Computing Private Set Operations with Linear Complexities

Alex Davidson and Carlos Cid

Royal Holloway, University of London
{carlos.cid, alex.davidson.2014}@rhul.ac.uk

Abstract. Private set operation (PSO) protocols provide a natural way of securely performing operations on data sets, such that crucial details of the input sets are not revealed. Such protocols have an ever-increasing number of practical applications, particularly when implementing privacy-preserving data mining schemes. Protocols for computing private set operations have been prevalent in multi-party computation literature over the past decade, and in the case of private set intersection (PSI), have become practically feasible to run in real applications. In contrast, other set operations such as union have received less attention from the research community, and the few existing designs are often limited in their feasibility. In this work we aim to fill this gap, and present a new technique using Bloom filter data structures and additive homomorphic encryption to develop the first private set union (PSU) protocol with both linear computation and communication complexities. The overheads of our protocol scale better with increasing set sizes than existing PSU designs and we thus provide the first potentially practical realisation of the PSU functionality. Moreover, we show how to adapt this protocol to give novel ways of computing PSI and private set intersection/union cardinality (PSI/PSU-CA). The resulting schemes have complexities that are comparable to current solutions that are considered practical. We therefore present an adaptable way for efficiently computing the main set operations, with linear complexities, and with the possibility of extending to compute other more complex functionalities. Our constructions can be proven secure with respect to both semi-honest and malicious adversaries.

Keywords: Private set operations, Bloom filters, additively homomorphic encryption, secure computation, data mining

1 Introduction

The emergence of Big Data has resulted in an increasing need for analytical techniques such as data mining that allow entities to gain information from the large data sets they own. Even more can however be learnt by combining internal data sets with other entities, although to do this privacy-preserving measures must be put in place to stop data being leaked to competitors or untrusted parties.

Private set operation (PSO) protocols provide a natural way of securely performing operations on these combined data sets, such that crucial details of the different input sets are not revealed to participating parties. In the last decade research into private set intersection (PSI) protocols has resulted in designs that are practically feasible for real-world use. Particularly, the works of [14, 30, 31] have shown that certain techniques and data structures like oblivious transfer (OT) and Bloom filters can be used to design protocols that scale and perform well, even for huge data sets (e.g containing billions of items). These constructions are crucial for developing large-scale data mining applications where data privacy and efficient computation are important for practical usage. For example computations over genetic data as shown in [20] may require comparing records from databases with hundreds of millions of elements.

Despite of much recent progress in the design of PSI protocols, research into performing other set operations in similar secure way has not been as comprehensive, and thus has not been able to provide protocols that meet the same efficiency requirements. For instance, the state of the art in private set union (PSU) protocols requires a quadratic number of exponentiations in the size of the sets involved in the computation. More generally, Kissner and Song [26] introduced an adaptable way of computing multiple PSOs with quadratic overheads for communication and computation; no other work since has managed to improve on these. This means that complex data mining applications that rely on the computation of multiple set operations is likely to require implementation of inefficient protocols. We address this problem in this paper, by proposing a method for implementing different privacy-preserving set operations with linear overheads for both computation and communication.

1.1 Applications of PSOs

Private set operation protocols have an ever-increasing number of practical applications, particularly in privacy-preserving data mining schemes. We provide below examples of applications for the main types of set operations.

1. **Law enforcement:** An airline must alert law enforcement authorities if anyone who has been placed on a no-flight list attempts to board a flight. The airline may not want to give away customer data unnecessarily and the authorities must not give away data on criminals to the airlines. A private set intersection (PSI) protocol allows the two entities to compare the sets of people they are aware of and learn only the intersection.
2. **International crime:** Two or more states may want to combine the information they have about criminals operating on an international scale. It benefits all parties to know about all the individuals operating at this level as they could target any of the countries involved. The entities can use a private set union (PSU) protocol to learn all of their combined data.
3. **Changes in customer base:** An industry sector wants to evaluate whether the customer base has grown or declined over a certain time period. To do this the major entities in the sector must combine the number of customers

they have currently together. The companies can enact a private set union cardinality (PSU-CA) protocol on their set of customers to learn the combined number of customers in the sector.

4. **Ad-space usage:** An online marketplace selling ad-spaces on their web platform seeks to find out how many new customers current ad-space purchasers gain to give figures out to prospective buyers. In this case the entities will look to perform a private set intersection cardinality (PSI-CA) protocol to learn the number of common customers that they both have.

Further concrete examples of the applicability of PSOs range from proximity testing [28] to botnet discovery [27] to implementation of mediating mechanisms in cyber security information sharing schemes [25]. All these examples highlight the need for efficient and scalable techniques for computing different set operations, particularly on large data sets.

1.2 Our contributions

We first address the void in efficient PSU protocols by developing a new two-party construction, secure against semi-honest adversaries. The design is based on an inverted representation of Bloom filters alongside the usage of an additively homomorphic encryption scheme. We extend previous works by [16, 17, 19] by utilising a similar encryption technique but make use of the optimal data structure provided by Bloom filters to gain better complexities for computation. Our PSU protocol is the first to demonstrate both linear computation and communication complexities. Furthermore, similar to previous private union protocols of [2, 17, 26], we show in the appendix how to adapt our protocol to retain security in the malicious model, whilst retaining the linear complexities that we achieved in our semi-honest design. To do this we require an authorisation step by a third party for authorising the client's set (similar to the APSI designs highlighted in [10, 11, 13]), and a homomorphic signature scheme that allows for polynomial operations and satisfies a context-hiding privacy requirement.

Moreover, we show how the simplicity of our PSU proposal means that we can adapt our protocol to compute other fundamental set operations, such as PSI or PSI/PSU-CA. These constructions also have linear complexities putting them in line with current practical solutions in the wider research area. These adaptations are achieved with minimal changes in the design structure and result in a suite of protocols that enables computation of all the major PSOs. Consequently we have devised an easily implementable technique that allows developers to create a library of PSOs with very similar functional requirements. We see this as an advantage for privacy-preserving data mining applications that require the running of multiple efficient instantiations of PSOs.

Our adaptable construction can be seen as an extension of previous work by [26] and subsequent attempts by [2, 13, 19] to provide multiple PSOs. Our work is the first to provide a scalable solution to this problem, with the first PSU protocol with linear asymptotic efficiency, along with very similar designs for PSI and PSI/PSU-CA with almost identical efficiency. Our design plausibly

lends itself to further extensions for computing more complex set operations while retaining efficiency and security, for instance it seems likely that we could implement the more granular threshold designs shown in [2, 26]. We consider this an interesting area of future work. The simplicity of our toolkit lies in the need for just an instantiation of an additive and a multiplicative homomorphic encryption scheme alongside a Bloom filter implementation.

1.3 Layout of work

In Section 2 we provide a summary of the current standard of PSO protocols along with the tools we use to develop our solution. Section 3 presents our PSU protocol, while Section 4 demonstrates how to adapt our PSU design to compute PSI and PSI/PSU-CA with similar performance. Finally, it should be noted that in Appendix G we discuss the alterations that can be made to realise malicious security for our PSU design (resulting in an authenticated variant i.e. APSU) and then in Appendix H we give a proof of our maliciously secure protocol.

2 Background

2.1 Notation

We will primarily consider two-party protocols with players P_1 and P_2 who own sets S_1 and S_2, respectively. We commonly denote the cardinalities of the private sets by $n = |S_1|$ and $m = |S_2|$. We denote the domain of elements by E, the security parameter by λ and, for multi-party protocols, the number of players by N, where $c < N$ denotes the number of corrupted players in a protocol instantiation. We refer to B as the length of the Bloom filters that we use, while k refers to the number of corresponding hash functions. Appendix B provides an explanation of how Bloom filters work and how they are constructed, including how to optimally choose parameters for efficiency. When discussing the use of homomorphic operations over ciphertexts, we use $+_H$ when invoking additions and $\times_H$ on the invocation of multiplication. Appendix C fully describes our notation regarding partially homomorphic encryption (PHE) schemes.

2.2 Current methods for PSOs

PSI protocols Freedman et al. were the first to develop a protocol for specifically tackling PSI in [16]. Their design was based on oblivious polynomial evaluation (OPE) where they use additively homomorphic encryption in order to realise the OPE functionality. Their initial design has communication of $O(n)$ bits and computation of $O(n^2)$ exponentiations, however they leverage a hashing optimisation to get computation complexities of $O(n \log \log n)$. While works such as [12, 19, 26] followed with newer designs based on OPE, all the constructions have been shown to require at least a sub-quadratic amount of exponentiations.

Since then research has focused on achieving PSI with more efficient constructions. De Cristofaro et al. were the first to demonstrate the possibility of

a practical design in [10, 11], notably featuring linear complexities of $O(n+m)$ in both computation of exponentiations and in communication. Their protocols were based on a public-key technique that they called 'blind-RSA', and was shown to be secure with respect to semi-honest and malicious adversaries.

Later, Huang et al. showed in [21] that using garbled circuits they could achieve a semi-honest secure protocol that ran in comparably quick times to [11], and quicker in settings where very large keys were used (e.g. 256 bits). The focus on experimentation was carried over by Dong et al. in [14] who used a technique named 'Garbled Bloom filters' to construct a PSI protocol achieving complexities favourable to [11, 21], with runtime performance several orders of magnitude faster for both short and long security levels. Their protocol utilises parallelisation and fast operations to make significant gains in runtimes, with good performance even when sets contain ~ 2 million elements.

The most recent and efficient works have come from Pinkas et al. in [30] (with a further optimisation in [31]). Their protocols are based on oblivious transfer (OT) and makes usage of the fact that OT extension allows for a polynomial amount of OTs to be generated from an initial much smaller amount of OTs (see [22]). Using various external hashing optimisations, they show that their protocols run faster than any previous designs. Tables 4 and 5 in Appendix A display values taken from [30] demonstrating the runtimes and communication demands of these protocols when run on the same platforms over sets of sizes $2^{10} - 2^{18}$. The low run-times of the protocols coupled with the minimal effect of scaling set sizes, especially for the works of [14] and [30], establishes the applicability of these protocols to real-world scenarios.

PSU protocols Work on PSU protocols has been relatively scarce in comparison, with most protocols being derived as part of a generic construction for multiple PSOs. The work of [26] was the first to develop multi-party protocols for all of the main set operations, additionally including some more specific primitives that included thresholds into multi-set union and intersection protocols. Their work is based on OPE and demonstrates communication of $O(cNn \log |E|)$ with quadratic computation $O(nm)$. Both of these complexities are unfavourable and highlight the scalability problem with OPE-based protocols.

Additionally, Brickell and Shmatikov [8] devise two PSU protocols, although both require communication and computation of $O((n+m) \log |E|)$ and are thus too inefficient to be considered as practical for large set sizes. Their most efficient design requires $O((n+m) \log |E|)$ garbled circuit evaluations for computing the minimum of two values across a total of $O((n+m) \log |E|)$ communication rounds. Furthermore it requires that both parties assign an ordering to the entire universe of set elements E which requires the universe to be polynomially-sized which is a limitation on the design. It should be noted however that the PSU protocol that the authors design hides the set cardinality of both players (unlike the other constructions that we consider here) and additionally the output of the union is provided for both players, standing in contrast to our protocol alongside other PSU designs such as [17].

Since then Frikken [17] proposed a OPE-based PSU protocol very similar to the design of [16], with identical complexities. Hazay et al. in [19] make use of an oblivious pseudorandom function to build on the protocol of [16] and generate a maliciously secure protocol, however the complexities are not competitive with semi-honest protocols.

More recently Blanton et al. [2] focus on multi-party PSO protocols that meet composability guarantees. Their work achieves only quadratic complexities for both communication and computation, using similar techniques to the work of [8] except that the ordering of elements in the universe is no longer required. The protocols also achieve PSU and other operations where the elements are unknown to the computing parties (the inputs are secret shared), therefore the technique is better suited to an outsourced computation setting.

Table 6 in Appendix A compares the complexities of these main constructions, the works [26] and [2] display complexities that are relevant in the multi-party case. Other constructions for constructing PSU protocols, such as [32], do exist but none of these have managed to capture linear complexities in their design.

PSI/PSU-CA protocols Two of the most efficient protocols for computing set cardinality are the public-key based design by De Cristofaro et al. in [9] and the multi-party Bloom filter based construction by Egert et al. in [15]. The complexities of the designs for the two-party case are shown in Table 7. Both designs showcase linear complexities and can therefore be thought of as practical designs for PSI/PSU-CA. The reason for this efficiency is undoubtedly the usage of techniques such as those shown in [11, 14] to aid the construction.

Encrypted Bloom filters Our constructions rely on the encryption of each entry in a Bloom filter (see Appendix B for an explanation) with a partially homomorphic encryption scheme that allows another player to obliviously calculate the status of their elements regarding the original set. The creator of the Bloom filter can then decrypt these computations and learn elements depending on the functionality of the protocol.

The work of [1] defined a form of encrypted Bloom filter where the encryption was applied to the output of the hash functions. Boneh et al. presented their own work detailing how encrypted Bloom filters could be used for private information retrieval in [5] using the encryption scheme of [4].

Kerschbaum later developed a PSI protocol (secure in the semi-honest and malicious settings) in [23] along with a work guaranteeing supply chain integrity in [24]. Both works use encrypted Bloom filters, with [23] using the Goldwasser-Micali (GM) scheme to perform these encryptions. To test for intersection, the server is required to compute k multiplications for each of the m elements in their set. The design incurs a ciphertext element expansion of k, and P_1 is required to perform a total of km decryptions. We avoid these overheads in our protocols. It should also be noted that the malicious designs of [23] do not protect against the possibility of a malicious server preventing P_1 from receiving the correct

output. Our malicious protocol in Section G ensures that correctness of output is achieved as well as privacy for both players.

The most recent work to involve the use of encrypted Bloom filters is by Debnath et al. in [13]. The scheme that they propose uses GM encryption in the same way as [23] to encrypt each entry of an inverted Bloom filter. They then show how it is possible to build PSI-CA and PSI protocols for two parties. Their work requires that the set of both parties be represented as a bit string (since GM encryption operates over single bits) resulting in very large strings for large sets. Coupling this with the requirement for a quadratic number XOR operations in the string length leads to unfavourable computational demands. Moreover, their protocol is not proven secure for a malicious server (i.e. P_2 in our case). It should be noted that none of the previous designs using this technique have addressed PSU protocols or versatile tools for computing all of the main PSOs detailed here.

2.3 Security model for PSOs

Protocols for computing PSOs can be proven secure with respect to either semi-honest or malicious adversaries. Before we show what this means we first define the notion of computational indistinguishability for probability distributions:

Definition 1. *Let $X = \{\mathcal{X}_\lambda\}_{\lambda \in S}$ and $Y = \{\mathcal{Y}_\lambda\}_{\lambda \in S}$ be probability ensembles indexed by S. We say that these ensembles are computationally indistinguishable for all probabilistic polynomial time (PPT) algorithms, $\{\mathcal{D}_n\}_{n \in \mathbb{N}}$, if there exists a negligible function $\mathsf{negl} : \mathbb{N} \mapsto [0, 1]$ where*

$$|\Pr[\mathcal{D}_n(\lambda, \mathcal{X}) = 1] - \Pr[\mathcal{D}_n(\lambda, \mathcal{Y}) = 1]| < \mathsf{negl}(n)$$

and we write $X \simeq Y$ to denote this.

Semi-honest security Now assume that we have a protocol π that is required to securely represent a specific polynomial-time functionality f. Let S_i be the input set of P_i for $i \in \{1, 2\}$ and let aux_i be a set of auxiliary information that P_i holds (we specifically consider the case where f computes an operation over input sets). For each P_i define the view of the protocol for P_i to be $\mathsf{view}_i^\pi(S_1, S_2) = (\mathsf{Inp}_i, r_i, msg_i, \pi(S_1, S_2))$ where $\mathsf{Inp}_i = (S_i, aux_i)$ is the combined input of P_i to π, r_i are the internal coin tosses of P_i, msg_i is the messages viewed by P_i in the protocol and $\pi(S_1, S_2)_i$ is the output witnessed by P_i. Then we can formulate the following definition for semi-honest adversaries.

Definition 2. *Protocol π securely computes the functionality f in the presence of static semi-honest adversaries if there exists polynomial-time simulators Sim_1, Sim_2 where*

$$\{Sim_1(\mathsf{Inp}_1, f(S_1, S_2))\} \simeq \{\mathsf{view}_1^\pi(S_1, S_2)\}$$

$$\{Sim_2(\mathsf{Inp}_2, f(S_1, S_2))\} \simeq \{\mathsf{view}_2^\pi(S_1, S_2)\}$$

Intuitively, this states that each party's view of the protocol can be simulated using only the input they hold and the output that they receive from the protocol. Therefore a corrupted P_1 or P_2 is unable to learn any extra information that cannot be derived from the input and output explicitly.

Malicious security For the malicious case we have to argue slightly differently. Let $\mathsf{Real}_i^{\pi}(S_1, S_2)$ be the view of P_i in the real execution of the protocol π and let $\mathsf{Ideal}_i^{\pi}(S_1, S_2)$ be the view of P_i in the ideal execution where all inputs Inp_i are submitted to directly to the functionality f by a trusted party who then returns the output $f(S_1, S_2)$ to the relevant player. Informally we need to show that any attack that a corrupted P_i can perform in the real execution can also be performed in the ideal world and thus we get the following definition

Definition 3. *Protocol π securely computes the functionality f in the presence of malicious adversaries if we have that*

$$\{\mathsf{Real}_1^{\pi}(S_1, S_2)\} \simeq \{\mathsf{Ideal}_1^{\pi}(S_1, S_2)\}$$

$$\{\mathsf{Real}_2^{\pi}(S_1, S_2)\} \simeq \{\mathsf{Ideal}_2^{\pi}(S_1, S_2)\}$$

3 PSU protocol

The current designs in Section 2.2 highlight certain ways of developing protocols that lead to practical designs. One of these ways was to use Bloom filters where [14] and [15] used the data structure to develop protocols with linear complexities and, in the PSI case, with demonstrably quick run-times even with large set sizes.

Our protocol is the first to use Bloom filters to retrieve PSU on two sets. We achieve this by encrypting the Bloom filter using an additively homomorphic encryption (AHE) scheme, appendices C and D give descriptions of an AHE scheme and in particular the Paillier encryption scheme that allows for these operations over ciphertexts. This form of encryption allows receiving parties to obliviously evaluate their elements with respect to the Bloom filter.

We also use a non-standard representation of a Bloom filter by inverting each entry prior to encryption. This allows us to compute PSU without the need for fully homomorphic operations which require heavy computation and large parameters in practice for current encryption schemes. In the following we formalise the notions of encrypted and inverted Bloom filters. Both techniques are crucial to our protocol design.

Definition 4. *Let $(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$ be a public-key cryptosystem and $(pk, sk) \leftarrow \mathcal{K}$ be generated at random from the key space $\mathcal{K}$. Let the Bloom filter calculated by P_i for the set S_i be denoted as $\boldsymbol{BF}_i$ and have B entries such that $\forall\ b \in \{1, \ldots, B\}$ we have $\boldsymbol{BF}_i[b] \in \{0, 1\}$. The corresponding encrypted Bloom filter is denoted by $\boldsymbol{EBF}_i$[1] and has B entries where each entry is defined in the following way:*

$$\boldsymbol{EBF}_i[b] = E_{pk}(\boldsymbol{BF}_i[b])$$

[1] When referring to an encrypted and inverted Bloom filter we will write $\mathbf{EIBF}_i$.

where $E_{pk}(x)$ denotes the encryption of $x \in \mathcal{P}$ with respect to the public key pk. In the following we define $\boldsymbol{EBF}_i = \{C[1], \ldots, C[B]\}$ and for $y_j \in S_i$, then $\boldsymbol{EBF}_i[h_u(y_j)] = C_u^{(j)}$ for $u = \{1, \ldots, k\}$ and where h_u is the u^{th} hash function used in computing the original Bloom filter.

Definition 5. *Let $\boldsymbol{BF}_j$ be a Bloom filter that represents the set of party P_j. We define the corresponding inverted Bloom filter to be $\boldsymbol{IBF}_j$ where*

$$\boldsymbol{IBF}_j[i] = \begin{cases} 1 \text{ if } \boldsymbol{BF}_j[i] = 0 \\ 0 \text{ otherwise.} \end{cases}$$

The premise of the encrypted Bloom filter is that if we encrypt with an IND-CPA encryption scheme then another party will be unable to distinguish from an encrypted '0' or '1' entry. To make the encryption of the Bloom filter well-defined we use 0 and 1 entries (where 1 is the identity element) from a ring instead of using bits, this allows us to use Paillier encryption where we use the plaintext ring $\mathbb{Z}_N$ for $N = p \cdot q$ for primes p, q. We provide a more thorough discussion of the scheme in appendix D and we will denote these ring elements by 0 and 1 throughout. The usage of this encryption method allows the creator of the Bloom filter to send out **EBF**/**EIBF** and not have to worry about the receiver being able to infer anything from it. In the next section we will show how our PSU protocol, $\pi_{\cup}^{\text{EBF}}$, uses the security of the encryption to prevent the server from learning extra information about the client's set.

3.1 Preliminaries

Both parties are given the k hash functions which are chosen to evaluate the Bloom filter for elements in the sets. The elements $y_j \in S_2$ are assumed to be represented by elements in $\mathbb{Z}_N$ as they are in the work of [17]. We reiterate here that we will be using the optimal Bloom filter parameters detailed in Appendix B in Equations (6) and (7).

Additionally we assume that P_1 has a public key pk which is made available to P_2. P_1 also has a secret key sk that she uses for decryption. Both parties also have access to sources of internal randomness that they can use for computing any tasks that require it such as encryption and sampling random values. We additionally only assume that P_1 is aware of the size of S_2, meaning that P_2 (the server) remains unaware of $|S_1|$. The following is a description of how the protocol works; in Figure 1 in Appendix A we provide an overview of our design.

3.2 Protocol steps

Inputs - P_1: [(pk, sk), S_1, $|S_2|$], P_2: [pk, S_2]

1. P_1 calculates $\mathbf{BF}_1$ representing S_1 by evaluating $h_1, \ldots, h_k$ on their set of elements. She then inverts each entry in $\mathbf{BF}_1$ to retrieve $\mathbf{IBF}_1$.

2. P_1 separately encrypts each element $\mathbf{IBF}_1[l]$ of the inverted Bloom filter, where $1 \leq l \leq B$, using pk. P_1 now possesses $\mathbf{EIBF}_1$, where $\mathbf{EIBF}_1[l] = C[l]$. She sends $\mathbf{EIBF}_1$ to P_2.
3. P_2 evaluates each element $y_j \in S_2$ using the k hash functions and retrieves $\{C_1^{(j)}, \ldots, C_k^{(j)}\}$ where $C_d^{(j)} = \mathbf{EIBF}_1[h_d(y_j)]$ for $j \in \{1, \ldots, m\}$.
4. P_2 computes $c_j = (C_1^{(j)} +_H \ldots +_H C_k^{(j)})$ and sends $(\tilde{p}_j = c_j \cdot y_j, c_j)$ to P_1.
5. P_1 receives $(\tilde{p}_j,\ c_j)$. First she checks the value of c_j by computing $D_{sk}(c_j) = q_j$. If $q_j = 0$ then $D_{sk}(c_j \cdot y_j) = 0$ so nothing can be learnt and she does not decrypt. Else $D_{sk}(c_j \cdot y_j) = z_j \cdot y_j = p_j$, where she does decrypt.
6. P_1 computes ${z_j}^{-1}$ for $q_j \neq 0$ and then calculates $p_j \cdot {z_j}^{-1} = y_j$
7. P_1 adds all y_j to the set V where the corresponding $q_j \neq 0$ and outputs the set $S_1 \cup V$.

Remark 1. As we describe in Appendix D, we adopt the notation $c_j \cdot y_j$ for scalar multiplication with scalar y_j. This preserves the generality of the protocol relative to the AHE scheme used, for Paillier encryption however this multiplication would usually be invoked via an exponentiation, i.e. $c_j^{y_j}$

Remark 2. It should be noted that the protocol leaks the size of the intersection cardinality between the players P_1 and P_2, however this is similar to the previous PSU designs of [2, 17, 19].

3.3 Protocol correctness

Since the Bloom filter is inverted before encryption then for any $y_j \in S_1 \cap S_2$ we have that $D_{sk}(c_j) = 0$, therefore any message $c_j \cdot y_j$ that is received for such a y_j also decrypts to 0 and so cannot be learnt. For a value $y_j \notin S_1$ then we have that $D_{sk}(c_j) = z_j \in \mathbb{N}$, then decrypting $c_j \cdot y_j$ reveals $z_j \cdot y_j$ and then P_1 can add all values y_j to V by multiplying by z_j^{-1}. Since V contains all values $(y_j \in S_2) \wedge (y_j \notin S_1)$ then $S_1 \cup V = S_1 \cup S_2$. Clearly correctness is not perfect due to the possibility of false positives occurring in the Bloom filter though we can make this negligible in k as discussed in Appendix B.

3.4 Protocol security

We show that this protocol is secure with respect to the ideal functionality of a PSU computation defined by $\mathcal{F}_\cup$ as in definition 2. For two parties P_1 and P_2 with sets S_1, S_2 respectively we define the functionality for the definition to be:

$$\mathcal{F}_\cup(S_1, S_2) = S_1 \cup S_2 \tag{1}$$

As the definition suggests we need to show that it is impossible to derive anything from the execution of the protocol that is not implied by possession of the input and output of the corrupted player in question.

Theorem 1. *Suppose that the protocol, $\pi_\cup^{EBF}$, is instantiated with an IND-CPA secure AHE scheme. Then $\pi_\cup^{EBF}$ securely realises $\mathcal{F}_\cup$, as in Equation* (1)*, in the presence of static semi-honest adversaries.*

Proof. See Appendix E.1.

3.5 Performance analysis

Communication complexity Firstly we note that the protocol requires two rounds (assuming the messages from P_2 are sent in parallel). Once when P_1 sends $\mathbf{EIBF}_1$ to P_2 and again when P_2 sends $(c_j \cdot y_j, c_j)$ back to P_1.

In the first round of communication P_1 sends over the encrypted values of the Bloom filter that she computed. Therefore P_1 sends over B ciphertexts in total as we have one ciphertext for each entry in the Bloom filter.

In the second round P_2 sends m different sets of $(\tilde{p}_j, c_j)$, where $m = |S_2|$. Both $\tilde{p}_j$ and c_j are ciphertexts and so in this round we send over $2m$ ciphertexts. Overall the communication complexity is equal to sending a total of $(2m+B)$ ciphertexts. By choice of optimal Bloom filter parameters as shown in Appendix B in Equations (5) and (6) and by choice of $n = m$ (i.e. equal set sizes) we get that the communication is $O(n)$ bits and so linear in the size of the sets.

Our protocol outperforms those shown in [2, 26] where the communication scales quadratically as the size of the sets held by the parties is increased. It is comparable with the communication complexity of the two-party protocol in [17] that is also $O(n)$.

Computational complexity P_1 computes B encryptions and $2m$ decryptions (in the worst case). P_1 must also compute m inverses of group elements, though techniques for doing this are very efficient. In practice, we can also reduce the number of decryptions by not computing $D_{sk}(c_j \cdot y_j)$ if $D_{sk}(c_j) = 0$. On average this will lead to savings that are proportional to the size of the intersection.

Alternatively, P_2 will compute $m(k-1)$ homomorphic additions which corresponds to computing $m(k-1)$ exponentiations. It is clear that work done by both parties is linear in $m = |S_2|$ and by assuming that $n = m$ we get that computation is $O(n)$ exponentiations. We know this because the value k depends only on the desired false-positive probability of the Bloom filter (see Equation (5)) and so it is a constant factor. The protocols of [2, 8, 17, 19, 26] all exhibit computational complexity which is quadratic in the sizes of the sets and so these protocols will not scale as well as our protocol for larger set sizes. Table 1 highlights the comparison between our protocol and the previous designs in terms of computational and communication efficiency.

	Communication	Computation
Kissner et al. [26]	$O(N^2 n \log \lvert E \rvert)$	$O(n^2)$
Frikken [17]	$O(n)$	$O(n \log \log n)$
Brickell et al.[8]	$O((n+m) \log \lvert E \rvert)$	$O((n+m) \log \lvert E \rvert)$
Blanton et al. [2]	$O(N^2 M \log M)$	$O(N^2 M \log M)$
$\pi_{\cup}^{\mathrm{EBF}}$	$\boldsymbol{O(n)}$	$\boldsymbol{O(n)}$

Table 1. Comparison of $\pi_{\cup}^{\mathrm{EBF}}$ complexities with prior designs. All designs apart from [2] refer to the amount of exponentiations in terms of computation.

4 Adaptations to PSI and PSI/PSU-CA

An interesting outcome of our simple protocol construction is the ease that we can adapt the design to securely compute different set operations. Here we consider the widely used operations PSI and PSI/PSU-CA and how we can adapt our technique for securely computing PSU to compute these functionalities instead. These adaptations highlight how our protocol design resembles a plausible technique for being able to compute multiple set operations, in the same vein as [26], while achieving linear asymptotic overheads. We define the ideal functionalities for PSI ($\mathcal{F}_{\cap}$) and PSI-CA ($\mathcal{F}_{|\cap|}$) as:

$$\mathcal{F}_{\cap}(S_1, S_2) = S_1 \cap S_2 \tag{2}$$

$$\mathcal{F}_{|\cap|}(S_1, S_2) = |S_1 \cap S_2| \tag{3}$$

(with $\mathcal{F}_{|\cup|}$ defined analogously). We will prove the security of our designs with respect to these functionalities.

4.1 PSI protocol

Our PSI protocol ($\pi_{\cap}^{\text{EBF}}$) operates without requiring the inverted Bloom filter that we require for our PSU protocol $\pi_{\cup}^{\text{EBF}}$. For this design we require the usage of a multiplicatively homomorphic encryption (MHE) scheme (such as ElGamal) where the multiplication operation, $\times_H$, is defined analogously to $+_H$. P_2 now computes $c_j = C_1^{(j)} \times_H \ldots \times_H C_k^{(j)}$ for each $y_j \in S_2$ after receiving $\mathbf{EBF}_1$ from P_1. Since P_1 no longer computes an inversion before encryption we have the following property for c_j:

$$c_j = \begin{cases} E_{pk}(0) \text{ if } y_j \notin S_1 \\ E_{pk}(1) \text{ if } y_j \in S_1 \end{cases}$$

Now P_2 can compute the message $c_j \times_H E_{pk}(y_j)$ for each y_j to send back to P_1. Thus P_1 can decrypt and retrieve y_j if and only if it belongs to the intersection of S_1 and S_2 (except for negligible chance of false-positives occurring). It is clear to see that we now only need homomorphic multiplication operations to complete these messages instead of the addition operations, fortunately there are many schemes that allow for this operation with the computation of exponentiations still the only necessity (e.g. ElGamal). We provide an overview of our protocol $\pi_{\cap}^{\text{EBF}}$ in Figure 2 in Appendix A.

For this scheme to be well-defined we need to also encrypt the element y_j before we multiply which incurs extra computation per element. The effect of this is captured by the asymptotic performance of the protocol on the whole however. Alternatively, we could define this protocol with an additive homomorphic scheme by using Bloom filter inversion and calculating c_j in the same way as the PSU protocol before adding an encryption y_j. However, in this setting we would need to use randomness to mask values that are not in the intersection. Nevertheless, this could be preferable in a situation where an implementer would favour instantiating these set operations with only a single AHE scheme.

Protocol correctness The correctness of the protocol follows since P_1 outputs those y_j such that $c_j = E_{pk}(1)$ after decrypting c_j, since this allows for P_1 to decrypt $\tilde{p}_j$ to retrieve y_j. This follows since $D_{sk}(\tilde{p}_j) = y_j$ if $D_{sk}(c_j) = 1$. We know these values form the intersection of the sets by the construction of $\mathbf{BF}_1$.

Protocol security Like our proof for the protocol $\pi_{\cup}^{\text{EBF}}$ we need to prove that this protocol securely realises the ideal functionality. Therefore we give the following theorem and provide a proof in Appendix E.2 (the framework is very similar to the proof of Theorem 1).

Theorem 2. *Suppose that the protocol, $\pi_{\cap}^{EBF}$, is instantiated with an IND-CPA secure, MHE scheme. Then $\pi_{\cap}^{EBF}$ securely realises $\mathcal{F}_{\cap}$ as in Equation* (2) *in the presence of static semi-honest adversaries.*

Proof. See Appendix E.2.

4.2 PSI/PSU-CA protocol

We can make use of the fact that by calculating one of PSI-CA or PSU-CA then we can calculate the other using the following relation:

$$|X \cap Y| = |X| + |Y| - |X \cup Y| \tag{4}$$

and thus we can concentrate on only computing one of the operations. We can create a secure protocol, $\pi_{|\cap|}^{\text{EBF}}$, for calculating PSI-CA by adapting the protocol $\pi_{\cup}^{\text{EBF}}$ to have P_2 to just send the message (c_j) where c_j is calculated in the same way as the previous protocols. Like the PSU protocol $\pi_{\cup}^{\text{EBF}}$ we can use Paillier's scheme to compute c_j additively. If we compute c_j multiplicatively then we can use a scheme like ElGamal. Both schemes lead to computation of a linear number of exponentiations and so it comes down to the implementer's discretion.

The protocol proceeds in the same way except that P_1 only decrypts c_j. If $D_{sk}(c_j) = 0$ then increments a counter c by one. Once all c_j have been decrypted then P_1 outputs c as the answer. For PSU-CA we compute the number, c, of c_j that do not decrypt to 0 and then output $|S_1| + c$.

Protocol correctness Correctness is satisfied since $D_{sk}(c_j) = 0$ if and only if $y_j \in S_1$ (and thus $y_j \in S_1 \cap S_2$) with all but the negligible probability of a false positive occurring.

Protocol security

Theorem 3. *Suppose that the protocol, $\pi_{|\cap|}^{EBF}$, is instantiated with an IND-CPA secure AHE scheme. Then $\pi_{|\cap|}^{EBF}$ securely realises $\mathcal{F}_{|\cap|}$, as in Equation* (3)*, in the presence of static semi-honest adversaries.*

Proof. See Appendix E.3.

4.3 Performance evaluation of protocols

Both of the adaptations can be instantiated with an MHE or AHE scheme. This allows for greater degrees of freedom for implementations which makes it easier for the designs to be realised in practical environments. In terms of operations required for our PSI protocol, P_1 must compute B encryptions as before and then a maximum of $2m$ decryptions. P_2 must compute km exponentiations to satisfy the k multiplications needed for each element $y_j \in S_2$ (we require slightly less than in the additive case though there is no effect on the complexities). Since k is essentially a constant, by optimal choice as shown in Equation (5), then the number of exponentiations is $O(n)$ for $n = m$ and is thus linear. The communication remains as $2m + B$ ciphertexts and is thus also $O(n)$ bits. These complexities coupled with an efficient cryptosystem like ElGamal place the scheme in line with currently practical options for PSI such as [10, 11] which use public-key based operations. The design provides better asymptotic performance than the schemes of [16, 19] that require quadratic computation even after hashing optimisations have been leveraged. Since we primarily make use of public-key operations here it is clear that our protocol will not run in as practical parameters and timings as the schemes provided in [14, 30, 31]

For the PSI/PSU-CA construction we only need a total of m decryptions now (since we only send over m ciphertexts), while requiring the same amount of encryptions and homomorphic operations as our PSI protocol, therefore the computational complexity of the protocol is $O(n)$ for $n = m$ and the communication is similarly $O(n)$ bits. Consequently our design demonstrates favourable complexities compared with the design of [15] in the two-party case and comparable with those of [9] (see Table 2).

	Communication	Computation
De Cristofaro et al. [9]	$O(n)$	$O(n)$
Egert et al. [15]	$O(B)$	$O(B)$
$\pi^{\text{EBF}}_{\lvert\cap\rvert}/\pi^{\text{EBF}}_{\lvert\cup\rvert}$	$\boldsymbol{O(n)}$	$\boldsymbol{O(n)}$

Table 2. Comparison of our PSI/PSU-CA protocols with [9, 15]. Computation refers to number of exponentiations.

Both of these designs, along with our PSU protocol, give favourable results in the form of a library of easily implementable protocols with instantiations differing only in the message P_2 sends followed by the simple processing that is required by P_1. It is of great importance that we make designs easy to implement as we can undermine security by trying to construct a protocol that is very complex as implementers may take shortcuts to make it simpler. Furthermore, all the protocols can be proven secure in the semi-honest model in the standard model, this guarantees users of the protocol security under realistic constraints. We show how to guarantee malicious security in Appendix G.

Our work is the first to design multiple PSOs, all with linear complexities, and so it is worth comparing our work with that of Kissner et al. in [26] who were the first to demonstrate ways of computing different set operations from a singular underlying concept. Table 3 highlights how our protocols outperform these protocols considerably in an asymptotic sense.

		Communication	Computation
[26]	**PSI**	$O(cNn \log \lvert E \rvert)$	$O(n^2)$
	PSU	$O(N^2 n \log \lvert E \rvert)$	$O(n^2)$
	PSI/PSU-CA	$O(N^2 n \log \lvert E \rvert)$	$O(n^2)$
π^{EBF}	**PSI**	$2n + B$	$O(n)$
	PSU	$2n + B$	$O(n)$
	PSI/PSU-CA	$n + B$	$O(n)$

Table 3. Comparison of our complexities with the protocols of [26]. Computation refers to number of exponentiations.

5 Conclusion

In this paper we have devised a new method of computing the main private set operations with linear complexities. Our PSU protocol is the first construction that demonstrates both linear computation and communication, while our adapted PSI and PSI/PSU-CA protocols have complexities that are comparable with current practical designs. Our method appears to allow the computation of more complex set operations with relatively simple changes in the computation and without compromising security. In addition, it is possible to realise malicious security for our protocols using existing homomorphic signature schemes and while keeping the linear overheads. This provides a new interesting dimension to the efficiency of PSOs in the malicious setting, since practical, homomorphic signature schemes will have immediate implications for the efficiency of our protocols.

Our work contains therefore the first tool for computing the main set operations, with linear complexities, and with the possibility of extending to compute other more complex functionalities. Interesting future work would concretely establish the efficiency of our protocols via experimental work. This will result in the open-source implementation of our toolkit that will allow developers to use these constructions as part of applications that require multiple efficient PSO functionality.

References

[1] Steven M. Bellovin and William R. Cheswick. "Privacy-Enhanced Searches Using Encrypted Bloom Filters". In: *IACR Cryptology ePrint Archive* 2004 (2004), p. 22.

[2] Marina Blanton and Everaldo Aguiar. "Private and oblivious set and multiset operations". In: *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*. 2012, pp. 40–41.

[3] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (1970), pp. 422–426.

[4] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts". In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. 2005, pp. 325–341.

[5] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. "Public Key Encryption That Allows PIR Queries". In: *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*. Ed. by Alfred Menezes. Vol. 4622. Lecture Notes in Computer Science. Springer, 2007, pp. 50–67.

[6] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. "Signing a Linear Subspace: Signature Schemes for Network Coding". In: *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*. 2009, pp. 68–87.

[7] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang. "On the false-positive rate of Bloom filters". In: *Inf. Process. Lett.* 108.4 (2008), pp. 210–213.

[8] Justin Brickell and Vitaly Shmatikov. "Privacy-Preserving Graph Algorithms in the Semi-honest Model". In: *ASIACRYPT*. Vol. 3788. Lecture Notes in Computer Science. Springer, 2005, pp. 236–252.

[9] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. "Fast and Private Computation of Cardinality of Set Intersection and Union". In: *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*. 2012, pp. 218–231.

[10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. "Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model". In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*. 2010, pp. 213–231.

[11] Emiliano De Cristofaro and Gene Tsudik. "Practical Private Set Intersection Protocols with Linear Complexity". In: *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers*. 2010, pp. 143–159.

[12] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. "Efficient robust private set intersection". In: *IJACT* 2.4 (2012), pp. 289–303.

[13] Sumit Kumar Debnath and Ratna Dutta. "Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter". In: *Information Security - 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings*. 2015, pp. 209–226.

[14] Changyu Dong, Liqun Chen, and Zikai Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. 2013, pp. 789–800.

[15] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. "Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters". In: *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*. 2015, pp. 413–430.

[16] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. "Efficient Private Matching and Set Intersection". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. 2004, pp. 1–19.

[17] Keith B. Frikken. "Privacy-Preserving Set Union". In: *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*. 2007, pp. 237–252.

[18] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. "Leveled Fully Homomorphic Signatures from Standard Lattices". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 2015, pp. 469–477.

[19] Carmit Hazay and Kobbi Nissim. "Efficient Set Operations in the Presence of Malicious Adversaries". In: *J. Cryptology* 25.3 (2012), pp. 383–433.

[20] Farhad Hormozdiari, Jong Wha J. Joo, Akshay Wadia, Feng Guan, Rafail Ostrovsky, Amit Sahai, and Eleazar Eskin. "Privacy preserving protocol for detecting genetic relatives using rare variants". In: *Bioinformatics* 30.12 (2014), pp. 204–211. DOI: 10.1093/bioinformatics/btu294. URL: http://dx.doi.org/10.1093/bioinformatics/btu294.

[21] Yan Huang, David Evans, and Jonathan Katz. "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?" In: *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. 2012.

[22] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. "Extending Oblivious Transfers Efficiently". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. 2003, pp. 145–161.

[23] Florian Kerschbaum. "Outsourced private set intersection using homomorphic encryption". In: *7th ACM Symposium on Information, Compuer and*

Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012. 2012, pp. 85–86.

[24] Florian Kerschbaum. "Public-Key Encrypted Bloom Filters with Applications to Supply Chain Integrity". In: *Data and Applications Security and Privacy XXV - 25th Annual IFIP WG 11.3 Conference, DBSec 2011, Richmond, VA, USA, July 11-13, 2011. Proceedings*. Vol. 6818. Lecture Notes in Computer Science. Springer, 2011, pp. 60–75.

[25] M. H. R. Khouzani, Viet Pham, and Carlos Cid. "Strategic Discovery and Sharing of Vulnerabilities in Competitive Environments". In: *Decision and Game Theory for Security - 5th International Conference, GameSec 2014, Los Angeles, CA, USA, November 6-7, 2014. Proceedings*. 2014, pp. 59–78.

[26] Lea Kissner and Dawn Xiaodong Song. "Privacy-Preserving Set Operations". In: *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*. 2005, pp. 241–257.

[27] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. "BotGrep: Finding P2P Bots with Structured Graph Analysis". In: *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. 2010, pp. 95–110.

[28] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. "Location Privacy via Private Proximity Testing". In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011.

[29] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. 1999, pp. 223–238.

[30] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Faster Private Set Intersection Based on OT Extension". In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. 2014, pp. 797–812.

[31] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. "Phasing: Private Set Intersection using Permutation-based Hashing". In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 634.

[32] Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. "Constant-Round Multi-party Private Set Union Using Reversed Laurent Series". In: *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*. 2012, pp. 398–412.

[33] Giulia Traverso, Denise Demirel, and Johannes Buchmann. *Homomorphic Signature Schemes - A survey*. Cryptology ePrint Archive, Report 2015/653. `http://eprint.iacr.org/`. 2015.

A Diagrams from the main body

Security level	**80-bit**					**128-bit**				
Set sizes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
De Cristofaro et al. [11]	0.5	2.0	7.9	31.3	124.9	7.7	31.0	124.3	497.2	1982.1
Huang et al.[21]*	1.2	5.1	21.2	100.3	462.7	1.9	7.8	36.5	168.9	762.4
Dong et al.[14]*	0.15	0.5	2.0	8.1	34.3	0.27	1.0	4.1	16.7	67.6
Pinkas et al.[30]	0.13	0.2	0.8	3.3	13.5	0.26	0.3	0.9	3.7	13.8

Table 4. Runtimes (seconds) taken from [30]

Security level	**80-bit**					**128-bit**				
Set sizes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}
De Cristofaro et al. [11]	0.3	1.1	4.3	17.3	69.0	0.8	3.1	12.5	50.0	200.0
Huang et al.[21]*	18.8	90.0	420.0	1920.0	8640.0	30.0	144.0	672.0	3072.0	13824.0
Dong et al.[14]*	1.1	4.5	18.1	72.6	290.4	2.9	11.6	46.2	184.9	739.7
Pinkas et al.[30]	0.2	0.8	3.3	13.4	54.3	0.3	1.2	4.8	19.4	78.3

* With optimisations from [30]

Table 5. Communication costs (MB) taken from [30]

	Communication	Computation	Multi-party?
Kissner et al.[26]	$O(N^2 n \log \lvert E \rvert)$	$O(n^2)$	Y
Brickell et al.[8]	$O((n+m) \log \lvert E \rvert)$	$O((n+m) \log \lvert E \rvert)$	N
Frikken [17]	$O(n)$	$O(n \log \log n)$	N
Blanton et al.[2]	$O(N^3 n \log(Nn))$	$O(N^3 n \log(Nn))$	Y

Table 6. Complexities for previous PSU protocols.

	Communication	Computation
De Cristofaro et al.[9]	$O(n)$	$O(n)$
Egert et al.[15]	$O(B)$	$O(B)$

Table 7. Complexities for the protocol of [9] and the two-party variant of [15].

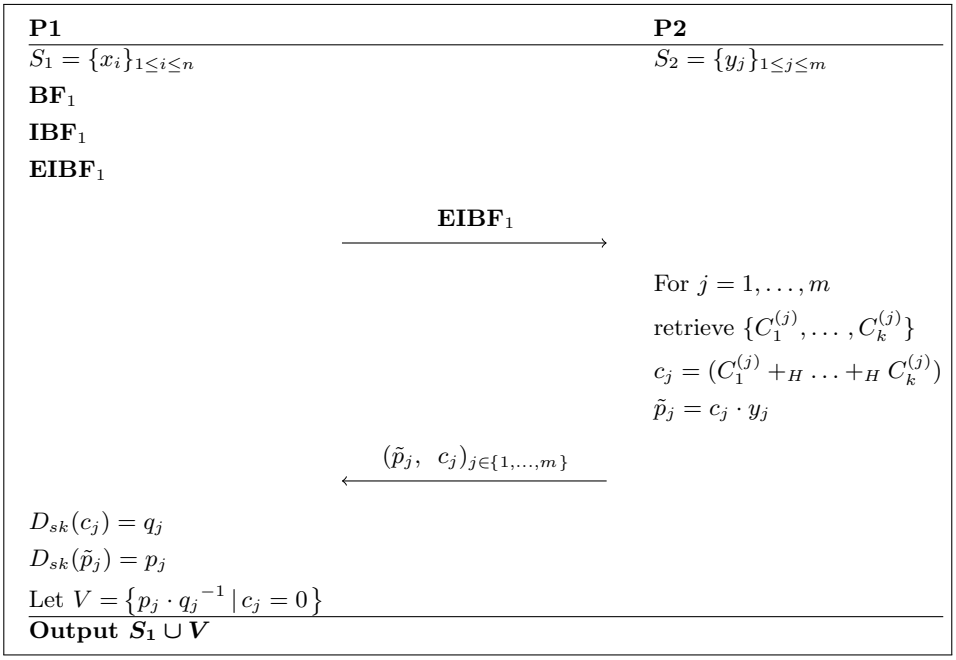

Fig. 1. An overview of our $\pi_{\cup}^{\text{EBF}}$ protocol that uses encrypted, inverted Bloom filters

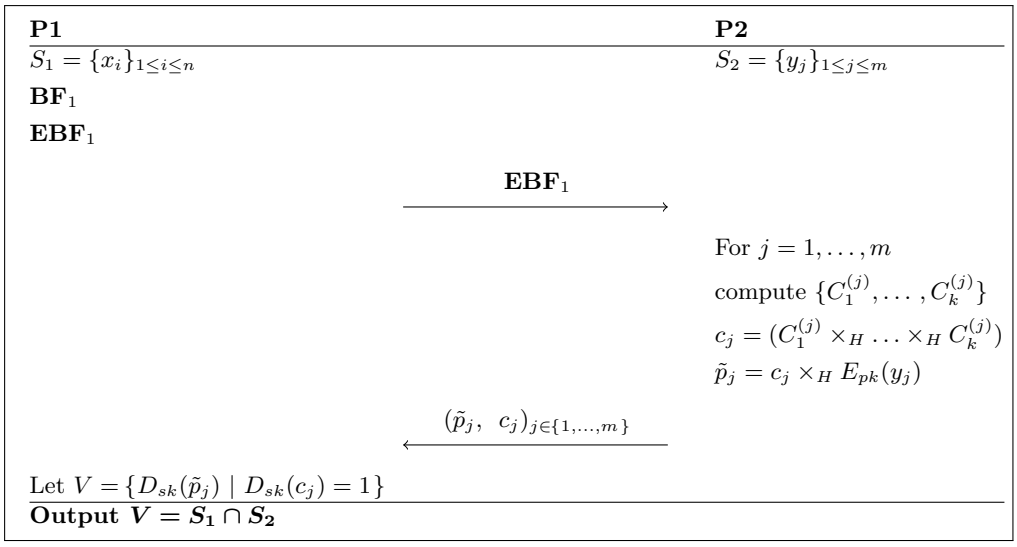

Fig. 2. A protocol that securely realises $\mathcal{F}_{\cap}$ in a similar way to $\pi_{\cup}^{\text{EBF}}$.

B Bloom filters

Bloom filters were first introduced by Bloom in [3] as a lightweight data structure that allows for the representation data sets and checking of inclusion using only hash function evaluations. A Bloom filter is initially represented by a string of B bits that are all initialised to 0. There are k hash functions $h_l : \{0,1\}^\lambda \mapsto \{1,\ldots,B\}$ for $l \in \{1,\ldots,k\}$ published alongside the Bloom filter. We then represent set elements $x \in X$ in the Bloom filter by evaluating $h_1(x),\ldots,h_k(x)$ and changing each index that these hash functions point to from 0 to 1. If a value has already been changed to 1 then it is left alone.

The resulting Bloom filter can then be checked against to see if different elements lie in the set by evaluating the k hash functions and checking if all the positions are set to 1. The following definition establishes some notation that we shall carry forward throughout.

Definition 6. *We say that an element, e, is* represented *in the Bloom filter, **BF**, if we have that*

$$\boldsymbol{BF}[h_i(e)] = 1, \ \ \forall i \in \{1,\ldots,k\}$$

*where $\{h_1,\ldots,h_k\}$ are the hash functions used in conjunction with **BF**. We say that the set S is* represented by ***BF*** *if every element $e \in S$ is represented in **BF**.*

We can then query whether an element y is contained in S by checking if y is represented in the corresponding Bloom filter **BF**. This construction clearly lends itself to fast set intersection computation since only hash functions are used to evaluate whether an element is part of the underlying set.

One constraint on Bloom filters is that they can lead to false positives when checking membership, i.e. an element $y \notin X$ may appear to be in X after checking all the hash outputs if all the values have been set to 1 beforehand. However, as shown in [14], if $p = 1-(1-1/B)^{kn}$ is the probability that a particular bit in the Bloom filter is set to 1, then the upper bound of the false-positive probability is given by

$$\epsilon = p^k \times \left(1 + O\left(\frac{k}{p}\sqrt{\frac{ln\, B - k\, ln\, p}{B}}\right)\right)$$

which is negligible in k, the number of hash functions. In practice one will select the values of k and B when building a Bloom filter for a set of size n such that ϵ is capped at a specific low value (e.g. 2^{-100}). In [14] it is claimed that performance optimality is achieved when

$$k = \frac{B}{n}\ln 2 \tag{5}$$

$$B \geq n \log_2 e \cdot \log_2 1/\epsilon \tag{6}$$

where e is the base of the natural logarithm. By minimising B we get the optimal value of k to be

$$k = \log_2 1/\epsilon. \tag{7}$$

We will assume (as in [14]) that these parameters are always chosen optimally in this way. The proofs that these values are optimal can be found in [7].

C Partially homomorphic encryption

To preserve secrecy of the client's set we require that the Bloom filter be encrypted with a partially homomorphic encryption (PHE) scheme. This allows the server to compute operations over the client's data without compromising privacy of the client's set. We use the following to informally define a homomorphic encryption scheme

Let $(\mathcal{P}, \mathcal{C}, \mathcal{K}, E, D)$ be a public key cryptosystem with public key pk and secret key sk where $\tilde{x} = E_{pk}(x)$ and $\tilde{y} = E_{pk}(y)$ are ciphertexts of the scheme for plaintexts x and y. We say that the encryption scheme is *partially homomorphic* if we have one of the following properties:

- There is a homomorphic addition operation, $+_H$, over $\tilde{x}$ and $\tilde{y}$ such that $D_{sk}(\tilde{x} +_H \tilde{y}) = x + y$. A scheme satisfying this is known as an *additively homomorphic encryption* (AHE) scheme.
- There is a homomorphic multiplication operation, $\times_H$, over $\tilde{x}$ and $\tilde{y}$ such that $D_{sk}(\tilde{x} \times_H \tilde{y}) = x \times y$. A scheme satisfying this is known as a *multiplicatively homomorphic encryption* (MHE) scheme.

One of the most popular AHE schemes is the one devised by Paillier in [29], we use such a scheme for our PSU and PSI/PSU-CA protocols. A popular MHE scheme is ElGamal, we could use this for our PSI protocol. Both schemes incur one exponentiation per homomorphic operation which we note in the analysis of our protocols.

D Paillier cryptosystem

Paillier's encryption scheme is introduced in [29], the scheme is a semantically secure public key encryption scheme and allows for additively homomorphic operations over the ciphertexts. It operates in the following way:

- **Key generation**: Sample two large random primes p, q and set $N = pq$. Let $\lambda = \mathsf{lcm}(p-1, q-1)$ i.e. λ is the Carmichael number of N. Sample $g \in \mathbb{Z}^*_{N^2}$ and ensure that N divides the order of g by checking the existence of $\mu = L(g^\lambda \mod N^2)^{-1} \mod N$ where $L(x) = \frac{x-1}{N}$. The public key is $pk = (N, g)$ and the secret key is $sk = (\lambda, \mu)$.
- **Encryption**: Let $m \in \mathbb{Z}_N$ be a plaintext. Randomly sample $r \in \mathbb{Z}^*_N$ and compute $C = g^m \cdot r^N \mod N^2$.
- **Decryption**: $m = L(C^\lambda \mod N^2) \cdot \mu \mod N$
- **Homomorphic evaluation**: Suppose that we have ciphertexts c_1 and c_2 from a Paillier encryption of two plaintexts m_1 and m_2, then we have the following property:

$$D_{sk}(c_1 \times c_2) = m_1 + m_2. \tag{8}$$

  We also have the possibility of multiplying the underlying plaintext by a scalar $k \in \mathbb{Z}$, i.e.

$$D_{sk}({c_1}^k) = k \cdot m_1. \tag{9}$$

We require both of these properties in our PSU protocol of Section 3. To maintain encryption agnosticism in our protocol we make use of property (8) while denoting the additive homomorphic operation as $+_H$ and thus stating that $D_{sk}(c_1 +_H c_2) = m_1 + m_2$. For property (9) we state that $D_{sk}(k \cdot c_1) = k \cdot m_1$.

E Proofs from main body

E.1 Proof of Theorem 1

We will show that the protocol is secure when P_2 is corrupted first due to the relative simplicity of the proof relative to the P_1 corruption case. Note that - similarly to the previous PSU protocols in [2, 17, 26] - our protocol reveals the cardinality of the intersection to P_1. Therefore, we define the input for player P_i to be $\mathsf{Inp}_i = (S_i, aux_i = |S_j|)$ for $i \in \{1, 2\}$ and $((j \in \{1, 2\}) \wedge (j \neq i))$.

P_2 corrupted It is fairly trivial to construct a simulator in this case since P_2 does not receive any output from the protocol. That being said, the simulator must construct a random, encrypted, inverted Bloom filter for the view of P_2 that is indistinguishable from the real Bloom filter that P_2 receives. Here we use the IND-CPA security of the encryption scheme to argue that it is impossible for P_2 to distinguish between the real Bloom filter and the simulated version.

P_1 corrupted The simulation here is significantly more nuanced since P_1 receives the output $S_1 \cup S_2$ from the protocol. Firstly the simulator can derive $|S_1 \cap S_2| = I$ from S_1 and $|S_2|$, which is present from the input of the party, by calculating $|(S_1 \cup S_2) \setminus S_1| = |S_2 \setminus S_1| = U$ and subsequently $|S_2| - |(S_2 \setminus S_1)| = I$. They then construct I encryptions, c_g, of 0 and U encryptions $c_j = C_1^{(j)} +_H \ldots +_H C_k^{(j)}$ computed as in the original protocol using the elements $y_j \in (S_1 \cup S_2) \setminus S_1$ constructed via the output and the input set. The simulator then sends $m = I + U$ messages in total where I messages are represented by $(E_{pk}(0), E_{pk}(0))$ and U messages are represented by $(c_j \cdot y_j, c_j)$ as in the real execution.

The view of P_1 in the real protocol is indistinguishable from the view of P_1 constructed by the simulator since they can still decrypt and learn the values of y_j that they should be able to, and the I encryptions of 0 in the simulation are indistinguishable from the values that shouldn't be learnt (from the intersection) in the real protocol by the IND-CPA security of the encryption scheme.

□

E.2 Proof of Theorem 2

The security when P_2 is corrupted follows from the same arguments as before due to the use of an IND-CPA encryption scheme for encrypting $\mathbf{BF}_1$. We do not compromise security by not inverting as the encryption renders $\mathbf{EIBF}_1$ and $\mathbf{EBF}_1$ indistinguishable by the same argument in the proof of Theorem 1.

For the corruption of P_1 we note that the security relies now on P_1 not being able to learn elements $y'_j \notin S_1 \cap S_2$ in order to realise $\mathcal{F}_\cap$ securely. This is trivially stopped since

$$D_{sk}(\tilde{p}_j) = \begin{cases} y_j \text{ if } c_j = E_{pk}(1) \\ 0 \text{ if } c_j = E_{pk}(0) \end{cases}$$

and by definition c_j is an encryption of 1 if $y_j \in S_1 \cap S_2$ and 0 otherwise except for negligible chance of failure.

□

E.3 Proof of Theorem 3

Security when P_2 is corrupted follows exactly from the previous cases. For security when P_1 is corrupted we just have to prove that the adversary cannot learn anything about the sets from the values c_j that they receive. Since c_j is the sum of encrypted inverted bloom filter entries then we know that all elements in the intersection will have $D_{sk}(c_j) = 0$ and so it is impossible to learn anything from these values. For elements not in the intersection we have $D_{sk}(c_j) = z_j$ for some $z_j \in \mathbb{N}$ as before, which does not reveal anything about the elements y_j that are represented in the Bloom filter.

□

F Homomorphic signature scheme

We give a general construction of a homomorphic signature scheme that allows our protocol to ensure security against malicious adversaries. For a specific scheme that meets our requirements see the work of Gorbunov et al. [18].

$\mathsf{SIGKEYGEN}(1^\lambda)$ Samples $ssk, spk \xleftarrow{\$} \mathcal{K}$ from a keyspace $\mathcal{K}$. The key ssk is a secret key that is used to authenticate messages, spk is a public key allowing for verification and homomorphic evaluation of signatures.

$\mathsf{AUTH}(ssk, m)$ Outputs a signature σ correctly authenticating the message m.

$\mathsf{EVAL}(spk, f, (\sigma_1, \ldots, \sigma_l))$ Outputs a signature σ^* authenticating the message $m = f(m_1, \ldots, m_l)$. Where σ_i is a valid signature on m_i for $1 \leq i \leq l$

$\mathsf{VRF}(spk, m^*, \sigma^*, f)$ Outputs 1 if σ^* is a valid signature on $m = f(m_1, \ldots, m_n)$ where the m_i's are messages with corresponding signatures σ_i for $1 \leq i \leq l$. It is also possible to verify direct signatures on elements where no evaluation has taken place. In these cases we do not specify a function f.

For a scheme to be appropriate for our protocol we require that it is correct and satisfies unforgeability requirements (see [33] for concrete definitions). We also need that the signing function f allows for computation of polynomial operations over the pieces of data that are input to it. An instantiation of Paillier's

encryption scheme in our protocol requires signed ciphertexts to be multiplied to invoke the additive homomorphic properties. Existing signature schemes allow for linear, polynomial or fully homomorphic operations. In addition we need that the signature scheme is context-hiding implying that the derived signatures σ^* reveal nothing about the underlying pieces of data $m_1, \ldots, m_l$ with signatures $\sigma_1, \ldots, \sigma_l$ (aside from what m^* reveals naturally). We require this in our protocol since the adversary may be able to piece together elements that they should not be able to learn if the signatures reveal which components are used in the derivation of each c_j.

The only scheme that currently satisfies these requirements is detailed in [18]. This scheme allows for leveled fully homomorphic operations (i.e. circuits of depth d that is specified before computation takes place) and security can be reduced to the hardness of the SIS problem on lattices. Since the number of homomorphic operations that we need is calculable based on the set sizes, this does not pose a problem for our protocol. An AHE scheme that didn't invoke each $+_H$ with a multiplication would allow us to use a linearly homomorphic scheme such as [6].

G Achieving malicious security

We discuss, somewhat informally, how to protect our PSU protocol against a malicious adversary, a formal proof of our malicious design is contained in Appendix H.[2] By definition a malicious adversary can interact with the protocol in any possible manner and so our construction needs to demonstrate that the adversary cannot deviate in a way that will leave the computation insecure. In fact, we construct a stronger variant, namely APSU, similar to the APSI designs covered in [10, 11] where the input sets are authenticated by a third party to also prevent the usage of badly formed inputs that could contravene the security of the corresponding input set. For instance, P_1 could encrypt the inverted Bloom filter that is simply initialised with all entries set to 1 allowing her to learn all the elements in S_2. Analogously, she could set entries in her Bloom filter to 0 before encrypting in the PSI computation resulting in her learning the full set S_2. It is interesting to note that these concerns are not considered in PSO protocols that only guarantee security against malicious adversaries without authentication.

Secondly, though P_2 learns no output, he can still influence the output of P_1 in some guaranteed ways. For example, P_2 can alter the value of c_j so that P_1 does not learn a particular element in the PSU computation (i.e. set $c_j = E_{pk}(0)$). This would allow him to hide particular elements if he didn't want P_1 to learn these and so there is quite strong motivation for doing this. We discuss below how to mitigate these situations in order to realise security for $\pi_{\cup}^{\text{EBF}}$ with respect to a malicious adversary. As stated previously, a proof of security can be found in Appendix H, where it should be noted that we require that the hash functions are now modelled as random oracles.

[2] The similarity of the designs means that it is simple to further derive maliciously secure PSI and PSI/PSU-CA.

G.1 Bloom filter manipulation

Work to address the first concern is prevalent in PSO literature. One of the ways suggested in [10, 11] is to have the set of the client authorised before computation takes place. This is done by a trusted certificate authority who 'signs' the set of P_1 before computation takes place so that P_2 can be sure of the legitimacy of the computation. Thus we refer to the construction as authenticated PSU (APSU).

We apply the technique by making the client send their inverted Bloom filter, $\mathbf{IBF}_1$, to a trusted signing authority (SA).[3] SA receives $\mathbf{IBF}_1$ and checks if it meets certain requirements on the size of the set represented and then encrypts each entry of $\mathbf{IBF}_1$ using pk and sends the resulting $\mathbf{EIBF}_1$ back to P_1. The main requirement is that, since we have n elements and k hash functions, the number of ones in $\mathbf{IBF}_1$ cannot become higher than kn. With optimal choice of parameters $1.44kn = B$ so SA can just check that this is the case and that the number of zeros is above some pre-determined threshold (agreed with P_2) before computing and authorising $\mathbf{EIBF}_1$. The authorisation consists of SA signing $\mathbf{EIBF}_1$ after it has performed encryption. P_2 will then verify this signature when it receives $\mathbf{EIBF}_1$ from P_1 and will abort the protocol if it does not verify successfully.

G.2 Output manipulation

To protect against the second concern we need to prevent the ability of a corrupted P_2 from having an effect on the output of the protocol beyond their actual input. This means that we need to stop P_2 from being able to create their own versions of c_j via exploitation of public-key encryption to hide elements of their choice. We choose to counter this by introducing a homomorphic signature scheme that satisfies standard unforgeability requirements.[4] Such a signature scheme has the property that; for messages $m_1, \ldots, m_l$ with signatures $\sigma_1, \ldots, \sigma_l$ then we can compute a valid signature σ for the message $m = f(m_1, \ldots, m_l)$ where f is a polynomial. A public verify algorithm VRF can check the validity of the signature with respect to f.

For our malicious protocol, SA will also sign each individual ciphertext (computing $\sigma[l]$ for each ciphertext $C[l]$) in $\mathbf{EIBF}_1$ and return these to P_1. When P_1 sends $\mathbf{EIBF}_1$ to P_2 she will also send the signatures σ (signing the whole Bloom filter) and $\sigma[1], \ldots, \sigma[l]$ to P_2. Now, when P_2 computes c_j he will also have to homomorphically evaluate new signatures σ_j and θ_j that sign c_j and $\tilde{p}_j$ respectively. P_2 will send these signatures back with the computation of each message $(\tilde{p}_j, c_j)$ to P_1 who will then be able to verify that both ciphertexts have been computed correctly and thus stopping P_2 from being able to alter the output maliciously. It is important to ensure that the signature σ on the whole Bloom filter is generated as a standard digital signature. Otherwise if this was also homomorphic then $\mathbf{BF}_1$ would retain malleability since P_1 could alter values in the

[3] P_1 sends $\mathbf{IBF}_1$ since we are referring to the PSU functionality.
[4] See [33] for unforgeability definitions.

structure and generate valid signatures using the homomorphic properties. The signature σ is therefore integral to ensuring that P_1 cannot change $\mathbf{BF}_1$ before sending it to P_2. Signature schemes that are compatible with our requirements, i.e. allow additions over the signed data, include [6, 18].[5]

G.3 Efficiency

Communication is increased due to the extra signatures that have to be sent between the parties. P_1 sends $B + 1$ extra signatures on top of the encrypted Bloom filter that they send to P_2, P_2 sends over a total of $2m$ extra signatures to P_1 for verifying their computation.

Computation complexities are now directly affected by the need to sign and verify data during the protocol. If we were to use the scheme of [18] then the signatures have length that is polynomial in the depth, d, of circuits that we use to evaluate the homomorphic operations. This is favourable as this value will depend only on the number of multiplications that we need per element in S_2 which is k rather than the size of the data sets or the size of the circuit required. The verification procedure for this scheme also allows for the pre-computation of the same homomorphic evaluation function f if we are verifying over it multiple times. This perfectly fits our scenario where f is fixed, so P_1 can perform this pre-computation and only performs work proportional to the size of f once. P_1 is then able to verify arbitrary amounts of signatures based on this function f, making verification of each σ_j and θ_j much more efficient.

These complexities mean that the malicious scheme is in line with previous solutions such as [10, 14] that also have linear complexities. We also have the first maliciously secure APSU protocol to have linear complexities. We view this construction as an important example of new techniques that can be used to prove protocols secure in the malicious model rather than using the ZK proofs that [17, 19, 26] do. Further advances in the efficiency of homomorphic signature schemes will directly improve the efficiency of our protocol.

H Proof of security in malicious model

In this section we first give our updated PSU construction before proving that this protocol retains security and correctness in the presence of malicious adversaries. Our construction relies on a trusted certificate authority (SA) who carries out the encryption and signing of $\mathbf{IBF}_1$. The signatures σ are generated by an unforgeable signature scheme, while $(\sigma_1, \ldots, \sigma_B)$ are generated by a fully homomorphic signature scheme satisfying unforgeability and maintaining context-hiding privacy such as [18]. Unlike our previous proof the hash function evaluations are modelled as calls to non-programmable random oracles so that the simulator can extract the queries made to the oracles during the execution.

[5] We also require that the schemes also satisfy a context-hiding requirement in order to ensure the privacy of the ciphertexts that are used to construct are kept hidden.

We can avoid this requirement for this model by making P_1 send S_1 along with $\mathbf{IBF}_1$ to SA.

Below we provide a description of the protocol with security against malicious adversaries along with a diagrammatic overview in Figure 3. In the description we reference functions f and g below when we evaluate new signatures, the function f evaluates $f(x_1, \ldots, x_k) = x_1 +_H \ldots +_H x_k$ and g evaluates $g(x, y) = y \cdot x$.

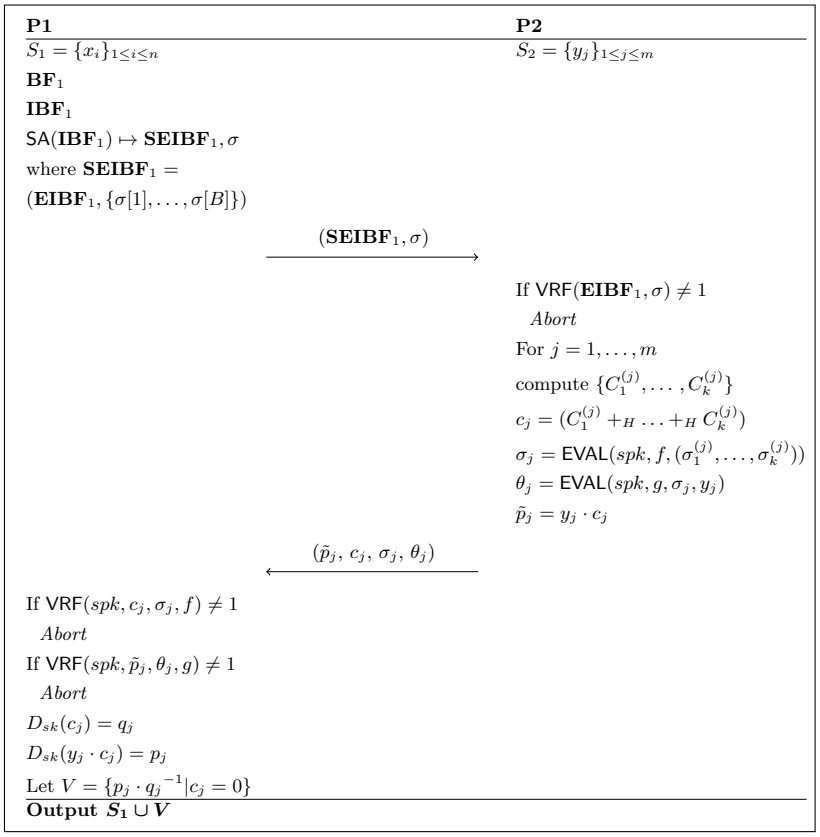

Fig. 3. An overview of our maliciously secure PSU protocol

H.1 Protocol steps

Inputs - P_1: [(pk, sk), S_1, $|S_2|$], P_2: [pk, S_2, $|S_1|$]

1. P_1 calculates $\mathbf{BF}_1$ using $h_1, \ldots, h_k$ and their set S_1 and then inverts each entry to retrieve $\mathbf{IBF}_1$.
2. P_1 sends $\mathbf{IBF}_1$ to SA and receives back $\mathbf{EIBF}_1$ and a standard signature σ computed over $\mathbf{EIBF}_1$, and homomorphic signatures $\sigma[l]$ for each $\mathbf{EIBF}_1[l] = C[l]$.

3. P_1 sends $(\mathbf{EIBF}_1, \{\sigma[1], \ldots, \sigma[B]\}, \sigma)$ to P_2.
4. P_2 checks that $\mathsf{VRF}(spk, \mathbf{EIBF}_1, \sigma) = 1$, if this fails then P_2 aborts the protocol.
5. He then evaluates each element $y_j \in S_2$ using the k hash functions and receives $\{C_1^{(j)}, \ldots, C_k^{(j)}\}$ where $C_d^{(j)} = \mathbf{EIBF}_1[h_d(y_j)]$ for $j \in \{1, \ldots, m\}$.
6. He then computes $c_j = f(C_1^{(j)}, \ldots, C_k^{(j)})$ and $\sigma_j = \mathsf{EVAL}(spk, f, (\sigma_1^{(j)}, \ldots, \sigma_k^{(j)}))$ for each j where $\sigma_d^{(j)}$ authenticates $C_d^{(j)}$.
7. P_2 computes $\tilde{p}_j = g(c_j, y_j)$ and $\theta_j = \mathsf{EVAL}(spk, g, \sigma_j, y_j)$ and sends $(\tilde{p}_j, c_j, \sigma_j, \theta_j)$ to P_1.
8. P_1 receives $(\tilde{p}_j, c_j, \sigma_j, \theta_j)$ and checks $\mathsf{VRF}(spk, c_j, \sigma_j, f) = 1$ and that $\mathsf{VRF}(spk, \tilde{p}_j, \theta_j, g) = 1$. If either of these checks fail then P_1 aborts the protocol.
9. P_1 reveals the value of c_j by computing $D_{sk}(c_j) = q_j$. If $q_j = 0$ then $D_{sk}(y_j \cdot c_j) = 0$ so she does not decrypt. If $q_j = z_j$ then $D_{sk}(y_j \cdot c_j) = y_j \cdot z_j = p_j$, where $1 \leq z_j \leq k$.
10. P_1 computes ${z_j}^{-1}$ for $q_j \neq 0$ and calculates $p_j \times {z_j}^{-1} = y_j$. She adds each y_j to the set V and outputs $S_1 \cup V$.

H.2 Correctness/security analysis

The correctness of our protocol trivially follows from the fact that our semi-honest PSU protocol is also correct, since we do not change the computation that is performed aside from verifying and computing signatures. For proving our PSU protocol secure in the presence of malicious adversaries we have to alter the security model that we use. Specifically we need to show that definition 3 holds for our protocol and thus that a simulator with access to an ideal functionality can simulate the real world with access to the ideal functionality.

Our security analysis starts by showing the privacy of P_2 is maintained when P_1 is corrupted. In both proofs we assume that the simulators have access to the values sent to SA.

Theorem 4. *The protocol above, when instantiated with an IND-CPA secure encryption scheme and unforgeable signature schemes, securely realises the ideal functionality in Equation* (1) *in the presence of malicious adversaries.*

Proof. When P_i is corrupted we denote the simulator by Sim_i and the adversary by $\mathcal{A}_i$.

P_1 corrupted. Sim_1 can learn the set S_1' of all elements that are queried to the random oracles $h_1, \ldots, h_k$ by $\mathcal{A}_1$ using the extractable property, where the actual set used to compute $\mathbf{BF}_1$ is $S_1 \subseteq S_1'$. Sim_1 constructs the actual set S_1 by querying the elements of S_1' on the inverted Bloom filter $\mathbf{IBF}_1$ that is sent to SA. Sim_1 now inverts, encrypts and signs both the full Bloom filter (with an unforgeable digital signature) and each individual ciphertext (with a unforgeable homomorphic signature) and sends everything back to P_1. Sim_1 can now send S_1 to the ideal functionality and receive back $S_1 \cup S_2$ and can use this to compute the messages that P_2 would send back to P_1 so that P_1 learns the union.

This is indistinguishable from the original game since $\mathcal{A}_1$ is unable to alter the contents of the Bloom filter once it has been sent to SA by the unforgeability of the signature scheme and cannot distinguish the messages they receive by the semantic security of the encryption scheme.

P_2 corrupted. As in Section 3 this case is slightly easier since P_2 receives no output in the protocol. Our aim here is to make sure that $\mathcal{A}_2$ cannot alter messages in a way that would mean that the output is no longer correct. Since each ciphertext is signed we know that c_j for each element must be constructed in a certain way, Sim_2 constructs an admissible encrypted Bloom filter $\mathbf{EIBF}_1$ that consists of all ones and sends this along with the signatures to $\mathcal{A}_2$. When it receives the result back it checks the homomorphic signatures for c_j and $c_j \cdot y_j$ verify correctly. Sim_2 then proceeds in the normal way and decrypts the values and learns the entire set S_2 and submits this to the ideal functionality to learn $S_1 \cup S_2$. The simulation now finishes and outputs nothing to P_2. this is clearly indistinguishable from real for $\mathcal{A}_2$ since the semantic security of encryption means that $\mathcal{A}_2$ cannot distinguish the messages they receive and the unforgeability of the signature scheme means that they cannot trick the simulator into outputting an incorrect union.

□