     for $x \in X$ do
4:         for $i = 1, \ldots, h$ do
5:             $\hat{X}[\mathcal{H}_i(x)] \leftarrow 1$
6:     return $\hat{X}$
```

To check whether a Bloom filter contains a given element, we check whether the corresponding bins chosen by the hash functions are indeed all set to 1. This operation is approximate because it might falsely conclude that an element is contained in the Bloom filter when the bins were set to 1 by coincidence through the insertion of other elements. Fortunately, this problem is well-studied, and Goel & Gupta [20] provide an upper bound for the probability of such a false positive ε when N elements have been inserted in a Bloom filter:

$$\varepsilon \leq \left(1 - e^{-\frac{h(N+0.5)}{m-1}}\right)^h . \tag{2}$$

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

$$\min_{h \geq 1} \left\lceil -\frac{h(N+0.5)}{\ln 1 - \sqrt[h]{\varepsilon}} \right\rceil + 1 . \tag{3}$$

Finally, one can combine multiple Bloom filters to construct a filter representing the intersection or union using logical operations. For example, computing an AND operation between the bins of two respective filters yields a third filter representing their intersection, where ε is equal to that of the original Bloom filters [33].

4 PRIVATE ORS & ANDS

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

$$x_1 \wedge \cdots \wedge x_n = \overline{\overline{x_1} \vee \cdots \vee \overline{x_n}} . \tag{4}$$

4.1 Protocol description

Before executing any of our protocols, the parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ execute a short distributed setup operation over public authenticated channels. We assume that the identities of the parties are known, and a leader has been chosen beforehand. The parties aim to generate n secret ElGamal keys sk_i and a corresponding public key pk. Each party chooses their secret key randomly $sk_i \in_R \mathbb{Z}_q$. Then, they send $pk_i \leftarrow sk_i\, G$ to the leader $\mathcal{P}_1$, where G is the public generator element. G is typically chosen by the same authority that chooses group $\mathbb{G}$. Eventually, the leader computes and broadcasts public key $pk \leftarrow \sum_{i=1}^{n} pk_i$. This setup operation can also take place in a distributed fashion, where each party broadcasts pk_i. The result is a threshold version of ElGamal that requires all parties to decrypt, denoted by (n, n)-ElGamal. One can also use a custom (t, n) setup, although this would lower the protocol's collusion resistance to t. We present our private `OR` operation in Protocol 4.1.

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

Private `OR` protocol

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Protocol 4.1: Our multi-party private `OR` protocol.

A second optimization that is particularly relevant for our *composed* private logic comes by letting the leader randomize the summation in step 2 rather than step 3 like the assistants. In doing so, the aggregated ciphertext does not reveal its constituent ciphertexts without having to perform a rerandomization operation by adding a fresh encryption of O. Finally, instead of performing a full ElGamal decryption, the leader sums all σ_i and checks if it equals $\overline{\beta}$, saving a point subtraction in the process. We also optimize point compression when performing multiple `OR` operations in parallel. We elaborate on this in Section 4.5, where we summarize the cost in elliptic curve operations.

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

4.2 Composed logic

In the previous protocol, each party $\mathcal{P}_i$ contributes one bit x_i and the leader outputs $x_1 \vee \cdots \vee x_n$. The parties can also perform multiple parallel operations, where each party submits k bits, and the leader outputs the k results of the `OR` operations. An interesting case arises when the leader wants to compute an `OR` operation over bits of its choosing, which is a generalization of the former functionality. In this section, we propose Protocol 4.2 for this purpose. We show that this protocol is also privacy-preserving, and that the assistants do not learn the pattern of bits selected by the leader.

Private composed `OR` protocol

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

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

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

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

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

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

The parties continue the remaining steps in the same way as they did in Protocol 4.1, albeit as k parallel runs over $\langle \alpha^t, \beta^t \rangle$ for $t = 1, \dots, k$. The leader outputs results z^t.

Protocol 4.2: Our composed `OR` protocol, where the leader chooses the bits to perform the logic over.

More formally, we let the leader select an evaluation pattern f_i^t that is a subset of $\{1, \ldots, k\}$, representing the index of the bits of party $\mathcal{P}_i$ that should be incorporated in the tth logic operation. For example, $\forall_i \; f_i^1 = \{1, \ldots, k\}$ would denote that the first logic operation incorporates all parties' bits, so the leader would learn the OR over *all* submitted bits. If $f_1^t = \emptyset$, this tth evaluation does not incorporate any of the leader $\mathcal{P}_1$'s bits. The equivalent private composed AND protocol achieved through DeMorgan's law forms the basis for the Bloom filter-based MPSI protocol in Section 6.

4.3 Correctness

Protocol 4.2 must output $x_1 \vee \cdots \vee x_n$ with overwhelming probability, which also implies the correctness of Protocol 4.1. In other words, the output is only 0 when all inputs were 0, otherwise it is 1:

Theorem 4.1. *With overwhelming probability, $z^t = 0$ if and only if $\forall_{i=1}^n \forall_{j \in f_i^t} \; x_j^i = 0$.*

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

$$\overline{\beta}^t = \sum_{i=1}^n \sigma^t = \sum_{i=1}^n sk_i \, \overline{\alpha}^t = sk \, \overline{\alpha}^t \; , \tag{5}$$

where $sk = \sum_{i=1}^n$, the underlying key of the threshold cryptosystem. After substituting steps 3 and 4, we get:

$$\sum_{i=2}^n r_i^t \beta^t = sk \sum_{i=2}^n r_i^t \alpha^t \; . \tag{6}$$

Now, we show that $\beta^t = sk \, \alpha^t$ by substituting steps 1 and 2 and using the fact that $pk = sk \, G$:

$$r_1^t \sum_{i=1}^n \sum_{j \in f_i^t} y_i^j pk = sk \, r_1^t \sum_{i=1}^n \sum_{j \in f_i^t} y_i^j G \; , \tag{7}$$

$$r_1^t \sum_{i=1}^n \sum_{j \in f_i^t} y_i^j pk = r_1^t \sum_{i=1}^n \sum_{j \in f_i^t} y_i^j pk \; . \tag{8}$$

Next, we prove the necessary condition, so $z^t = 1$ when $\exists_{i=1}^n \exists j \in f_i^t \; x_i^j = 1$ with overwhelming probability. Since some $x_i^j = 1$, the corresponding y_i^j in the LHS of Equation 8 will be replaced by some $y_i'^j \in_R \mathbb{Z}_q$. As a result the equality only holds with a uniformly random probability of $\frac{1}{q}$, which is negligible. □

4.4 Privacy

We now provide a simulation-based proof to formally show that our protocols are indeed private in the semi-honest model, following the requirements for a deterministic functionality as described in [30]. Notice that when $f_i^t = \{t\}$ for $t = 1, \ldots, k$ and $i = 1, \ldots, n$, Protocol 4.2 reduces down to k parallel executions of Protocol 4.1. In other words, proving security of the first implies security of the latter. For this reason we only provide such a proof for Protocol 4.2. We pose that our protocol is secure against $n - 1$ colluding parties, so two cases arise:

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

Multiple parts of our proof rely on the following lemma:

Lemma 4.2. *Consider group $\mathbb{G}$ for which the decisional Diffie-Hellman (DDH) assumption is assumed to hold. Given element $G' \in \mathbb{G}$, unknown randomness $r \in_R \mathbb{Z}_q$, and $p = sG'$ for the unknown $s \in \mathbb{Z}_q$, it holds that:*

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

Proof. The first term is statistically indistinguishable from randomness, since $r \in_R \mathbb{Z}_q$, so $rG' \overset{s}{\equiv} \mathcal{R}(\mathbb{G})$. The second term rsG' is computationally indistinguishable by the DDH assumption in Definition 3.1 by taking $a \leftarrow r$, $b \leftarrow s$, $G \leftarrow G'$ (so $bG \leftarrow p$). So, $rsG' \overset{c}{\equiv} \mathcal{R}(\mathbb{G})$. □

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

Theorem 4.3. *For a set of colluding parties $C \subseteq \{2, \ldots, n\}$ there exists a simulator $\mathcal{S}_1$ so that:*

$$\mathcal{S}_1() \overset{c}{\equiv} \bigcup_{c \in C} \text{view}_c(x_c) \; . \tag{9}$$

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

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

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

$$\langle y_i^j G, \; y_i'^j \, pk \rangle \overset{s}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle \; , \tag{10}$$

$$\langle y_i^j G, y_i^j pk \rangle \overset{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle \; . \tag{11}$$

Equation 10 holds because y_i^j and $y_i'^j$ are sampled randomly, covering the case where $x_i^j = 1$. Next, Equation 11 holds by Lemma 4.2 where $s \leftarrow sk$, $G' \leftarrow G$, and $r \leftarrow y_i^j$, covering the case $x_i^j = 0$.

For step 2 of the protocol we must show $\langle \alpha^t, \beta^t \rangle \overset{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$. To simplify notation, we say that:

$$v_i'^j = \begin{cases} y_i^j & \text{if } x_i^j = 0 \\ y_i'^j & \text{if } x_i^j = 1 \end{cases} \; . \tag{12}$$

Then, our former statement holds by Lemma 4.2, where:

$$r \leftarrow r_1^t, \quad G' \leftarrow \sum_{i=1}^n \sum_{j \in f_i^t} \alpha_i^j, \quad s \leftarrow sk \sum_{i=1}^n \sum_{j \in f_i^t} \frac{v_i^j}{y_i^j} \; ,$$

and s is unknown because it is a factor of the unknown key sk.

For step 3 of the protocol we must show $\langle \overline{\alpha}_i^t, \overline{\beta}_i^t \rangle \overset{c}{\equiv} \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$. Again, this holds by Lemma 4.2, where $r \leftarrow r_i$, $G' \leftarrow r_1 G'$, and $s \leftarrow r_1 s$. Here we reuse the values from our previous argument.

For step 4 of the protocol, the simulator executes the step as usual for $\overline{\alpha}^t$. For corrupted parties $c \in C$, the simulator samples $\langle \overline{\alpha}_c^t, \overline{\beta}_c^t \rangle \leftarrow \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$, which is statistically indistinguishable from the actual view.

For step 5 of the protocol we must show that $\sigma_i \overset{c}{\equiv} \mathcal{R}(\mathbb{G})$. This holds because sk_i is unknown to the corrupted parties and is sampled uniformly from $\mathbb{G}$. $\square$

We also prove that a simulator exists that generates a view for a colluding leader and up to $n-2$ colluding assistants in Theorem C.1 in Appendix C, which is indistinguishable from their own, given these parties' inputs and the protocol's output. Moreover, apart from being private on paper, we show that our implemented protocol indeed runs in constant-time in the next subsection.

4.5 Efficiency

When presenting the protocol, we hinted that we can optimize the point compression step when performing multiple OR operations in parallel. The reason is that the compression operation is batchable, if we allow the compressed point to be doubled. Note that for steps 1 to 4 of this protocol, this has no impact on the encrypted value, but for steps 5 and 6 there will be a factor 2 discrepancy between σ and β. Fortunately, one can compensate for this in the secret keys. In short, after generating the public key pk, each party divides their secret key by 4 offsetting the factor induced by batch-compressions, so the actual key becomes $sk \leftarrow \frac{1}{4} sk_i$.

We summarize the computational cost of our protocol in Table 3 as the number of elliptic curve operations performed, and compare it against the naive approach of computing $(r_1 + \cdots + r_n)(x_1 + \cdots + x_n)$ using additively homomorphic encryption. Communication-wise, the leader must send four compressed points to each assistant, while each assistants sends five compressed points to the leader. Given our choice of Curve25519, this means that for one private logic operation, the leader sends $128(n-1)$ bytes and an assistant sends 160 bytes. Asymptotically, both the computational and communicational complexities are $O(n)$ for the leader and $O(1)$ for an assistant, although the number of point multiplications stays constant, regardless of n. As described in Section 2.1, one can also perform this arithmetic circuit using secret sharing. However, the total communication cost would scale quadratically with n since it requires all parties to communicate. We analytically compare the communication cost of this approach with our protocol in Figure 1, where shares are 5 bytes in size and the parties compute the multiplication using pre-distributed Beaver triplets. For $n \geq 10$, the communication overhead of this approach would exceed that of our protocol.

Table 3: EC operations for our private OR protocol on one bit.

		Addition	Fixed mult.	Variable mult.
Naive	Leader	$5n-1$	4	3
	Assistant	1	2	3
Ours	Leader	$5n-7$	2	3
	Assistant	-	2	3

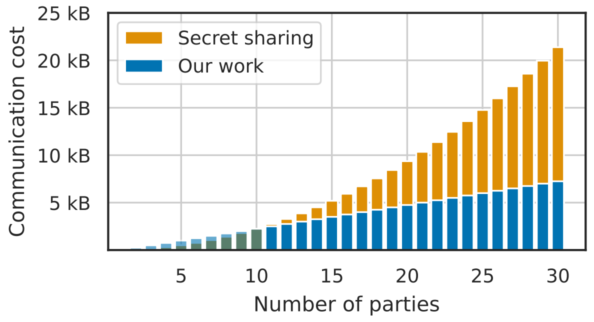

Figure 1: Communication in one private OR computation

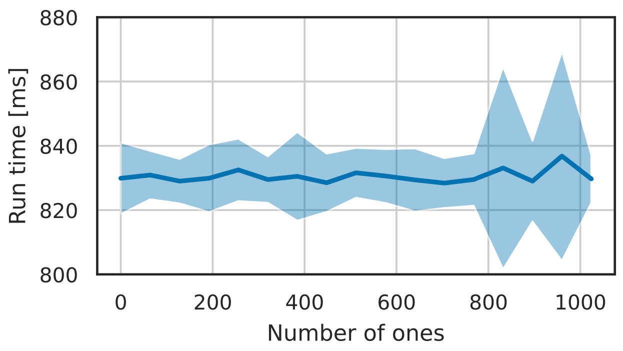

Figure 2: The run time of 1024 private ORs is constant w.r.t. the number of 1s. The shaded area is the 99% confidence interval.

While we described our private OR protocol to function on single-bit inputs, the parties can perform this operation on many bits in parallel. We explicitly use this technique to perform efficient MPSI and MPSU protocols. We provide an open-source implementation of such parallel ORs and ANDs. In our implementation, we do not simulate communication delays, but we do route the messages through Unix streams. Figure 2 shows the run time of our private OR protocol on 1024 bits in parallel for an increasing number of inputs that are 1. The figure underlines that the run time of our protocol indeed does not depend on the input. In the remainder of this work we perform all our experiments on a Unix machine with 30 virtual Intel® Xeon® Cascade Lake CPUs at 3100 MHz. We assign each party one execution thread to run on. The machine also has 120 GB of memory allocated to it, but in our experiments we only use a fraction of this. All our implementations are written in Rust.

5 PRIVATE SET OPERATIONS FOR SMALL UNIVERSES

One approach for computing a set intersection is to check for each element in the universe that it is present in all the sets. For the union, one checks if an element occurs in at least one of the sets. This is the idea behind the protocols of Bay et al. [4], which use the bitset representation. A bitset represents a set as a vector of bits corresponding to each element in the universe. When an element is in the set, the corresponding bit is set to 1; otherwise, it is 0. Computing the intersection then constitutes an element-wise AND operation, and the union constitutes an OR operation. In this section, we instantiate such bitset-based protocols using our private AND and OR protocols, as presented in Protocol 5.1.

MPSI protocol for small universes

(1) All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the bitset $\hat{X}_i$ of their set X_i:

$$\hat{X}_i[j] = \begin{cases} 1 & \text{if } j \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases}$$

(2) All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ take part in a private AND protocol on $\hat{X}_i$, so the leader $\mathcal{P}_1$ retrieves $\hat{Z}$.

(3) The leader $\mathcal{P}_1$ returns the result Z:

$$Z = \{j \in \mathcal{U} \mid \hat{Z}[j] = 1\}$$

Protocol 5.1: A bitset-based MPSI protocol. An MPSU protocol would use a private OR protocol instead.

This MPSI protocol inherits its security properties from the private AND protocol since the private data is only accessed during the execution of that sub-protocol. The same holds for the MPSU protocol. The efficiency of these protocols is also decided by the private logic protocols as they dominate the computation required. Since the parties perform one private logic operations for each element in the universe, the computational complexity is $O(n\,|\mathcal{U}|)$ for the leader and $O(|\mathcal{U}|)$ for an assistant.

We experimentally compare the run time of this MPSI protocol with the implementation by Bay et al. [4] using the same setup as before. Note, however, that the original work uses a 1024 bit modulus to instantiate the Paillier cryptosystem, which corresponds to a legacy security strength of 80 bits. To ensure a fair comparison and cryptographic security, we instead choose a 3072 bit modulus as per NIST's standard [3], corresponding to 128 bits of security. The results of this experiment are in Figure 3.

For $n = 2$ and $|\mathcal{U}| = 256$, this protocol outperforms the implementation by Bay et al. by almost two orders of magnitude. While, the implementation by Bay et al. seems to be hardly affected by the number of parties in Figure 3, this is an artifact of the logarithmic axis. The absolute increase in run time when the number of parties grows is comparable to ours: for $|\mathcal{U}| = 256$, Bay et al. takes 9.47, 9.62, 9.73 seconds for $n = 2, 5, 10$, while this protocol takes 0.12, 0.20, 0.34 seconds.

Finally, notice that Protocol 5.1 can actually be further optimized by instantiating it with a *composed* AND protocol, in exchange for leaking the leader's set size. The initial steps of this protocol would still scale with $|\mathcal{U}|$, but the remaining steps would scale with k.

6 PRIVATE SET INTERSECTIONS FOR LARGE UNIVERSES

In practice, the size of a party's set is often significantly smaller than the size of the universe, meaning $k \ll |\mathcal{U}|$. In this section, we

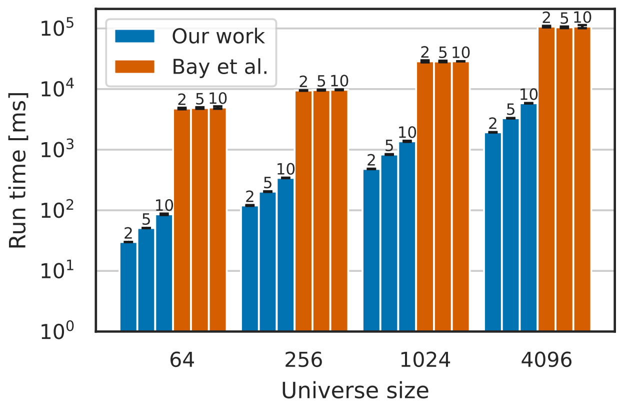

Figure 3: Run time comparison between our MPSI protocol and Bay et al. The numbers over the bars indicate the number of parties n and the error bars the 99% confidence interval.

instantiate an MPSI protocol with our private logic that scales only with k rather than $|\mathcal{U}|$ in exchange for a false positive rate ε.

The difference between this protocol and the MPSI protocol for small universes is that parties represent their set as a Bloom filter rather than a bitset. As we discuss in Section 6.3, this causes the protocol to scale independently of the size of the universe. We aggregate the Bloom filters similarly to bitsets, but using our private composed AND protocol. We adapt this idea from Bay et al. [5], although the same idea has been applied more often in MPSI protocols, such as by Miyaji & Nishida [31] and Debnath et al. [16], but these suffer from security flaws as described in Section 2.1.

We present the updated Protocol 6.1, where $X_1[t]$ represents the tth element of the leader's set. Here, the parties engage in a private *composed* logic protocol to ensure that no information is leaked from the resulting Bloom filter. In other words, the leader computes an AND operation between the bins corresponding to each of its elements, essentially performing at most k private membership checks. The privacy of the leader's elements in turn relies on the assistants not learning the evaluation pattern f_i^x. We provide more details on the security properties of the protocol in Section 6.2.

MPSI protocol for large universes

(1) All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the Bloom filter $\hat{X}_i$ of their set X_i:

$$\hat{X}_i = \textsc{CreateBF}(X, m, h)$$

(2) All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ take part in a private composed AND protocol on $\hat{X}_i$ with $f_i^t = \{\mathcal{H}_j(X_1[t]) \mid t = 1, \ldots, k,\ j = 1, \ldots, h\}$, so that the leader $\mathcal{P}_1$ retrieves $\hat{Z}$.

(3) The leader $\mathcal{P}_1$ returns the result Z:

$$Z = \left\{X_1[t] \mid \hat{Z}[t] = 1 \text{ for } t = 1, \ldots, k\right\}$$

Protocol 6.1: A Bloom filter-based MPSI protocol.

6.1 Correctness

There are three properties that must hold with overwhelming probability for the protocol to be correct:

- When all parties have an element $X_1[t] = x$ in their set it must hold that $\hat{Z}[t] = 1$.
- When the leader has an element $X_1[t] = x$ but at least one other party does not have x in their set $\hat{Z}[t] = 0$ must hold.
- When an element $x \notin X_1$ it must hold that $x \notin Z$.

For the first case, notice that $\hat{X}_i[j] = 1$ for all parties $i = 1, \ldots, n$ and $j = \mathcal{H}_1(x), \ldots, \mathcal{H}_h(x)$ after the parties create their Bloom filters. Since $X_1[t] = x$, it holds that:

$$\hat{Z}[t] = \bigwedge_j \hat{X}_1[j] \wedge \cdots \wedge \hat{X}_n[j] = 1 \,. \tag{13}$$

In the second case, there is a probability ε that all bins corresponding to the element x are set to 1. However, as explained in Section 3.2, we can choose parameters so that ε is negligible. Then, with overwhelming probability, there is at least one party i' and Bloom filter bin j' for which it holds that $\hat{X}_{i'}[j'] = 0$. Now:

$$\hat{Z}[t] = \cdots \wedge \hat{X}_{i'}[j'] \wedge \cdots = 0 \,. \tag{14}$$

In the last case, for each $t = 1, \ldots, k$ it also holds with overwhelming probability that at least for one bin $\hat{X}_1[j'] = 0$ with $j' \in \{\mathcal{H}_1(x), \ldots, \mathcal{H}_h(x)\}$. As a result:

$$\hat{Z}[t] = \cdots \wedge \hat{X}_1[j'] \wedge \cdots = 0 \,. \tag{15}$$

6.2 Privacy

Since the leader chooses the evaluation pattern to reflect the operation of checking whether an element is contained in the Bloom filter, the protocol's entire security again relies on the private logic primitive. Note that if we had not used the *composed* AND protocol and therefore exposed the entire resulting Bloom filter, the protocol would leak information that is inherent to the way Bloom filters combine under intersections. This problem was also hinted at in previous works [10, 17]. This leakage arises because it does **not** necessarily hold that:

$$\begin{gathered}\textsc{CreateBF}(X_1 \cap \cdots \cap X_n, m, h) = \\ \textsc{CreateBF}(X_1) \texttt{ AND } \ldots \texttt{ AND } \textsc{CreateBF}(X_n) \,,\end{gathered} \tag{16}$$

which occurs when 1s in the input Bloom filters not belonging to the actual intersection align by accident.

6.3 Efficiency

We rewrite Equation 2 to isolate m:

$$\varepsilon \leq \left(1 - e^{-\frac{h(N+0.5)}{m-1}}\right)^h \tag{17}$$

$$m \geq -\frac{h(N+0.5)}{\ln 1 - \sqrt[h]{\varepsilon}} + 1 \tag{18}$$

$$m \geq \left(\frac{-h}{\ln 1 - \sqrt[h]{\varepsilon}}\right) N - \left(\frac{0.5h}{\ln 1 - \sqrt[h]{\varepsilon}}\right) + 1 \tag{19}$$

As such, for a constant h and ε, it holds that the minimal number of bins m scales linearly with N. In short, $m = O(N)$. In practice we choose h depending on ε and N to choose the smallest m in that situation. In the protocol, the private AND operations dominate the run time. Each party takes part in h of such operations over m bits, so the computational complexity for the leader is $O(nmh)$, and for an assistant is $O(m)$. Since $m = O(N)$ and $N = k$, we write the final complexities as $O(nkh)$ for the leader and $O(k)$ for an assistant.

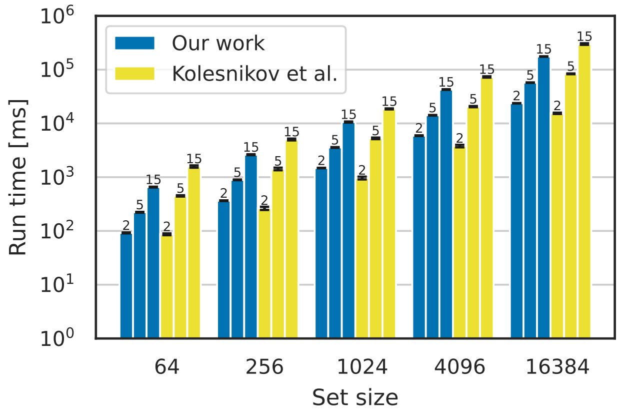

Figure 4: Run time comparison of the MPSI protocol for large universes $\varepsilon = 0.1\%$ and Kolesnikov et al. [28]. The numbers over the bars indicate the number of parties n and the error bars the 99% confidence interval. Our work is faster for $n > 2$.

6.4 Results

Since we expect to see the same advantage over the work by Bay et al. [5] as we saw for bitsets, we only compare our protocol against the work by Kolesnikov et al. [28]. The experimental setting in the original implementation differs from ours: each party has access to $n - 1$ threads, but in our setting, each party is allocated a single thread. Since [28] was executed on a powerful machine with two 36-core CPUs, this makes the run time scale linearly with the number of parties n. In the same way, for a sufficiently large number of threads, our protocol runs independently of set size k, but we do not consider this a realistic setup. For these reasons, we use an alternative implementation in our experiments that spawns a single thread for each party. As a second benefit, this implementation is also written in Rust, which we argue constitutes a fair comparison. In both protocols, parties communicate via Unix streams.

We provide results of our work and the work by Kolesnikov et al. [28] in Figure 4, comparing run time for an increasing set size k as well as an increasing number of parties n when $\varepsilon = 0.1\%$. While their protocol is faster for $n = 2$, our protocol is faster for larger values of n. Such an error rate may be permissible in situations where false positives are not detrimental. For example when multiple parties compare sets of potentially malicious IP addresses to highlight a subset of suspicious addresses to investigate further.

Of course, if one chooses an arbitrarily large ε, our protocol always outperforms the protocol by Kolesnikov et al., but the resulting set will be almost random. Instead, we study what value ε should take to match their performance. To do so, we run our MPSI protocol for increasing n on the error rates $\varepsilon = 2^{-1}, 2^{-2}, \ldots, 2^{-25}$ and compare at what point our protocol runs faster. We plot the results in Figure 5. One can observe that even when we tolerate a false positive rate $\varepsilon = 0.01\%$, our protocol is faster for $n \geq 3$, denoted by the arrow.

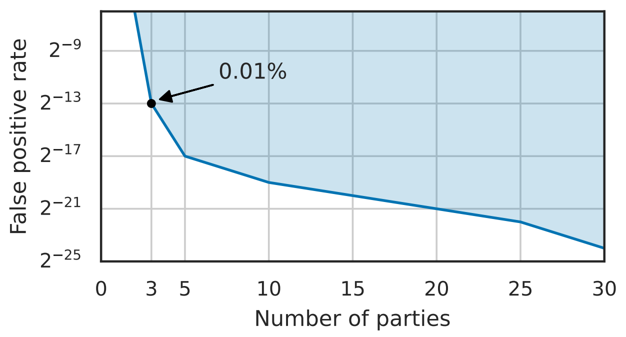

Figure 5: The error rate of our protocol when we choose the parameters so that the run time is equal to that of Kolesnikov et al. [28]. The shaded area represents the parameters for which our protocol is faster, e.g. at $\varepsilon \geq 0.01\%$ for $n = 3$.

7 PRIVATE SET UNIONS FOR LARGE UNIVERSES

When $k \ll |\mathcal{U}|$, a bitset representation would be filled almost entirely with 0s, but we are only interested in searching for the 1s. To prevent wasting computations on this sparse vector, we propose a divide-and-conquer algorithm that isolates these 1s. The intuition is as follows: Each party splits their bit vector into D partitions and locally computes the logical OR of the bits in each partition. After that, the parties take part in our private OR protocol on the aggregated bits. In this way, they can discard all partitions for which the result is 0, as none of the original bits in the partition is a 1. As a concession, this approach allows the assistants to learn information about the final set union, but they do not learn more than the leader. The parties repeat this process until a partition only contains one bit. We provide an example of this process in Figure 6. Instead of running 27 private OR protocols, the example only requires 15 runs. For larger universes, the difference will be even greater. We provide a formal description in Protocol 7.1.

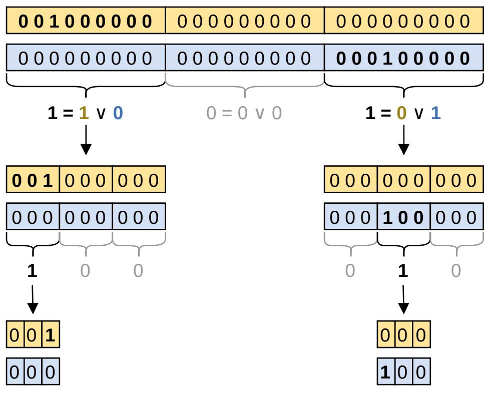

Figure 6: Example of the divide-and-conquer approach with two parties, $N = 27$, $T = 2$ and $D = 3$.

MPSU protocol for large universes

(1) All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ compute the bitset $\hat{X}_i$ of their set X_i:

$$\hat{X}_i[j] = \begin{cases} 1 & \text{if } j \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases}$$

(2) All parties $\mathcal{P}_i$ for $i = 1, \ldots, n$ execute DIVIDEANDCONQUER$(\hat{X}_i, D)$, computing the bitwise OR between their bitsets so that the leader $\mathcal{P}_1$ retrieves $\hat{Z}$.

(3) The leader $\mathcal{P}_1$ returns the result Z:

$$Z = \{j \in \mathcal{U} \mid \hat{Z}[j] = 1\}$$

Protocol 7.1: Our multi-party private set union protocol, relying on Algorithms 2 and 3 from Appendix D.

7.1 Choosing the number of divisions

Consider a vector with $N = 16$ bits, of which only one is 1. When the number of ones $T = 1$ and the number of divisions $D = 2$, we must perform four iterations of the divide-and-conquer approach to reach partitions containing only one bit. Notice that since $T = 1$, we always discard all but one division. In the general case, we require $\log_D N$ iterations of D runs of the OR protocol, so we require $D \log_D N$ private ORs in total. When $T > 1$, we can extend this to a loose upper bound of $TD \log_D N$. After all, in the worst case, each 1 ends up in a separate partition at each iteration.

While the optimal choice of D is Euler's constant e (see Appendix D), this is not practically attainable:

- For ease of implementation, we want D to be an integer.
- For a small D and large $|\mathcal{U}|$, we require many iterations.
- For a constant number of rounds, D cannot be constant.

Instead of choosing $D = e$, we select a suitable number of divisions based on a specified maximum number of iterations R. Let us define the function ORs that returns the expected number of private ORs; then choosing D comes down to a minimization problem:

$$\min_{D \geq \sqrt[R]{N}} \text{ORs}(T, N, D) \,. \tag{20}$$

The reason that $D \geq \sqrt[R]{N}$ is that splitting a vector into D parts for R iterations allows us to reach exactly $N = D^R$ partitions of size 1 in the final round.

To describe ORs, we analyze the cost of each iteration. The first iteration requires $C_0 = D$ OR operations. We can view this as a balls and bins problem, in which T balls are divided among D bins. Only those bins that contained at least one ball continue in the protocol. Assuming that such a bin has a limitless capacity, we express the expected number of filled bins by:

$$\text{spread}(\text{balls}, \text{bins}) = \text{bins}\left(1 - \exp -\frac{\text{balls}}{\text{bins}}\right) \,. \tag{21}$$

We derive this function from Equation 1 in [41] when $h = 1$. Given this equation, we express the expected cost of iteration i by $C_i = D \text{ spread}(T, C_{i-1})$, since each iteration splits the number of filled bins of the previous iteration again into D partitions. This expected cost holds for all but the last iteration, where there may not be as many bits N as the number of partitions we can form. We compensate for this by computing the expected number of partitions that remain as:

$$B = \frac{N}{N - D^{\lfloor \log_D N \rfloor}} \ . \tag{22}$$

Now, the expected number of private OR operations is:

$$\mathrm{ORs}(T, N, D) = B \, \mathrm{spread}(T, C_{\lfloor \log_D N \rfloor} B) + \sum_{i=0}^{\lfloor \log_D N \rfloor} C_i \ , \tag{23}$$

As mentioned before, in the final iteration, we may form more partitions than there are bits. This is only the case when $\log_D N$ is not a whole number. As such, we reduce the optimal choice for D that leads to the least OR operations to searching for $D = \sqrt[j]{N}$, where:

$$\min_{j=2,\ldots,R} \mathrm{ORs}(T, N, \sqrt[j]{N}) \ . \tag{24}$$

Here, j is the required number of iterations which is less than or equal to the pre-defined maximum R. To determine j, we evaluate all possible values $j = 2, \ldots, R$. For the MPSU protocol, the other parameters are $T = nk$ and $N = |\mathcal{U}|$. We note that we are optimizing for the average case, where 1s are randomly distributed, but ideally the 1s are bundled together. So, if there is some structure in the set elements, one can achieve performance gains by increasing the probability that ones end up together in the same partition.

7.2 Privacy

We argue that assuming our MPSU protocol for small universes is privacy-preserving, the same holds for this protocol. Consider two parties $\mathcal{P}_1$ and $\mathcal{P}_2$ with bitsets:

$$\hat{X}_1 = \left[x_1^1, x_1^2, \ldots, x_1^{|\mathcal{U}|}\right] \ , \quad \hat{X}_2 = \left[x_2^1, x_2^2, \ldots, x_2^{|\mathcal{U}|}\right] \ .$$

Then, in our previous MPSU protocol, these parties would learn:

$$\left[(x_1^1 \vee x_2^1), (x_1^2 \vee x_2^2), \ldots, (x_1^{|\mathcal{U}|} \vee x_2^{|\mathcal{U}|})\right] \ . \tag{25}$$

Let us say that the first two bits end up in one partition, then the parties learn:

$$(x_1^1 \vee x_1^2) \vee (x_2^1 \vee x_2^2) = (x_1^1 \vee x_2^1) \vee (x_1^2 \vee x_2^2) \ . \tag{26}$$

So the two parties only learn the logical OR of bits they would have learned regardless in our previous MPSU protocol. In other words, the information they learn is a function of the output, rather than their private inputs. Regarding timing attacks, the divide-and-conquer approach does not strictly run in constant time. Instead, the run time is correlated with the size of the output set.

7.3 Efficiency

An upper bound for the number of OR operations, which dominate the performance of the protocol, is $nkD \log_D |\mathcal{U}|$. So, when D is constant, both the computational and communication complexities are $O(n^2 k \log |\mathcal{U}|)$ for the leader, and $O(nk \log |\mathcal{U}|)$ for an assistant. The concrete run time scales with the size of the resulting union.

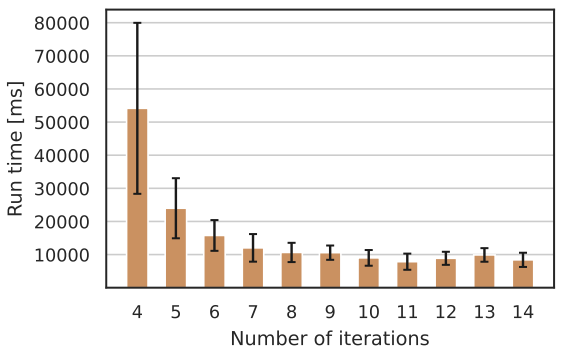

Figure 7: Run time of our MPSU protocol for large universes when the number of iteration increases. Here, $n = 5$, $k = 32$ and $|\mathcal{U}| = 2^{32}$. The error bars denote the standard deviation. The decrease in run time tapers off around $R = 8$.

7.4 Results

To the best of our knowledge, there are no public implementations of other MPSU protocols. Comparing against the MPSU protocol for small universes is also infeasible, as the run time would exceed hours for larger universes. Instead, we evaluate the run time of this protocol for a growing maximum number of iterations R in Figure 7, where the universe has the size of the IPv4 space. Since the decrease in run time tapers off at 8 iterations, we consider this the optimal choice in this instance. In our experiment we do not simulate additional communication delays, however, one might trade-off this delay with a party's computational effort. We note that the actual run time of the protocol does not scale linearly with set size, as the probability increases for two elements to map to the same partition, allowing the protocol to discard more partitions.

8 CONCLUSION

In this work, we instantiate existing MPSI and MPSU protocols with elliptic curve-based private logic protocols to perform fast set operations on any size of universe. Our novel private logic protocols may also be of independent interest. Most previous MPSI and MPSU protocols either use significantly slower integer-based homomorphic encryption or oblivious transfers that require interactions between all parties. Our protocols, however, enjoy the low computational cost of elliptic curve operations and function in the star topology. Moreover, we propose a novel MPSU protocol for large universes that uses a divide-and-conquer approach to significantly reduce computation at the cost of more interactions. Still, it remains an open question to design an exact elliptic curve-based MPSI or MPSU that does not depend on the size of the universe.

We open-source a proof-of-concept implementation of all protocols and compare it against the state of the art. We also demonstrate that the protocols' run times are constant and we underline their security using a simulation-based proof. The protocols are fast enough to be used in practice; the MPSI protocol for small universes is two orders of magnitude faster than the protocol by Bay et al. [4], and for a false positive rate of $\varepsilon \leq 0.01\%$ the MPSI protocol for large universes consistently outperforms the protocol by Kolesnikov et al. [28] when three or more parties are involved.

ACKNOWLEDGMENTS

We would like to thank the creators of the curve25519-dalek, subtle and swanky Rust libraries.

REFERENCES

[1] Aydin Abadi, Sotirios Terzis, and Changyu Dong. 2020. Feather: Lightweight Multi-party Updatable Delegated Private Set Intersection. *IACR Cryptol. ePrint Arch.* (2020), 407. https://eprint.iacr.org/2020/407

[2] Samiran Bag, Muhammad Ajmal Azad, and Feng Hao. 2019. PriVeto: a fully private two-round veto protocol. *IET Inf. Secur.* 13, 4 (2019), 311–320. https://doi.org/10.1049/iet-ifs.2018.5115

[3] Elaine Barker. 2020. *Recommendation for Key Management: Part 1 – General.* Technical Report SP 800-57 Part 1 Rev. 5. NIST.

[4] Asli Bay, Zeki Erkin, Mina Alishahi, and Jelle Vos. 2021. Multi-Party Private Set Intersection Protocols for Practical Applications. In *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021*, Sabrina De Capitani di Vimercati and Pierangela Samarati (Eds.). SCITEPRESS, 515–522. https://doi.org/10.5220/0010547605150522

[5] Aslí Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. 2022. Practical Multi-Party Private Set Intersection Protocols. *IEEE Trans. Inf. Forensics Secur.* 17 (2022), 1–15. https://doi.org/10.1109/TIFS.2021.3118879

[6] Daniel J. Bernstein. 2006. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3958)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.). Springer, 207–228. https://doi.org/10.1007/11745853_14

[7] Marina Blanton and Everaldo Aguiar. 2016. Private and oblivious set and multiset operations. *Int. J. Inf. Sec.* 15, 5 (2016), 493–518. https://doi.org/10.1007/s10207-015-0301-1

[8] Colin Boyd, Kristian Gjøsteen, Clémentine Gritti, and Thomas Haines. 2019. A Blind Coupon Mechanism Enabling Veto Voting over Unreliable Networks. In *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11898)*, Feng Hao, Sushmita Ruj, and Sourav Sen Gupta (Eds.). Springer, 250–270. https://doi.org/10.1007/978-3-030-35423-7_13

[9] Felix Brandt. 2005. Efficient Cryptographic Protocol Design Based on Distributed El Gamal Encryption. In *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3935)*, Dongho Won and Seungjoo Kim (Eds.). Springer, 32–47. https://doi.org/10.1007/11734727_5

[10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas A. Dimitropoulos. 2010. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings.* USENIX Association, 223–240. http://www.usenix.org/events/sec10/tech/full_papers/Burkhart.pdf

[11] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. 2021. Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI. Cryptology ePrint Archive, Report 2021/172. https://ia.cr/2021/172.

[12] David Chaum. 1988. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *J. Cryptol.* 1, 1 (1988), 65–75. https://doi.org/10.1007/BF00206326

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

[14] Yann Collet. 2021. xxHash. https://cyan4973.github.io/xxHash/

[15] Sumit Kumar Debnath, Tanmay Choudhury, Nibedita Kundu, and Kunal Dey. 2021. Post-quantum secure multi-party private set-intersection in star network topology. *J. Inf. Secur. Appl.* 58 (2021), 102731. https://doi.org/10.1016/j.jisa.2020.102731

[16] Sumit Kumar Debnath, Pantelimon Stanica, Nibedita Kundu, and Tanmay Choudhury. 2021. Secure and efficient multiparty private set intersection cardinality. *Adv. Math. Commun.* 15, 2 (2021), 365–386. https://doi.org/10.3934/amc.2020071

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

[18] Taher ElGamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings (Lecture Notes in Computer Science, Vol. 196)*, G. R. Blakley and David Chaum (Eds.). Springer, 10–18. https://doi.org/10.1007/3-540-39568-7_2

[19] Keith B. Frikken. 2007. Privacy-Preserving Set Union. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4521)*, Jonathan Katz and Moti Yung (Eds.). Springer, 237–252. https://doi.org/10.1007/978-3-540-72738-5_16

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

[21] Mike Hamburg. 2015. Decaf: Eliminating Cofactors Through Point Compression. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9215)*, Rosario Gennaro and Matthew Robshaw (Eds.). Springer, 705–723. https://doi.org/10.1007/978-3-662-47989-6_34

[22] Feng Hao and Piotr Zielinski. 2006. A 2-Round Anonymous Veto Protocol. In *Security Protocols, 14th International Workshop, Cambridge, UK, March 27-29, 2006, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 5087)*, Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe (Eds.). Springer, 202–211. https://doi.org/10.1007/978-3-642-04904-0_28

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

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

[25] Florian Kerschbaum. 2012. Outsourced private set intersection using homomorphic encryption. In *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*, Heung Youl Youm and Yoojae Won (Eds.). ACM, 85–86. https://doi.org/10.1145/2414456.2414506

[26] Aggelos Kiayias and Moti Yung. 2003. Non-interactive Zero-Sharing with Applications to Private Distributed Decision Making. In *Financial Cryptography, 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003, Revised Papers (Lecture Notes in Computer Science, Vol. 2742)*, Rebecca N. Wright (Ed.). Springer, 303–320. https://doi.org/10.1007/978-3-540-45126-6_22

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

[28] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical Multi-party Private Set Intersection from Symmetric-Key Techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1257–1272. https://doi.org/10.1145/3133956.3134065

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

[30] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, Yehuda Lindell (Ed.). Springer International Publishing, 277–346. https://doi.org/10.1007/978-3-319-57048-8_6

[31] Atsuko Miyaji and Shohei Nishida. 2015. A Scalable Multiparty Private Set Intersection. In *Network and System Security - 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9408)*, Meikang Qiu, Shouhuai Xu, Moti Yung, and Haibo Zhang (Eds.). Springer, 376–385. https://doi.org/10.1007/978-3-319-25645-0_26

[32] Ofri Nevo, Ni Trieu, and Avishay Yanai. 2021. Simple, Fast Malicious Multiparty Private Set Intersection. Cryptology ePrint Archive, Report 2021/1221. https://ia.cr/2021/1221.

[33] Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl. 2010. Cardinality estimation and dynamic length adaptation for Bloom filters. *Distributed Parallel Databases* 28, 2-3 (2010), 119–156. https://doi.org/10.1007/s10619-010-7067-2

[34] Ou Ruan, Zihao Wang, Jing Mi, and Mingwu Zhang. 2019. New Approach to Set Representation and Practical Private Set-Intersection Protocols. *IEEE Access* 7 (2019), 64897–64906. https://doi.org/10.1109/ACCESS.2019.2917057

[35] Yingpeng Sang and Hong Shen. 2009. Efficient and secure protocols for privacy-preserving set operations. *ACM Trans. Inf. Syst. Secur.* 13, 1 (2009), 9:1–9:35.

https://doi.org/10.1145/1609956.1609965

[36] Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. 2012. Constant-Round Multi-party Private Set Union Using Reversed Laurent Series. In *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7293)*, Marc Fischlin, Johannes Buchmann, and Mark Manulis (Eds.). Springer, 398–412. https://doi.org/10.1007/978-3-642-30057-8_24

[37] Katsunari Shishido and Atsuko Miyaji. 2018. Efficient and Quasi-accurate Multi-party Private Set Union. In *2018 IEEE International Conference on Smart Computing, SMARTCOMP 2018, Taormina, Sicily, Italy, June 18-20, 2018*. IEEE Computer Society, 309–314. https://doi.org/10.1109/SMARTCOMP.2018.00021

[38] Anselme Tueno, Florian Kerschbaum, Stefan Katzenbeisser, Yordan Boev, and Mubashir Qureshi. 2020. Secure Computation of the k$^{\text{th}}$-Ranked Element in a Star Network. In *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, 386–403. https://doi.org/10.1007/978-3-030-51280-4_21

[39] Henry de Valence, Isis Lovecruft, and Tony Arcieri. 2021. The Ristretto Group. https://ristretto.group/why_ristretto.html

[40] Henry de Valence, Isis Lovecruft, and Tony Arcieri. 2021. Testing Equality. https://ristretto.group/formulas/equality.html

[41] Jelle Vos, Zekeriya Erkin, and Christian Doerr. 2021. Compare Before You Buy: Privacy-Preserving Selection of Threat Intelligence Providers. Cryptology ePrint Archive, Report 2021/1260. https://ia.cr/2021/1260.

A COMPLEXITIES OF MPSI PROTOCOLS

In this section we provide our reasoning for the altered complexities in Table 1. We use the notation from Table 2. In the final complexities we substitute m with $O(k)$, as explained in Section 6.

A.1 Kissner & Song [27]

The authors already provide a computational and communication complexity but we are interested in the complexity per party rather than the total complexity.

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

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

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

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

A.2 Hazay et al. [23]

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

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

A.3 Inbar et al. [24]

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

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

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

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

A.4 Bay et al. [4]

(1) Each assistant sends an encrypted bitset, which takes $O(|\mathcal{U}|)$ bits.
(2) The leader sends all assistants at most k aggregated bits, which takes $O(k)$ bits.
(3) Each assistant partially decrypts at most k aggregated bits, which takes $O(k)$ bits.

This results in a communication complexity of $O(nk)$ for the leader and $O(|\mathcal{U}|)$ for an assistant.

(1) Each assistant generates an encrypted bitset, which takes $O(|\mathcal{U}|)$.
(2) The leader aggregates the results using homomorphic addition for the bins corresponding to its set elements, which takes $O(nkh)$.
(3) Each assistant partially decrypts at most k aggregated bins, which takes $O(k)$.

As a result, the leader performs $O(nkh)$ operations, while the assistants perform $O(|\mathcal{U}|)$ operations.

A.5 Debnath et al. [16] & Bay et al. [5]

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

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

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

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

B COMPLEXITIES OF MPSU PROTOCOLS

B.1 Frikken [19]

Following the author's complexity analysis, the parties each share $O(nk)$ tuples with two ciphertexts in step 2b of the protocol, so the communication complexity for each individual party is $O(nk)$ bits. Each party must be online at least once in steps 1a, 1b, 1c, 2b, 3 and 4, so the protocol requires at least 6 stages.

(1) Each party encodes their set as encrypted polynomial coefficients, which takes $O(k)$.
(2) Each party $\mathcal{P}_i$ homomorphically multiplies i encrypted polynomials together, which takes at most $O(nk^2)$.
(3) Each party $\mathcal{P}_i$ encrypts $2k$ values and homomorphically multiplies $i-1$ encrypted polynomials together, which takes at most $O(nk^2)$.
(4) Each party $\mathcal{P}_i$ homomorphically multiplies i ciphertexts together, which takes at most $O(nk)$.
(5) Each party takes part in a secure shuffle protocol with at most nk tuples, which we assume to be linear with the number of parties and tuples.
(6) All parties work together to decrypt at most nk tuples, which scales linearly with n and k.

So, the overall computational complexity is $O(nk^2)$ for each party.

B.2 Seo et al. [36]

In this work, p is the order of the finite field used in secret sharing. It needs to hold that $p \leq |\mathcal{U}|$, so the computational complexity becomes $O(n^4k^2 + n^2k^2 \log |\mathcal{U}|)$. For brevity, we omit the logarithmic term: $\tilde{O}(n^4k^2)$.

C EXTENDED SECURITY PROOF

Theorem C.1. *For a set of colluding parties $C = \{1\} \cup C'$ $C' \subset \{2,\ldots,n\}$ there exists a simulator $\mathcal{S}_2$ so that:*

$$\mathcal{S}_2(z) \overset{c}{\equiv} \bigcup_{c\in C} \text{view}_c(x_c) \ . \tag{27}$$

Proof. We construct simulator $\mathcal{S}_2$, which takes the protocol's output z. The simulator takes no inputs because we can generate an indistinguishable view without explicitly incorporating them. The output of the simulator is a complete set of simulated incoming messages from the remaining honest assistants: $\{\alpha_i^j, \beta_i^j, \overline{\alpha}_i^t, \overline{\beta}_i^t, \sigma_i^t\}$ for all honest $\mathcal{P}_i$, in other words $i \in \overline{C}$.

First, the simulator randomly samples $\langle \alpha_i^j, \beta_i^j \rangle \leftarrow \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$ and $\langle \overline{\alpha}_i^t, \overline{\beta}_i^t \rangle \leftarrow \langle \mathcal{R}(\mathbb{G}), \mathcal{R}(\mathbb{G}) \rangle$, which are computationally indistinguishable from actual views as argued in Theorem 4.3.

The simulator then generates σ_i^t, which depends on the expected output z. If $z = 1$, the simulator randomly samples $\sigma_i^t \leftarrow \mathcal{R}(\mathbb{G})$. However, if $z = 0$, the simulated output would be incorrect with overwhelming probability, as shown at the end of Theorem 4.1. Instead, the simulator will choose one honest assistant $H \in \overline{C}$ for which it generates another σ_H^t. For the other assistants $c \in \overline{C} \setminus \{H\}$, the simulator samples $\sigma_c^t \leftarrow \mathcal{R}(\mathbb{G})$. The simulator computes:

$$\sigma_H \leftarrow \overline{\beta}^t - \left(\sum_{c\in C} \sigma_c + \sum_{c\in \overline{C}\setminus\{H\}} \sigma_c \right) . \tag{28}$$

It is clear to see that $\sum_{i=1}^n \sigma_i^t = \overline{\beta}^t$, so the output is 0.

Finally, we show that σ_i^t is indeed indistinguishable from the actual views. If it was sampled randomly, then it holds that $\sigma_i^t \overset{s}{\equiv} sk_i\, \overline{\alpha}^t$, because sk_i is an unknown value sampled randomly from $\mathbb{Z}_q$. Moreover, since we only choose assistant $H \in \overline{C}$ when $z = 0$, its σ_H^t is also statistically indistinguishable from an actual view since the other values in the summation are statistically indistinguishable and the output is known to be 0. □

D DETAILS OF THE DIVIDE-AND-CONQUER APPROACH

We provide pseudocode for the underlying algorithms of the divide-and-conquer approach for multi-party private set unions on bitsets. Algorithm 2 provides the functionality to split a range up into D similarly sized partitions.

Algorithm 2 Selects the indices of D partitions

```
1: procedure SPLIT(min, max, $D$)
2:     $s \leftarrow \frac{\max - \min}{D}$
3:     indices ← []
4:     $i \leftarrow \min$
5:     while $i < \max$ do
6:         $j \leftarrow i + s$
7:         Append $(\lfloor i \rceil, \lfloor j \rceil)$ to indices
8:         $i \leftarrow j$
9:     return indices
```

Algorithm 3 in turn uses the splitting algorithm to run the actual recursive MPSU protocol.

Next, we provide a derivation for finding that the optimal partition number $D = e$. For $N > 1$ we have the following minimization problem:

$$\min_{D>1} \quad D \log_D N \ . \tag{29}$$

Algorithm 3 Divide-and-conquer approach for one party

```
procedure DivideAndConquer($\hat{X}, D$)
    result $\leftarrow [0, \dots, 0]$                ▷ Bit vector of length $|\mathcal{U}|$
    $I_{\text{prev}} \leftarrow [(0, |\mathcal{U}|)]$
    while $|I_{\text{prev}}| > 0$ do
        $I_{\text{curr}} \leftarrow [\text{Split}(i, j, D) \quad \forall (i, j) \in I_{\text{prev}}]$
        $I_{\text{prev}} \leftarrow [\ ]$
        ▷ This for loop can be performed in parallel
        for $(i, j) \in I_{\text{curr}}$ do
            $x \leftarrow \hat{X}[i] \vee \cdots \vee \hat{X}[j]$
            Take part in a private OR protocol with $x$
            Receive result $z$ from the leader $\mathcal{P}_1$
            if $j - i = 1$ then
                result$[i] = 1$
            else if $z = 1$ then
                Append $(i, j)$ to $I_{\text{prev}}$
    return result
```

We find the optimum by differentiating for D and determining when the derivative is 0:

$$0 = \frac{d}{dD} D \log_D N \,, \tag{30}$$

$$= \frac{d}{dD} D \frac{\ln N}{\ln D} \,, \tag{31}$$

$$= \frac{\ln N}{\ln D} - \frac{\ln N}{\ln^2 x} \,. \tag{32}$$

Since $N > 1$, the only solution is $x = e$.