Multi-party Private Set Operations with an External Decider

Sara Ramezanian, Tommi Meskanen, Valtteri Niemi

Multi-party Private Set Operations with an External Decider

Sara Ramezanian, Tommi Meskanen, and Valtteri Niemi

University of Helsinki, Helsinki, Finland
Helsinki Institute for Information Technology (HIIT)
{first name.last name}@helsinki.fi

Abstract. A Private Set Operation (PSO) protocol involves at least two parties with their private input sets. The goal of the protocol is to learn the output of a set operation, e.g., set intersection, on the parties' input sets, without revealing any information about the items that are not in the output set. Commonly, the outcome of the set operation is revealed to parties and no one else. However, in many application areas of PSO, the result of the set operation should be learned by an external participant who does not have an input set. We call this participant *the decider*. In this paper, we present new variants of multi-party PSO, for the external decider setting. All parties except the decider have a private set. Parties other than the decider neither learn this result, nor anything else from this protocol. Moreover, we studied generic solutions to the problem of PSO in the presence of an external decider.

Keywords: Private Set Operation · Applied Cryptography · Secure Multi-party Computation · Homomorphic Encryption · Privacy Enhancing Technologies · Data Security

1 Introduction

Private Set Operations (PSO) [1] such as Private Set Intersection (PSI) [2], Private Set Union (PSU) [3], and Private Membership Test (PMT) [4], are special cases of Secure Multi-party Computation that have been in the interest of many researchers. In particular, the problem of PSI has been studied a lot and many PSI protocols have been proposed [5].

Electronic voting [6], botnet detection between different ISPs [7], and genomic applications [8] are a few real-life examples of set operations where the sets are about sensitive data. Also, there is a growing need for privacy-preserving data analysis, e.g., in the context of 5G networks. Therefore, a comprehensive study of the factors that affect the feasibility and efficiency of a PSO protocol is needed, and a general but still feasible solution to any multi-party PSO problem is worth seeking for.

Multi-party PSO is a protocol with several parties, each holding a private set, who want to perform a function on their input sets. Usually, at the end of the protocol all parties learn the outcome of this function, and nothing else.

However, in many realistic PSO scenarios, instead of parties themselves learning the outcome, there is a special party who does not have an input set but who needs to learn the result of the PSO. We call this special party *an external decider*, or for short, *a decider*. In our setting, the decider is a trusted third party who typically does not have an incentive to get any information about the input sets, except the output of the set operation. In Section 2 we present several examples of real life scenarios where the result of PSO is obtained only by the decider, and no one else.

The contributions of this work are as follows:

- We classify the problem of PSO with different criteria.
- We present a comprehensive study of PSO problem with an external decider. To the best of our knowledge, this variant of PSO has been studied only in special cases, such as secure electronic voting, but not in the general case.
- We specifically study the case where the set elements are chosen from a universe of limited size. We present a general solution to any PSO problem with external decider and with limited universe.
- We present another general solution to any PSO problem with external decider, where the universe is not limited. This protocol solves the emptiness and cardinality of the output set.
- We assume that all parties are semi-honest, but we also present a modification that provides protection in the presence of one malicious party.
- Finally, we implement our protocols and compare the efficiency of our protocols against the existing work for PSO problems.

Next we briefly explain the necessary concepts that are required to understand the rest of the paper.

Homomorphic encryption (HE) schemes allow computations to be carried out using encrypted values, without the need to decrypt them first. In our paper, we are interested in *additively* homomorphic encryption such as Paillier cryptosystem [9]. If two ciphertexts, $c_1 = enc(m_1)$ and $c_2 = enc(m_2)$, are generated using an additively homomorphic encryption scheme, then the product (or result of some other operation) of these two ciphertexts is decrypted to the sum of the two plaintexts: $dec(c_1 \cdot c_2) = m_1 + m_2$.

Keyed hash function, such as HMAC [10], is a cryptographic hash function with a secret key that is utilized to create fingerprints of messages. The key is only known to the designated parties. A keyed hash function is a collision-resistant one-way function. [11].

2 Motivational Examples of PSO with the Decider

In this section, we present several examples of scenarios where most parties provide only input data while the result of the set operation goes to an external decider. In each scenario preserving the privacy is important.

- **Example-1:** Several proposals are on the table for a board meeting of an organization. Every board member would choose which proposals are

acceptable for them. However, they are not willing to reveal their choices to others. Therefore, a PSO protocol is used and the secretary (the decider) computes what proposals are acceptable for everyone.

- **Example-2:** A traffic office has installed surveillance cameras in a city and each of them is collecting plate numbers of the passing cars. The office wants to collect all kinds of statistics from the traffic. For example, how many different cars were observed during a day, or how many of those cars that were seen at either point A or point B in the morning were also seen at one of these points in the afternoon. Of course, technically the cameras could simply send all observed data to the central office but this would be a privacy violation. Instead, the central office is an external decider and all cameras deliver input for various PSOs.
- **Example-3:** A decentralized social networking platform such as HELIOS[1] is inherently more privacy-friendly than centralized solutions. However, people would still like to make searches in larger setting than within their own direct contacts. For example, number of people have formed a group G (for some purpose). In order to extend the group, one of the members of the group would like to identify, in privacy-preserving manner, whether her/his friend is also a friend of at least three other existing group members and could be asked to join. By utilizing PSI between that group member and other group members, the platform (the decider) can learn which user is potentially interested to join the group G.
- **Example-4:** Privacy preserving digital parental control in 5G networks [12] is another use case where PSO could be needed. A parent has a set of unwanted attributes that websites can have and wants to prevent the child to access websites that contain any of these attributes. A child wants to access a website. Each child under digital parental control has an application installed on their device that is called the *kid-client*. The network analyses the website that the child wants to access and stores the attributes of that website in a private set. The kid-client should decide whether to grant or deny access to this child. This is a PSI between the network's and parent's respective private sets. It is enough for the kid-client (the decider) to learn whether the end result is empty or not.
- **Example-5:** The government of a country wants to find out whether the health-care system functions properly. For instance, the government wants to learn the percentage of the people at higher risk from certain disease who are covered by insurance. In this case, the government wants to know the number of people in a set A, which is the union of people ensured by different companies, that are also in a set B, which is the union of all people identified with higher risk by different hospitals. This is an example of cardinality for PSI+PSU, which will be learned by the government (the decider).
- **Example-6:** Conversion rate of an advertisement is measured by finding the size of the intersection between people who see the ad and who completed a relevant transaction. The advertisement is shown in different platforms and

[1] HELIOS project homepage. (2021). Retrieved from https://helios-h2020.eu/.

the transaction is completed with the owner of the product. The names of the individuals who visited different platforms, and also the names of the buyers are considered to be private. Therefore, the marketing company (the decider) has to use a PSO protocol to learn the conversion rate of its ad. In this example, the decider learns the cardinality of the intersection.

3 Classification of PSO Problems

The scenarios of Section 2 lead to different types of PSO problems. In general, PSO problems can be classified using different criteria. In the following, we discuss some of these criteria. Later we study some of the many possible PSO problems further and develop solutions. Note that the list of criteria is certainly not complete and there could be more factors that guide the future research directions.

- **Criterion 1: "What information is wanted from the set".** At least the following three questions could be asked about the end result set: "What is the cardinality of the end result set?" "What are the elements of this set?" "Is it empty?"
- **Criterion 2: "Who gets the outcome".** PSO problems vary also in the way the final result is learned. For instance, in some scenarios it is required that all parties learn the outcome, whereas in some cases only one party learns the result. In this paper, we focus on the case where the result of the protocol goes to an external decider.
- **Criterion 3: "Adversary model".** Semi-honest and malicious adversarial models are two common settings in privacy-preserving protocols.
- **Criterion 4: "Size of the universe".** In each PSO problem, elements of the input sets belong to an a priori defined set of potential elements. Hereafter we call this set *the universe*. Whether the total number of elements in the universe is limited or not is a criterion that could be taken into account when defining a PSO problem.
- **Criterion 5: "Number of parties that have input".** Number of participants can vary a lot, depending on the use case for which the PSO problem is solved.
- **Criterion 6: "What is the set operation".** We need to determine what is the set operation to be computed, defined by combining intersections, unions and complements of sets.
- **Criterion 7: "Size of input sets".** Whether the sets are the same (or almost the same) in size, whether the sets are large or small, or whether some sets are actually singletons.
- **Criterion 8: "Is it possible to use a trusted party".** Many security and privacy solutions become simpler if it is possible to get help from an outsider who can be trusted by all stakeholders.
- **Criterion 9: "On-line or off-line sets".** There are use cases where at least some part of the input data is known well before the output data is needed. Then it may be possible to do some off-line computations before the PSO protocol is run in on-line fashion.

4 Problem Statement

We study many variants of multi-party PSO problems. For the criterion "Who gets the outcome" we restrict ourselves to the case where the decider gets the result. For the Criterion 8, we only cover the case without the trusted third party. For the Criterion 3, we first assume that the parties are semi-honest. Later we show how to modify the protocols to fit in the malicious adversarial model.

We assume there are $n \geq 2$ parties plus a decider in the protocol. Each party P_i has a private set S_i, where $1 \leq i \leq n$. The decider does not have an input set and all the other parties trust the decider to calculate the result according to the protocol. It depends on the use case who the decider is. In the use cases, the decider does not typically have any incentive to learn more information than the final outcome. Moreover, the decider may want to prove that they cannot learn anything else than what is necessary. This can be for the reason of safety, data minimization, to avoid the necessity to destroy the unwanted data, to prevent conflict of interests, to avoid the need of legal documents, or to convince the public opinion that their privacy is preserved by the decider. For instance, in examples 2 and 5 the deciders want to show the citizens that their private lives are not monitored. In example 1, the decider wants to avoid the possibility of blackmailing, signing Non-disclosure Agreement (NDA) and the process of deleting the data. Deciders of examples 3, 4 and 6 want to show their users that their privacy is preserved, and avoid potential blackmailing and conflicts of interests inside their organizations.

Each example of Section 2 can be described as one of the following problems:

- **Problem-1:** All parties have their private sets as input. After executing the protocol, the decider wants to learn the answer to one of the following questions: 1) What are the elements in the union of all n sets. 2) What is the cardinality of the union. 3) Whether the union is an empty set.
- **Problem-2:** All parties have a set. After executing the protocol, the decider learns the answer to one of the following questions: 1) What are the elements in the intersection of n sets of all other parties. 2) What is the cardinality of the intersection. 3) Whether the intersection is empty.
- **Problem-3:** The parties all have their private sets. After executing the protocol, the decider learns the result of any given set operation. The general PSO can be written in Conjunctive Normal Form (CNF):

$$S_T = (A_{1,1} \cup ... \cup A_{1,\alpha_1}) \cap ... \cap (A_{\beta,1} \cup ... \cup A_{\beta,\alpha_\beta}) \tag{1}$$

  where $A_{i,j} \in \{S_1, ..., S_n, \bar{S}_1, ..., \bar{S}_n\}$, the set $\bar{S}_i$ is the complement of the set S_i, and $1 \leq \alpha \leq n$ and $\beta \in \mathbb{N}$. After the protocol has been executed the decider learns answer to one of the following questions: 1) What are the elements of S_T. 2) What is the cardinality of S_T. 3) Is S_T empty.

Note that Problem-3 covers both Problem-1 and Problem-2 but we consider those separately for two reasons: 1) so many use cases are only about Problem-1 or Problem-2, and 2) our solutions for Problem-1 and Problem-2 are used when building one of the solutions for Problem-3.

5 Related Work

In this section, we first present the state of the art in PSO protocols, then we present some of the related previous works on secure multi-party computation.

As we mentioned before, private set operations are applicable to variety of use cases, such as privacy-preserving genomic similarity [13] and private profile matching in social media [14]. Therefore, different variants of PSO problems have been studied extensively, see [5]. Kolesnikov et al. in [15] proposed a new function that is called Oblivious Programmable Pseudorandom Function, and used it to design a practical multi-party PSI protocol that is secure in a malicious setting. In 2019, Ghosh and Nilges proposed a novel approach to PSI [16], by utilizing Oblivious Linear Function Evaluation (OLE) to evaluate the intersection.

At the time of writing, the protocol of Kolesnikov et al., in [2] is the fastest two-party PSI protocol. In [2], the authors proposed a variant of Oblivious Pseudorandom Function, and utilized it to achieve a light-weight PSI protocol.

Chun et al., generalized the problem of PSO by studying any PSO problem in disjunctive normal form (DNF) [17]. Wang et al., further studied the general PSO problem in DNF for a limited universe [18]. In this paper, we study the general problem of PSO in the setting with an external decider. To the best of our knowledge, this is the first time that this problem is studied comprehensively.

Feige et al. proposed a general solution to compute any function in the secure multi-party computation setting where there is a party without input set who computes the final result and gives it to the other parties [19]. As an example case, they presented an efficient algorithm to compute the logical AND of n bits, each bit belonging to different party. They assumed that all parties follow the protocol honestly. For the limited universe case, computing the logical AND is essentially equivalent to computing set intersection. Although their protocol for computing the logical AND is efficient, they did not show how this can evolve such that it can solve any set operations in DNF form.

There are several variants of secure multi-party set operations that use a "special party" in their setting. We briefly explain these variants and why these are different from our setting.

The work of Feige et al. in [19], lead to introduction of a variant of secure multi-party computation called *Private Simultaneous Messages* (PSM) [20]. A PSM protocol is between n parties each with a private input, and a "referee" who does not have input data. Parties have access to a common secret. Each party computes a single message by utilizing the common secret and the party's private input, and sends this message to the referee. After receiving all the messages from all the n parties, the referee is able to compute the output of the function. The referee should not learn anything else than this outcome. The construction of PSM protocols focus on information theoretical security, rather than on computational efficiency [21]. In [22] the authors present protocols to compute any function with PSM schemes using a key that can be used only once, thus each protocol needs to be initialized with a different secret key every time it is run. Our solutions differ from these unconditionally secure solutions in that we allow usage of the same keys for several different PSO instances, with different

input sets and even with different operations. Moreover, the communication complexity of the protocols in [22] is increasing exponentially when the number of parties growing, whereas in our PSO protocols the communication complexity is independent of the number of participants.

Functional encryption (FE) is a variant of public key encryption that allows the holder of the secret key to learn a function of the plaintext [23]. The notion of FE can also be applied in the case of many parties with inputs [24]. In principle, if the decider is the party who holds the secret key, the setting of multi-input FE can fit into the PSO scenario with the decider. While there are accelerators for special cases of FE, no efficient solutions are known that would cover all cases [25].

In the server aided PSO setting the privacy preserving set operation protocol uses an untrusted server that carries out some part of the computations (similar to secure cloud computing). In other words, the server only helps the parties by performing some part of the computations and does not get the final result [26], [27].

The setting of secret-sharing based secure multi-party computation protocols [28] is typically designed in such a way that all parties execute their share of the computations and send their results to a party who will compute the result (the resulting party/the dealer), and later announces the final result. This setting is more complex than ours because resulting party also has a private set that they need to hide from other parties. Moreover, the resulting party has to prove to other parties that they performed the computations correctly, without revealing any extra information on the private sets to the parties. Therefore, the typical secret sharing based protocols perform much less efficiently than our solutions with a decider.

6 Protocols

In this section, we present our privacy preserving set operation protocols. These protocols are our proposed solutions to the problems of Section 4.

In each protocol there are $n+1$ participants involved with the set operations: n parties have input sets and the result of the protocol goes to the decider D, who does not have an input set. Other n parties do not get the final outcome of the protocol. We present our protocols with the assumption that the participants are semi-honest. Later in Section 9 we present a solution for malicious model as well.

6.1 Protocol-0

In this section and sections 6.2, 6.3 and 6.4 we assume that the universe is limited, i.e., it is possible to present it as an ordered set $U = \{a_1, ..., a_u\}$. For simplicity, we assume that the decider creates this ordered set. Parties create a shared repository which contains a vector with u components. Each component represents an item in U, and the order of the components is as defined in U.

Protocol 0: The off-line phase.

1. The decider sends the ordering of $U = \{a_1, ..., a_u\}$ to all the other parties.
2. The decider creates public and private keys that fulfil requirements for an additively homomorphic encryption scheme.
3. The decider picks one of the parties randomly and informs this party that they are responsible for sending the final result to the decider. For simplicity, let us assume that the decider chooses party P_n.
4. The decider sends the public key of the encryption scheme to all parties.
5. Each party P_i creates two sets of encrypted values by utilizing the decider's public key. One set contains u instances of enc(0), and the other set contains u instances of enc(r) where r is a random number chosen specifically for that instance.
6. Parties create a shared repository.
7. Parties together create a vector $V = (V_1, ..., V_u)$ with u components, where each component is an instance of enc(r), where r is a random number.

Each component is a bit string of a certain length. All parties can read and write to the repository but two parties cannot write at the same time, to avoid conflicts. The decider or anybody else than parties P_i does not have access to this repository. The details about the required technologies to create this kind of repository are outside the scope of this paper. The shared repository is the centralized part of our PSO protocol. Protocol 0 is the off-line phase used in all protocols of this section.

6.2 Protocol-1

Let us assume that party P_i has a private input set S_i, for $i = 1, ..., n$. The decider wants to learn one of the following cases about the union of these n sets: 1) What are the elements in $\bigcup_{i=1}^{n} S_i$. 2) What is the cardinality of $\bigcup_{i=1}^{n} S_i$. Our solution for these questions is by Protocol 1.

If the application area of the protocol is such that the decider only needs to know whether the union of all input sets is empty, the protocol can be simplified a lot. Instead of vector V we have just one value V that is initially set to enc(1). Then, each P_i multiplies V by enc(0) if S_i is empty and replaces V by enc(0) if S_i is not empty. When all the parties have altered V, party P_n sends it to D. The decider decrypts and if D gets zero it means that the union is non-empty.

After executing the on-line phase, only the decider learns the outcome, and other parties do not learn anything else about this protocol than that it was run.

6.3 Protocol-2

Let us again assume that the party P_i has the set S_i. The decider wants to learn one of the following cases about the intersection of these n sets: 1) What are the

Protocol 1:

0 Protocol-0 is run.
1 Each party P_i, where $1 \leq i \leq n$, modifies vector V as follows. If $u_j \in S_i$ then P_i replaces V_j by enc(0), which is one of the encryptions of zero that P_i generated in Protocol-0. If $u_j \notin S_i$ then P_i multiplies V_j by enc(0).
2 When all parties have finished their modifications on vector V then one of the following cases will take place.
3 **Case 1:** Party P_n sends V to the decider. The decider decrypts components of this vector, and $a_j \in \bigcup_{i=1}^{n} S_i$ if and only if $dec(V_j) = 0$.
Case 2: Party P_n permutes the components of V before sending them to D. The decider decrypts V. The number of zeros in the decrypted vector is equal to the cardinality of $\bigcup_{i=1}^{n} S_i$.

elements in $\bigcap_{i=1}^{n} S_i$. 2) What is the cardinality of $\bigcap_{i=1}^{n} S_i$. 3) Whether $\bigcap_{i=1}^{n} S_i$ is an empty set. Our solution to these questions is Protocol 2.

After executing this protocol, only the decider learns the result. Other parties do not learn anything about each others' sets or about the result of the protocol.

6.4 Protocol-3: A generic solution to perform any PSO with an external decider

It can be shown that every set that is obtained from a collection of sets by operations of intersection, union and complement can equivalently be computed by a conjunctive normal form. Thus, the general PSO problem can be written as presented in Equ. 1.

The general PSO problem can be formalized as follows: Party P_i has a private input set S_i, for $i = 1, ..., n$. The decider D wants to learn one or more of the following cases about the set S_T of Equation 1: 1) What elements are there in the set S_T. 2) What is the cardinality of S_T. 3) Whether S_T is an empty set. Other parties should not learn anything about this protocol except that it is executed. Our solution for these questions is by Protocol 3.

6.5 Protocol-4: Keyed hash functions with dummies

In this section we drop the assumption that the universe is limited, and present protocols for finding answers to the following questions about the set S_T that can also be written in Disjunctive Normal Form as $S_T = (A_{1,1} \cap ... \cap A_{1,\alpha_1}) \cup ... \cup (A_{\beta,1} \cap ... \cap A_{\beta,\alpha_\beta})$: 1) What is the cardinality of S_T? 2) Is this set empty? In other words, our protocols cover all private set operations but the decider cannot get the elements in the result set. In these protocols we use keyed hash function, and assume that the parties are semi-honest.

In the off-line phase of this protocol, the parties P_1 to P_n decide on a shared key k to be used for computing a keyed hash function. The decider should not learn this key. We may even assume that the key k is generated by a trusted hardware and anyway we assume that only the authorized parties (e.g. cameras

Protocol 2:

0 Protocol-0 is run, with one difference: in step 7, all components of vector V are initially set to instances of enc(0).

1 Each party P_i where $1 \leq i \leq n$ modifies every component j of the vector V as follows.

$$V_j = \begin{cases} V_j \cdot enc(0) & \text{if } a_j \in S_i \\ V_j \cdot enc(r) & \text{otherwise.} \end{cases} \quad (2)$$

After all parties have modified the vector V, one of the following cases will be executed.

2 **Case 1:** Party P_n sends V to the decider. After decrypting, for entries that do not decrypt to zero, D learns they are not in the intersection. Similarly, if decryption yields zero, D learns that the corresponding element is in every input set. Therefore, the decider learns $\bigcap_{i=1}^{n} S_i$.
Case 2: Party P_n shuffles V and sends it to D. The decider decrypts and the number of zero values in the decrypted vector is the cardinality of $\bigcap_{i=1}^{n} S_i$.
Case 3: To hide the true cardinality of the intersection, the parties need to create "clones" of elements in the universe. For each element that is in the intersection, there would be many zeros after decryption. The parties would also add many encryptions of non-zero numbers to hide further the number of elements in the intersection.

in Example-2) have access to this key. Because the parties $P_1, ..., P_n$ are semi-honest, they do not collude with the decider and reveal the key to D. In other words, the decider cannot access the key k.

The basic idea is simple. All parties replace elements in their input sets with images of the elements under keyed hash function. However, the parties cannot simply send the images to the decider because then the decider would get lots of information about the input sets.

Let us first consider the question 1. The true cardinalities of all input sets are hidden from the decider and from other parties by adding a big number of "dummies" among the true images of elements in input sets. These dummies are just random bit strings that look like results of the keyed hash function.

We present the protocol in the case of a simple example, for better illustration. It is straight-forward to convert the presentation to the general case but we skip it, for sake of compactness. Let us assume that there are three parties A, B and C in the protocol and the decider wants to know the cardinality of set $(A \cap B \cap \bar{C}) \cup (B \cap C)$. The Venn diagram for the three input sets is shown in Figure 1. Eight disjoint sets are formed by first choosing either the input set or its complement for each party and taking an intersection of the chosen three sets. Because the total universe could be very large, one of these eight sets (the intersection of complements) is assumed to be non-relevant for the end result of the PSO. For each of the other 7 disjoint sets, the parties A, B and C agree on the number of dummy values. Please note that we can only increase the total number of elements in each set, and if the number of dummies is small compared to the total number, the dummies do not change the whole picture. If the number

Protocol 3:

0 The off-line phase of this generic protocol is as explained in Protocol-0 with only one difference: in the step 7, the parties create β vectors W^k, where $1 \leq k \leq \beta$. Each vector W^k is created similarly to the vector V in step 7 of Protocol-0. On-line phase of Protocol-1 is used as a building block of the on-line phase of the generic solution.

1 In order to compute each W^k, parties should compute each set $(A_{k,1} \cup ... \cup A_{k,\alpha_k})$ by utilizing Protocol-1. Note that party P_i either inputs S_i or $\bar{S}_i$ or does not attend the computation for this term.

2 After all the vectors W^k have been computed, party P_n creates a new vector Z where every entry j of the vector is computed as $Z_j = \prod_{k=1}^{\beta} W_j^k$.

3 Now, one of the following cases will be executed.

4 **Case 1:** Party P_n sends vector Z to D. The decider decrypts Z. For every entry Z_j which decrypts to zero, the decider learns that the corresponding element a_j is in the set S_T. On the other hand, if decryption yields a non-zero, then the corresponding element is not on the set S_T.
Case 2: Party P_n shuffles vector Z and sends it to D. The decider then decrypts the vector Z. The cardinality of S_T is equal to the number of zero values in the decrypted vector.
Case 3: Similarly to Case 3 of Protocol-2, the party P_n creates a new vector Z' from the vector Z, by appending the components of Z and their duplicates to vector Z'. Then, P_n shuffles vector Z', and sends it to the decider. The decider decrypts the vector. If at least one of the values in Z' decrypts to zero, then S_T is non-empty. Otherwise, the set of Equ. 1 is empty.

of dummies is about the same as the total number, then the total number can still be found out. Thus, the number of dummies should dominate the actual size. In other words, the number of dummies should be at least one order of magnitude greater than the typical size of an input set. The agreed numbers of dummy values are shown in Fig. 1.

The parties also need to agree on the actual 97 dummy values that every party adds to the set of values they would later send to the decider. Similarly, A and C have to agree values for the 12 dummies that both of them include among keyed hash images of their input sets. Parties A and B agree on 23 joint dummy values, while B and C agree on 53 joint dummies. Finally, A would freely choose 34 random dummy values while B (resp., C) chooses 88 (resp., 145) dummies. The decider receives all the hash-values and dummy values. From the received values, the decider identifies those that appear in every set, those that have been received from A and B but not from C, and those that have been received from B and C but not from A. Finally the decider is asked to subtract 173 (= 23 + 97 + 53) from the gross number of collisions explained above.

Note that every PSO problem can be presented in CNF or alternatively in DNF. Our example PSO above was in disjunctive normal form. This is mainly just for making our solution easier to explain.

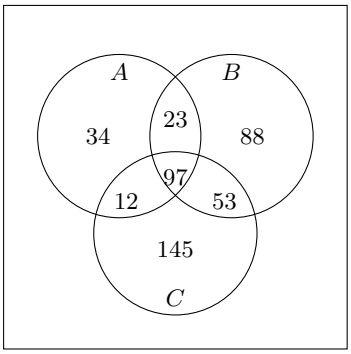

Fig. 1: An example of a PSO for three sets with keyed hash function.

Now, let us now assume that the decider only needs to learn whether the resulting set from the PSO is empty or not. In the off-line phase, each possible collection of parties would agree on several keys that would be used to compute images of elements in sets of parties in the collection. This would be done in addition to the images computed by the common key k. Effectively, for each element that can be found in several input sets there would be many collisions in the data received by the decider. Dummies would be added in addition to this "cloning" of elements. Apart from images by the common key k, parties should skip computing images of a few elements with the other keys. This is to further confuse the true number of collisions from the number of observed collisions.

7 Performance Evaluation

In this section we evaluate the communication and computation complexities of our protocols. We have only implemented the most critical part of our protocols which is the cryptographic part, and we present the results of our experiments. Moreover, we do some comparison between the performance of our protocols with the prior works.

Please note that any set operation can be considered a boolean function, therefore, the number of possible set operations between n different sets is equal to the number of truth tables with n different variables, which is 2^{2^n}. In other words, when the number of input sets increases, the number of possible set operations increases more than exponentially. This means in order to evaluate the performance of Protocol 3 and protocol with keyed hash function, we cannot run the experiment with all the possible set operations (because there are many of those operations). Therefore, we choose one operation that we find interesting, and is compatible in size and the number of operations with the experiments in [18]: In equation 1, we assume that $\alpha = \beta = n$. We have compared the complexity of our Protocol 3 against the protocol 3 by Wang et al. in [18], because they already compared their protocol against the state of the art and showed that their protocol 3 is more feasible in practice than other solutions.

Table 1: Number of operations of our protocol 3 and the protocol 3 of [18]

Operations	Protocol of Wang et al. in [18]	Our Protocol-3
Encryptions	$O(nu + \alpha\beta u)$	$O(\alpha\beta u)$
Decryptions	$O(nu)$	$O(u)$

Moreover,the underlying idea of our protocol is close to that of the protocols in [18]. It is important to note that our setting with an external decider differs from the settings in [18] and other prior art. The difference in setting gives an opportunity to get more efficient solutions.

Wang et al.'s protocol is based on threshold ElGamal while our protocols use Paillier encryption scheme or any additively homomorphic encryption. We have listed the number of operations in Table 1. The entry for number of encryptions includes the number of re-encryptions. Both in ElGamal and in Paillier this is done by multiplying the encrypted value by a random encryption of zero. The speed of multiplication is very fast compared to encryption in both cryptosystems. Thus we have not listed the number of multiplications in the table. The number of operations in a single union or intersection can be calculated from the values of the table by substituting $\beta = 1$.

We also compare the number of communication rounds in our protocol against the protocol of Wang et al. In [18], the n parties first need to decide together on the key for threshold ElGamal. Then each party needs to individually take part in calculating the union of all sets. Then each party needs to individually take part in calculating the result and finally all parties together decrypt the result.

In our protocol the decider picks the key, then all other parties individually, in any order and maybe even partially simultaneously, modify the values in vectors W^k. Party P_n does the multiplications and sends Z to D that will decrypt it. Thus the number of communication rounds is much smaller in our solution.

The time measurements are obtained by running crypto operations in our protocols on an x86-64 Intel Core i5 processor clocked at 2.7 GHz with a 4 MB L3 cache. In Protocols 1, 2, and 3, any additively homomorphic non-deterministic encryption can be used. For our experiments, we use Paillier cryptosystem. The modulus that Wang et al. used in ElGamal was set to 512 bits, therefore, to be able to do a comparison we set the modulus N for Paillier cryptosystem to non-secure 512 bits as well. In practice, 512 bits is considered to be too short for a public key, because it can be factored. To be in secure side, N should be 2048 bits long. We made experiment when N is 2048 bits long and reported the results in Table 2.

We now compare the execution time of our Protocol 3 and the protocol 3 by Wang et al. in [18]. We first tested the cases where $u \in \{10, 20, 40, 60, 80, 100\}$ and there are 3 parties with input sets in the protocol. The result of our implementation showed that our protocol is 5 times faster than the protocol of Wang et al. For example, when $u = 100$, our protocol 3 needs 0.71 seconds to compute the outcome set, while protocol 3 of Wang et al. needs 3.7 seconds. We next

Table 2: Execution time of the on-line phase our Protocol-3 in seconds, when modulo N in Paillier is of length 2048 bits and $u \in \{2^2, 2^5, 2^7, 2^{10}\}$ and $n \in \{3, 5, 10, 15, 20\}$. In equation 1, we assume that $\alpha = \beta = n$. The numbers in the table are required time for each party to modify Z with a single thread. When $u = 2^2, 2^5, 2^7, 2^{10}$ the decider needs $0.02, 0.17, 0.68, 5.51$ seconds respectively, to decrypt this vector with 32 threads.

	$n=3$	$n=5$	$n=10$	$n=15$	$n=20$
$u=2^2$	0.001	0.002	0.003	0.005	0.007
$u=2^5$	0.008	0.013	0.025	0.039	0.05
$u=2^7$	0.031	0.05	0.1	0.15	0.2
$u=2^{10}$	0.237	0.391	0.786	1.178	1.56

compared the execution time of our Protocol 3 and the protocol 3 of Wang et al. in the cases that $n \in \{3, 5, 10, 15, 20\}$ and $u = 20$. In our experiments our Protocol 3 performs significantly faster than the protocol of Wang et al., and the speed difference increases with the number of participants. For example, when $n = 20$, our protocol needs 0.15 seconds, the protocol of Wang et al. needs 24 seconds to perform PSO. Therefore, when there are 20 parties with input sets, our protocol performs more that 150 times faster than Wang et al.'s protocol. Please note that we utilized the same computational power as Wang et al. used.

The evaluation of the keyed hash function is many orders of magnitude faster compare to public key encryptions. For instance, for a set size of one million, the computation of keyed hash values only takes one second.

As we mentioned before, our setting with a decider differs from the typical secret-sharing schemes and the server-aided PSO protocols. In fact in our variant of multi-party computation, the existance of the decider makes the protocol more efficient. For instance, the server-aided PSI protocol in [27] has the computation complexity of $O(u)$ and $O(nu)$ for each party and the server respectively, whereas in our PSI protocol (Protocol-2) the computation complexities are $O(u)$ for each party and also for the decider.

8 Security and Privacy Analysis

In this section, we present the security and privacy analysis of our generic protocols in the semi-honest setting.

In our protocols we assume that the parties communicate through a secure channel. Moreover, we also assume that the repository that the parties $P_1, ..., P_n$ use is accessible by them only. Moreover, this repository has a secure version control system to log the activities of its users [29].

In protocols with homomorphic encryption, if we assume that all parties are semi-honest, all vectors W^k are calculated correctly and the correct vector Z is sent to D. Also the result D gets after decrypting Z is the correct answer.

All values in the vectors in the repository are (very likely to be) different and no party P_i can decrypt them without the help of D and thus parties P_i do not learn anything from them.

The decider does not know the values in the repository and thus does not get any information from the parties in addition to the encrypted vector sent to it in the end of each protocol. Therefore, D does not learn anything else than the decrypted values of the vector that P_n send to D.

Thus in the semi-honest setting no party P_i learns anything and the decider only learns the end result of the protocol.

In the protocols with keyed hash function, D only receives the hash values of the items and dummy values. Because of the one-way property of the hash functions, D cannot derive the items from their hash values. D does not know the key and thus cannot even check if a certain element is in the resulting set or in the input set of some party. Also, hash values are indistinguishable from random dummy values, hence D cannot tell these two apart. Moreover, the dummy values hide the cardinality of the elements that are not in the output set. Addition of "cloned" hash images (for the emptiness protocol) hide the true cardinality of the output set and just reveals whether the cardinality is zero or positive.

9 Modified Protocol with One Malicious Party

We consider now the adversarial model in which we assume that there are no collusions between the parties. Please note that the material of this section is a hint of future work.

If we do not assume that the parties P_i are semi-honest, there are several ways how they can try to cheat: (i) They can use different inputs S_i (or $\bar{S}_i$) in different parts of the protocol (in different unions); (ii) They can use an incorrect complement for their set S_i; (iii) They can calculate the elements in W^k incorrectly: for instance, instead of multiplying the previous value by enc(0) they replace the element by enc(r); (iv) Party P_n can send an incorrect vector Z to the decider.

Our protocol can be made secure against these actions in the following way:

1) Each party P_i for each element in V, chooses one encryption of 0 and one encryption of 1, puts these two encrypted values in random order and sends them to all other parties, including D. The decider can confirm that, indeed, the pair is of the form $\{enc(0), enc(1)\}$. The party P_i then, for each pair, chooses a random number a, publishes the number to other parties except D, and raises both encrypted values to power a. The result is a different pair enc(0) and enc(a) with different a, for each element in the vector V. The party P_i uses only these values later in the protocol. All other parties can later make sure that these are calculated correctly.

2) We repeat our protocol n times and change the order of parties P_i such that everyone must be once in the role of P_1. When a party is in the role of P_1, they must use the values enc(0) and enc(1) they created in step 1) to initialize

the vector V. When they are initializing V, they must use the same values for the same element every time a different union is calculated and their input should be the same (S_1 or $\bar{S_1}$). Also, they should use different values for the same element when the inputs are different (one is S_1 and the other is $\bar{S_1}$). Now, the parties cannot cheat by using different inputs for S_i (or $\bar{S_i}$), or miscalculating the set complement.

3) In every repetition every party calculates the vector Z and sends it to the Decider. If every party has acted honestly in every repetition, then (i) everybody should send the same Z in every repetition; (ii) the decryption of Z would give the same result for each repetition. On the other hand, if some party has used different S_i in some repetition than what they used when they were playing the role of P_1, then there is a fair chance that the results from these two repetitions do not match.

4) The previous steps are likely to reveal whether an individual party has been cheating while others have stayed honest. However, it is harder to identify the cheater. The following can be done in order to locate where the cheating has happened. After n runs of the protocol every party independently multiplies together all elements in every vector W^k in every run that they know must be encryptions of zero if computations have been done correctly. Then they further multiply the result with a random encryption of zero before sending it to the Decider. The Decider decrypts everything and verifies that nothing else than zeros come out.

10 Conclusion

Private set operations such as private set intersection and private set union can be used in many privacy sensitive use cases, and therefore they have been studied extensively. In this work, we studied a special variant of PSO where the outcome of the multi-party PSO is learned by an external decider while the parties who have provided an input set to the PSO do not learn the outcome. By providing realistic examples, we showed the importance of the setting with a decider. Furthermore, we showed how our setting is different from what has been presented so far in academia.

We presented two generic solutions (Protocol 3 and protocol with keyed hash) to any PSO problems that the decider seeks the cardinality of the output set or wants to investigate whether this set is empty. Moreover, Protocol 3 is a generic protocol for the decider to learn also elements in the output set, under the assumption that the universe, from where parties choose elements to their sets, is limited in size. In addition to the cardinality and emptiness, the decider can receive the elements in the output set.

We presented the security and privacy analysis of our protocols in the semi-honest setting, and presented a modified protocol for the malicious setting. Lastly, we implemented our protocol. The result of our experiments showed that our solutions for our special setting, i.e., having an external decider, are more efficient than applying state of the art solutions proposed for other settings to

our setting. The experiments also show that our solutions are feasible for many real-life use cases.

We present different criteria that can be used to determine which PSO protocol fits a certain setting. The criteria of Section 3 can give a future direction to the research in the field of PSO protocols. We only covered the case with one malicious party. The case with more malicious parties is left for future work.

ACKNOWLEDGEMENT

This paper is supported by 5GFORCE project funded by Business Finland, and HELIOS H2020 project funded by the European Unions Horizon 2020 research and innovation programme under grant agreement No 825585.

References

1. Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, pages 241–257. Springer, 2005.
2. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.
3. Keith Frikken. Privacy-preserving set union. In *International Conference on Applied Cryptography and Network Security*, pages 237–252. Springer, 2007.
4. Sara Ramezanian, Tommi Meskanen, Masoud Naderpour, Ville Junnila, and Valtteri Niemi. Private membership test protocol with low communication complexity. *Digital Communications and Networks*, 6(3):321–332, 2020.
5. Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.
6. Mona FM Mursi, Ghazy MR Assassa, Ahmed Abdelhafez, and Kareem M Abo Samra. On the development of electronic voting: a survey. *International Journal of Computer Applications*, 61(16), 2013.
7. Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *USENIX security symposium*, volume 10, pages 95–110, 2010.
8. Yaniv Erlich and Arvind Narayanan. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics*, 15(6):409–421, 2014.
9. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.
10. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication, 1997.
11. James M Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 198:1, 2008.
12. Sara Ramezanian, Tommi Meskanen, and Valtteri Niemi. Parental control with edge computing and 5g networks. In *2021 29th Conference of Open Innovations Association (FRUCT)*, pages 290–300. IEEE, 2021.

13. Tamara Dugan and Xukai Zou. A survey of secure multiparty computation protocols for privacy preserving genetic tests. In *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 173–182. IEEE, 2016.
14. Ming Li, Ning Cao, Shucheng Yu, and Wenjing Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *2011 Proceedings IEEE INFOCOM*, pages 2435–2443. IEEE, 2011.
15. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1257–1272, 2017.
16. Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 154–185. Springer, 2019.
17. Ji Young Chun, Dowon Hong, Ik Rae Jeong, and Dong Hoon Lee. Privacy-preserving disjunctive normal form operations on distributed sets. *Information Sciences*, 231:113–122, 2013.
18. Wenli Wang, Shundong Li, Jiawei Dou, and Runmeng Du. Privacy-preserving mixed set operations. *Information Sciences*, 525:67–81, 2020.
19. Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 554–563, 1994.
20. Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems*, pages 174–183. IEEE, 1997.
21. Leonard Assouline and Tianren Liu. Multi-party psm, revisited. *IACR Cryptol. ePrint Arch.*, 2019:657, 2019.
22. Amos Beimel, Eyal Kushilevitz, and Pnina Nissim. The complexity of multiparty psm protocols and related models. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 287–318. Springer, 2018.
23. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.
24. Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 578–602. Springer, 2014.
25. Milad Bahadori and Kimmo Järvinen. A programmable soc-based accelerator for privacy-enhancing technologies and functional encryption. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(10):2182–2195, 2020.
26. Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Scaling private set intersection to billion-element sets. In *International Conference on Financial Cryptography and Data Security*, pages 195–215. Springer, 2014.
27. En Zhang, Fenghua Li, Ben Niu, and Yanchao Wang. Server-aided private set intersection based on reputation. *Information Sciences*, 387:180–194, 2017.
28. Wenliang Du and Mikhail J Atallah. Secure multi-party computation problems and their applications: a review and open problems. In *Proceedings of the 2001 workshop on New security paradigms*, pages 13–22, 2001.
29. Paul Guthrie, Andrew Dale, Michael Tolson, and Chistopher Buchanan. Distributed secure repository, March 16 2006. US Patent App. 10/943,495.