Asynchronous Delegated Private Set Intersection with Hiding of Intersection Size

Wyatt Howe Andrei Lapets Frederick Jansen Tanner Braun Ben Getchell

Nth Party, Ltd.
Boston, MA
{wyatt, andrei, frederick, tanner, ben}@nthparty.com

Abstract

Integrating private set intersection (PSI) protocols within real-world data workflows, software applications, or web services can be challenging. This can occur because data contributors and result recipients do not have the technical expertise, information technology infrastructure, or other resources to participate throughout the execution of a protocol and/or to incur all the communication costs associated with participation. Furthermore, contemporary workflows, applications, and services are often designed around RESTful APIs that might not require contributors or recipients to remain online or to maintain state. Asynchronous delegated PSI protocol variants can better match the expectations of software engineers by (1) allowing data contributors to contribute their inputs and then to depart permanently, and (2) allowing result recipients to request their result only once they are ready to do so. However, such protocols usually accomplish this by introducing an additional party that learns some information about the size of the intersection. This work presents an asynchronous delegated PSI protocol variant that does not reveal the intersection size to the additional party. It is shown that such a protocol can have, on average, linear time and space complexity.

1 Motivation and Objectives

Many contemporary data workflows, software applications, and web services possess or rely on architectures in which components make asynchronous application programming interface (API) calls to one another. For example, a web service or mobile application developer might add a feature to their solution by making asynchronous API calls to a web service being operated by a third-party vendor. With entire ecosystems built around such conventions, significant obstacles exist to the integration of secure multi-party computation (MPC) protocols such as those for computing a private set intersection (PSI), which often require participating parties to remain online (or at least to maintain state) throughout significant portions of a protocol's execution time.

One way to facilitate the integration of PSI protocols within contemporary solutions is by developing PSI protocol variants that allow each data contributor and result recipient to participate in the protocol via just one or two asynchronous API requests. One way such a protocol can be realized is via the introduction of an additional party that does maintain some small amount of state (typically consisting of cryptographic keys and not related in size to the amount of data being processed). However, such protocols may allow the additional party to learn some information about the result (such as the intersection size). This work addresses this specific potential drawback of the approach by demonstrating that information about the intersection size need not be revealed to the additional party.

We consider a scenario in which two or more data contributors want to contribute their private data sets (*i.e.*, ordered collections of rows in which each row contains one value) to a workflow that allows a recipient to compute the intersection of those contributed data sets *at some later point in time.* We assume that the data contributors will only agree to participate if they are assured that only the portion of their data that is found within the overall intersection will be revealed to the recipient (and that none of their data is revealed to any other parties).

In order for the contributors to have an experience that is familiar within the contemporary API-oriented ecosystem, they should be able to (1) encrypt their data asynchronously via an API call, (2) submit that data asynchronously to the recipient, and (3) disengage permanently from the protocol. Similarly, the recipient should be able to (1) store the contributed encrypted data sets until it is time to run the computation and (2) compute the intersection of the encrypted data sets via at most one API call. These goals are accomplished via the introduction of an additional party (or *aide*) with which contributors and recipients can perform one request-and-response interaction.

This work focuses on the feasibility of designing a PSI protocol that delivers the experiences described above while also satisfying other common constraints associated with data privacy and ease of integration. The criteria that are of interest in this work are enumerated and described below.

- **Privacy:** Data contributors learn nothing except their own data (*i.e.*, they learn nothing new), and the designated recipient learns only the intersection (or only its size, depending on the purpose of the protocol). Any additional parties that may be involved do not learn the intersection or its size.

- **Asynchronicity:** Each data contributor can encrypt their data at any time, and doing so requires only a single request-and-response interaction (from that contributor's perspective). The encrypted data can be sent to the recipient at any time before the result has been computed. There are no other restrictions on the relative order or timing of the contributors' communications. In particular, contributors take no further action after sending their encrypted data.

- **Scalability:** The above two criteria are satisfied regardless of the number of data contributors or the sizes of their contributed data sets. Additionally, the cost of executing the protocol is linearly proportional to the amount of data processed (*i.e.*, a constant amount of work per row).

2 Related Work

The idea of introducing an additional party into an MPC protocol (to facilitate its execution) *that is not the recipient of the result* has been explored extensively. The server in a *delegated PSI* protocol need not be the recipient; some existing work [7] has the server send an encrypted result back to each contributor (rather than keeping or publicizing it). In other work [1, 5], the server itself is the recipient, or learns the result itself before forwarding it to a separate recipient. The protocol presented in Section 4 generalizes naturally to the case where the server is not the recipient and instead does more work than the separate recipient would have to do (and this separate recipient, if it were to exist, would only call the server's REST API, and never the additional party's).

Some recent protocols featuring a delegated computation [1] allow contributors to add to their originally contributed data; these are referred to as *updatable* or *repeated delegation* (vs. *one-off* *delegation*) protocols. We focus on the one-off case (whether our work also generalizes naturally to the repeated case in which old data can be reused is a question for future work). These three other works (and similar others of which we are aware) require multiple rounds of computation between compute servers, while we restrict communication over the network to at most a few messages per party. This allows our protocol to fit the constraints of a wider range of use cases. It may also increase confidence thanks to a comparably simpler security analysis. Furthermore, the security of the most efficient delegated PSI variants proposed in recent related work [1, 2, 5, 7] requires semi-honest participants, while we show that our protocol can be securely run by malicious parties.

More broadly, this work is situated within – and informed by – efforts to facilitate the adoption of MPC capabilities within contemporary software applications and web services by improving their usability [9]. Asynchronous delegated PSI protocols can be leveraged by vendors to operate MPC-as-a-service infrastructures [8] that allow software engineers to introduce PSI workflows into applications without learning to use MPC libraries (as opposed to directly implementing a solution using a web-compatible MPC library [4]).

3 Conventions and Background

In this section, we introduce the conventions and notation used throughout this work. We illustrate these by presenting a definition of a simple protocol for computing the size of the intersection of a collection of contributed data sets.

3.1 Formal Definitions and Notation

We assume that data contributors possess *data sets*, which are ordered deduplicated collections of elliptic curve points (*i.e.*, they could formally be represented as tuples of points in which each component corresponds to a row in the data set).[1] Data sets are denoted using uppercase variables such as X, and the two data sets of additive secret shares for a given data set X are denoted as X_1 and X_2 (such that the ith row of X is equivalent to the sum of the ith row of X_1 and the ith row of X_2. The reconstruction of all rows of a secret-shared data set is denoted using $X_1 + X_2$.

A permutation of a data set X (*i.e.*, a permutation of the rows of X) is denoted using π (where the specific choice of π within an execution is pseudorandom). The result of applying a permutation π to a data set X is denoted using $\pi(X)$.

Data sets consisting of masks (*i.e.*, elliptic curve scalars) are denoted using uppercase variables such as M, and masks are denoted using lowercase variables such as k or m. Given a scalar mask k, the superscript notation X^k refers to the data set obtained by multiplying every point in X by the scalar k. Given a data set of scalar masks M such that both X and M have the same number of rows, the notation X^M refers to the data set obtained by multiplying the ith row in X by the scalar found in the ith row of M. The data set of inverses of a data set of masks M is denoted M^{-1}.

3.2 Example: Protocol for Intersection Size

As an illustration of the conventions used in this work – and as a starting point for the protocol presented in Section 4 – we present a simple protocol for computing the intersection size for a collection of two or more contributed data sets.

Each contributor (of n distinct contributors) splits their data set (*e.g.*, X) into two data sets of secret shares (*e.g.*, X_1 and X_2). Each contributor also generates a data set of fresh masks (*e.g.*, M_x)

[1]In practice, the raw data value that is found in each row is hashed deterministically to an elliptic curve point.

and masks one of the sets of secret shares using these masks.[2] The other data set of secret shares and the data set of masks are masked using the aide's mask k (which must be queried independently by each contributor and is not known to the recipient). All three data sets are then delivered to the recipient and concatenated to obtain two overall data sets of secret shares (called S_1 and S_2) and one data set of masks (called M_s). These are unmasked and randomly permuted by the aide (with the unmasking ensuring that this permutation is done obliviously from the recipient's perspective). Finally, they are masked (using a random mask r chosen by the aide), returned, and reconstructed by the recipient, who counts the multiplicity of each row of the result to determine the total number of items known to n parties, to $n-1$ parties, to $n-2$ parties, and so on (as well as the overall number of unique items). See Fig. 2 for an example. Apart from these n distinct totals, nothing is revealed about which contributors knew which rows.

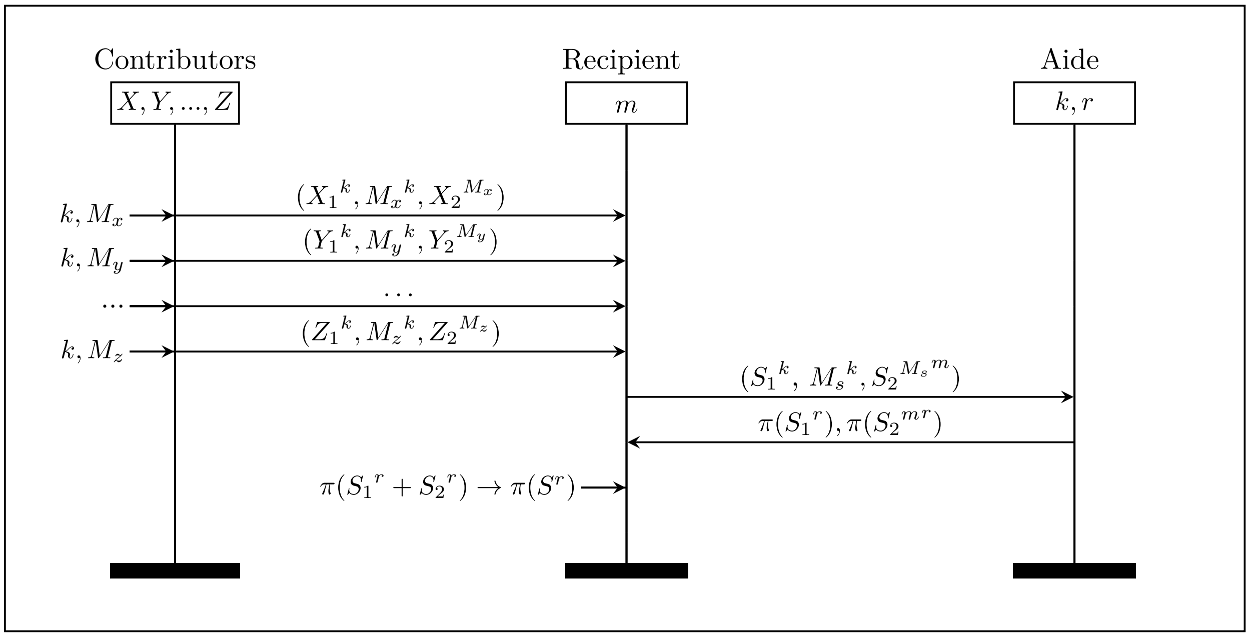

Contributors
$X, Y, ..., Z$
Recipient
m
Aide
k, r
k, M_x
$({X_1}^k, {M_x}^k, {X_2}^{M_x})$
k, M_y
$({Y_1}^k, {M_y}^k, {Y_2}^{M_y})$
$\dots$
$\dots$
k, M_z
$({Z_1}^k, {M_z}^k, {Z_2}^{M_z})$
$({S_1}^k,\ {M_s}^k, {S_2}^{{M_s}^m})$
$\pi({S_1}^r), \pi({S_2}^{mr})$
$\pi({S_1}^r + {S_2}^r) \to \pi(S^r)$

Figure 1: Diagram illustrating sequence of interactions in the protocol for computing intersection size.

Note that the data set values are not decipherable by the recipient because it does not know the aide's key k, and also does not know the masks in the data sets of masks. Only the lengths of contributions are revealed to the recipient.[3].

Note also that the recipient applies its own mask m to one of the data sets of secret shares before sending them to the aide. The aide is able to compute $({S_1}^k)^{k^{-1}} = S_1$ using the modular[4] inverse k^{-1} of its mask k, as well as $(({S_2}^{M_s})^m)^{{M_s}^{-1}} = (((S_2)^{M_s})^{M_s^{-1}})^m = {S_2}^m$. The aide can then shuffle these (by choosing a permutation in a pseudorandom manner and applying that permutation to both) and return them to the recipient. Because any secret sharing scheme yields shares that are indistinguishable from the underlying secret value, and because the aide also does not know the mask m, no reconstruction is possible and the aide learns nothing about the contributed data (other than the total number of contributed rows).

[2]If the same mask were used for all rows in a contributed data set, it would be possible for the aide to determine the size of each contributions by observing from which rows each contributor's single mask must be removed.

[3]Contributors can pad their data set in order to conceal some information about the size of their contribution. See Kamara *et al.* [7] for size-hiding options employing padding.

[4]This operation is performed modulo $2^{252} + 27742317777372353535851937790883648493$ in Curve25519-based groups such as Ristretto.

Once returned to the recipient, S_2^r can be unmasked using m^{-1}. Note that the values of the shares in S_1^r and S_2^r (and in S_1 or S_2, for that matter) have never been seen before by the recipient, so the recipient cannot match them to anything originally received from the contributors. The recipient computes $\pi(S_1^r + S_2^r) = \pi(S^r)$. This data set has the same element multiplicities as S itself, allowing the recipient to determine the multiplicity of each row. The rows that have multiplicity n (*i.e.*, those rows contributed by every contributor) belong to the overall intersection.

3.3 Other Terminology

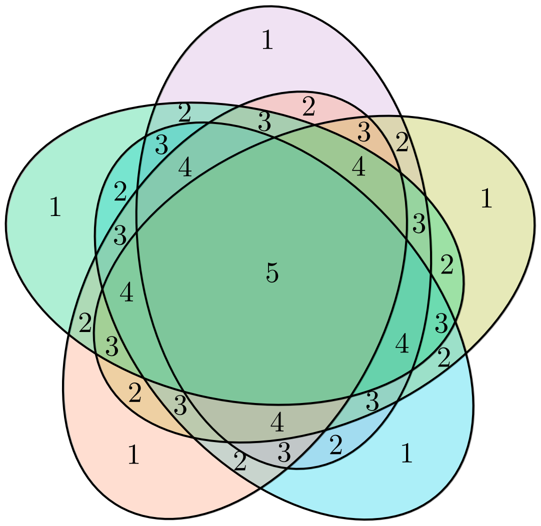


Figure 2: The 31 disjoint subsets that make up the 5 intermediate intersections of a 5-contributor PSI.

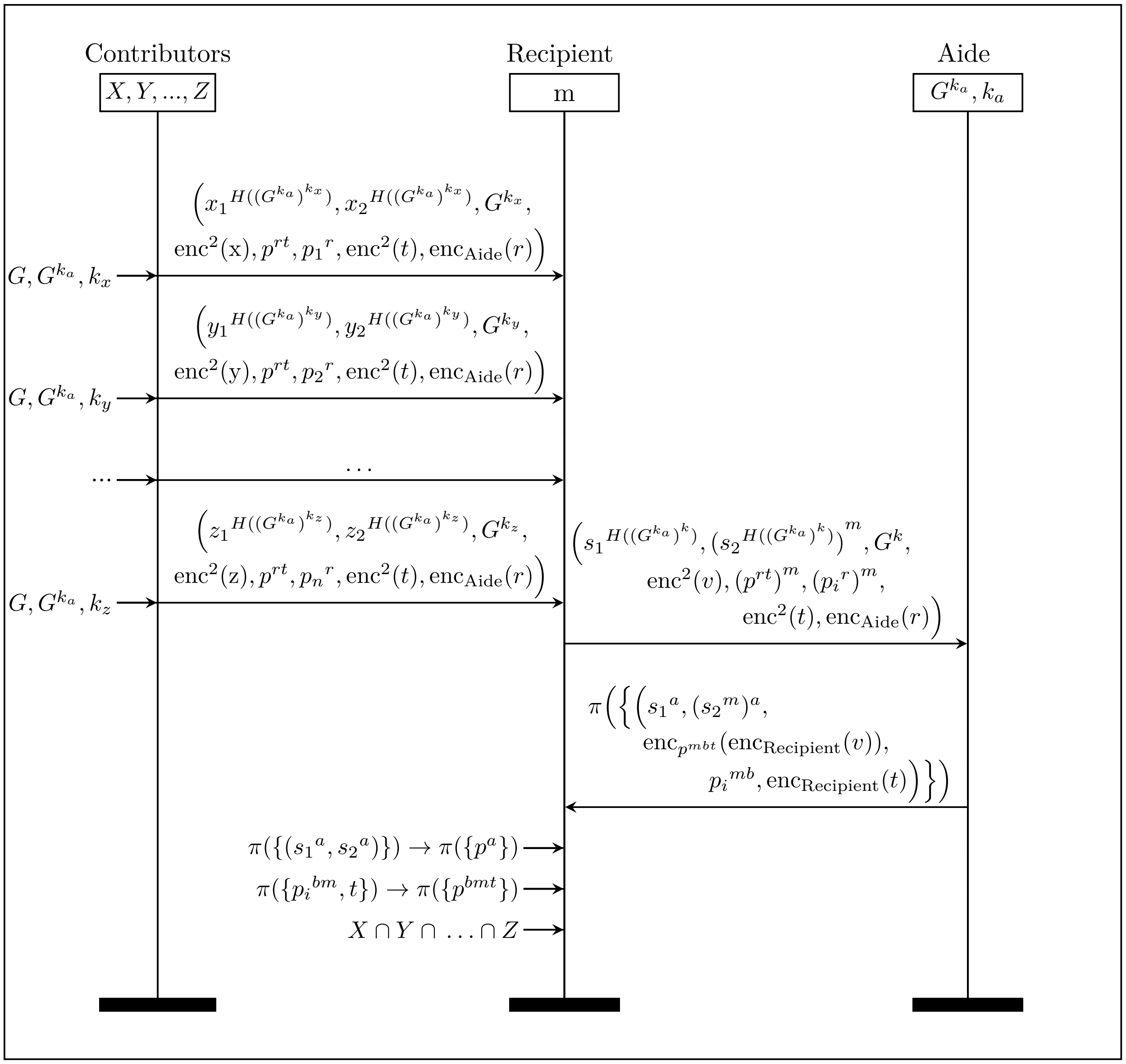
For illustration purposes, Fig. 2 presents the final intersection that can be revealed to the recipient within a protocol, as well as disjoint (but adjacent) regions making up the $n-1$ (here, $n-1=4$) intermediate intersections (the sizes of which may or may not be appropriate to reveal to the recipient, depending on the use case). All the regions labeled 3 are what we call the 3rd intermediate intersection, for example. The figure can also visually explain how contributors are anonymous from the perspective of the recipient. For example, if the five sets are intersected, the recipient would learn the total size of all regions labeled 1, or 4, or 1 and 4, *etc.*, but never anything asymmetrical (*e.g.*, never just blue, or the region corresponding to a specific contributor).

4 Protocol

The simple protocol presented in Section 3 (which allows the recipient to compute the intersection size) reveals the size of intermediate intersections (such as the number of unique rows, the number of rows known only to exactly two parties, and so on). It is also not secure against contributor impersonation: if the recipient knows k, then all security properties are lost (*e.g.*, the recipient can lie about the results without being detected) and all privacy properties are lost (*e.g.*, input data is leaked). In this section, we present a protocol that allows the recipient to learn the actual data in the intersection.

If the recipient were also a contributor, it would already possess all rows that could occur in the intersection, making this case a trivial reduction to the protocol presented in Section 3. Thus, we focus on the case in which the recipient *is not* a contributor, though we do aim to prevent the recipient from impersonating a contributor. We also assume that the contributors know a public key for the aide, G^{k_a}, and a public base point, G. We propose to perform masking via scalar-point multiplication on an elliptic curve, though any commutative group operation for which computing discrete logarithms is hard should suffice.

Let H be any function that casts or hashes a point to a valid scalar. Clearing the last nibble (in Curve25519-based groups, like Ristretto) should be sufficient because $256 - 252 = 4$ and $L > 2^{252}$.



Contributors
$X, Y, ..., Z$
Recipient
m
Aide
G^{k_a}, k_a
$\Big(x_1{}^{H((G^{k_a})^{k_x})}, x_2{}^{H((G^{k_a})^{k_x})}, G^{k_x},$
$\mathrm{enc}^2(\mathrm{x}), p^{rt}, p_1{}^r, \mathrm{enc}^2(t), \mathrm{enc}_{\mathrm{Aide}}(r)\Big)$
G, G^{k_a}, k_x
$\Big(y_1{}^{H((G^{k_a})^{k_y})}, y_2{}^{H((G^{k_a})^{k_y})}, G^{k_y},$
$\mathrm{enc}^2(\mathrm{y}), p^{rt}, p_2{}^r, \mathrm{enc}^2(t), \mathrm{enc}_{\mathrm{Aide}}(r)\Big)$
G, G^{k_a}, k_y
$\ldots$
$\ldots$
$\Big(z_1{}^{H((G^{k_a})^{k_z})}, z_2{}^{H((G^{k_a})^{k_z})}, G^{k_z},$
$\mathrm{enc}^2(\mathrm{z}), p^{rt}, p_n{}^r, \mathrm{enc}^2(t), \mathrm{enc}_{\mathrm{Aide}}(r)\Big)$
G, G^{k_a}, k_z
$\Big(s_1{}^{H((G^{k_a})^k)}, \big(s_2{}^{H((G^{k_a})^k)}\big)^m, G^k,$
$\mathrm{enc}^2(v), (p^{rt})^m, (p_i{}^r)^m,$
$\mathrm{enc}^2(t), \mathrm{enc}_{\mathrm{Aide}}(r)\Big)$
$\pi\Big(\Big\{\Big(s_1{}^a, (s_2{}^m)^a,$
$\mathrm{enc}_{p^{mbt}}(\mathrm{enc}_{\mathrm{Recipient}}(v)),$
$p_i{}^{mb}, \mathrm{enc}_{\mathrm{Recipient}}(t)\Big)\Big\}\Big)$
$\pi(\{(s_1{}^a, s_2{}^a)\}) \to \pi(\{p^a\})$
$\pi(\{p_i{}^{bm}, t\}) \to \pi(\{p^{bmt}\})$
$X \cap Y \cap \ldots \cap Z$

Figure 3: Diagram illustrating sequence of interactions within the intersection protocol.

To contribute an ordered collection of rows, for each row, the contributor computes the tuple

$$\Big(s_1{}^{H((G^{k_a})^k)}, s_2{}^{H((G^{k_a})^k)}, G^k, \mathrm{enc}_{\mathrm{Aide}}(\mathrm{enc}_{\mathrm{Recipient}}(v)), p^{rt}, p_i{}^r, \mathrm{enc}_{\mathrm{Aide}}(\mathrm{enc}_{\mathrm{Recipient}}(t)), \mathrm{enc}_{\mathrm{Aide}}(r)\Big)$$

where $p = \text{MapToCurvePoint}(\text{hash}(v))$ is the curve point mapped[5] from the hash of the row data v, where s_1 and s_2 are points and secret shares of p with threshold 2, p_i is a unique[6] secret share of the point p with threshold n for n parties, r is a random scalar mask, k is a random scalar secret key, and t is a random scalar (used to tweak the symmetric encryption of rows to prevent the aide from counting identical keys[7]). None of these randomly generated values are reused in multiple rows. The recipient then collects all contributions of such rows and for each row computes the tuple

$$\Big({s_1}^{H((G^{k_a})^k)}, \left({s_2}^{H((G^{k_a})^k)}\right)^m, G^k, \text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(v)),$$
$$(p^{rt})^m, ({p_i}^r)^m, \text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(t)), \text{enc}_{\text{Aide}}(r)\Big)$$

where m is a random scalar mask and is the same for every row in this computation. The recipient then sends the above tuple to the aide, who computes the tuple

$$\Big({s_1}^a, ({s_2}^m)^a, \text{enc}_{p^{mbt}}(\text{enc}_{\text{Recipient}}(v)), {p_i}^{mb}, \text{enc}_{\text{Recipient}}(t)\Big)$$

where a and b are random scalar masks reused for all rows in the computation. Finally, the aide sends the above tuple back to the recipient, who computes p^a for each row from ${s_1}^a + {s_2}^a$, and reconstructs p^{mb} on all rows that share n of the same identifier p^a. The value p^{mb}, when combined with the tweak to obtain p^{mbt}, decrypts the value of $\text{enc}_{\text{Recipient}}(v)$ (thus, decrypting the original row value).

4.1 Specialized Variant

A variant of the above protocol can be constructed that does not leak *any* intermediate rows or intersection sizes, but requires the set union of the data to be public, which in many cases it already is (*e.g.*, if the data consists of social security numbers or perhaps even email addresses). Instead of contributors computing the tuple

$$\Big({s_1}^{H((G^{k_a})^k)}, {s_2}^{H((G^{k_a})^k)}, G^k, \text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(v)), p^{rt}, {p_i}^r, \text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(t)), \text{enc}_{\text{Aide}}(r)\Big)$$

for every secret row s in their data set, they compute

$$\Big({\overline{s}_1}^{H((G^{k_a})^k)}, {\overline{s}_2}^{H((G^{k_a})^k)}, G^k, \text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(v)), p^{rt}, \overline{p_i}^r, \text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(t)), \text{enc}_{\text{Aide}}(r)\Big)$$

for every possible row $\overline{s}$ (in the multi-party set union $\overline{S}$).

When $\overline{s}$ is not equal to any row s in the party's own data set S, then $\overline{p_i}$ is chosen to be a random point (rather then the share of the mapping of the hash of $\overline{s}$ as a point). For efficiency, and depending on whether the recipient knows the union of all possible rows,[8] the row value v in the ciphertext $\text{enc}_{\text{Aide}}(\text{enc}_{\text{Recipient}}(v))$ may be replaced with a uniquely identifying row ID.

In the original protocol, the number of secret shares corresponding to a potential row in the final intersection would be equivalent to the number of parties who contributed that encrypted row (this information may be quite sensitive). In this variant, no sizes of intermediate intersections are revealed, because the recipient is given exactly the same number of shares for all possible rows in the intersection. Note also that these shares will only reconstruct to a valid key if and only if all contributors submitted the row that this key decrypts.

[5] The authors choose to use the Elligator map for this [6].

[6] Contributors do not need to get their unique secret share identifier (usually an x-coordinate) i from the aide, because the recipient has nothing to gain by lying about this and can simply tell each new contributor a new identifier. Depending on the secret sharing scheme, this i can be random, such that interaction with the recipient is not necessary.

[7] The value of p^m can be the same between rows, but the value of p^{tm} which is a value the aide computes never is.

[8] It may not, if the recipient party is separated into an oblivious compute-only server and a thin recipient.

5 Analysis

Our threat model is informed by the practical concerns that might apply when deploying and operating a RESTful, cloud-based workflow, application, or service.

5.1 Security

First, we show security against an honest-but-curious recipient and aide, and malicious contributors. Afterwards, we show security against all malicious (but non-colluding) parties.

For privacy, we aim to show that: (1) data contributors learn no new private data by contributing, (2) the designated recipient learns only the intersection (or sizes of data sets sent to it from the contributors), and (3) the aide learns nothing but the size of the data set sent to it by the recipient.

For security, we aim to show that no party can undetectably influence the result of the computation. A contributor *may* choose to not contribute rows it wants to exclude from the intersection, or even contribute rows it guesses the other contributors may be contributing, and we consider this acceptable given that this is unavoidable in general[9] in a secure multi-party computation.

Because the contributors do not receive information from any other party (except for a potentially random party ID from the recipient, and a public base point and public key from the aide, neither of which necessarily are sensitive), and because general MPC attacks[10] are out of scope, we will consider the protocol secure against attacks from contributors, joint or individual. A possible (but preventable) attack against one contributor by all other contributors *and* the recipient in the malicious case is described further below.

A curious recipient is ultimately interested in learning the rows contributed by the contributors. Before receiving a response from the aide, it only has seven values per contributor to analyze. We show through a proof of semantic security that these values do not provide a feasible way to reconstruct any new information about the contributors' row data. Information such as the sizes of intermediate intersections is included in this analysis. Within the notion of semantic security, we treat the row tuples as multipart ciphertexts that "encrypt" information about their corresponding rows. Four of these values depend (via scalar multiplication) on a key k or scalar r chosen at random for each row. A recipient attacker who could feasibly extract information from these curve points (such as distinguishing them from random points) would be able to break at least one of the Diffie–Hellman assumptions for this elliptic curve. Thus, these four values are semantically secure. The other three are protected by the aide's public key. Likewise, a recipient attacker able to extract information about the plaintexts encrypted with the aide's public key would be breaking public-key cryptography; hence, all seven values are individually semantically secure. A recipient attacker cannot distinguish any subset of these values, collectively, from a set of equal-length, equally-distributed random values (because these values have no algebraic relationships connecting one to any other). Furthermore, up to negligible probability, all are unique (except for their length, which should be identical for all of them).

The aide receives the exact same data as the recipient, except it is all combined into one data set (it could even be shuffled, though this has no effect); the second, fifth, and sixth value in each tuple is multiplied by the recipient's random mask m. The only secret information the aide can use to distinguish these values from random values that the recipient did not have is the value of its secret scalar k_a and its asymmetric decryption key. Note that there is no adaptive means of attack for the aide, because the recipient will never send it data related to this same request.

[9]Limiting the size of a contribution is, in specific use cases, one safeguard against guessing attacks. However, this paper will not speculate about such scenarios.

[10]See footnote above.

Using its two secrets, it can compute s_1 (a uniquely random value) and ${s_2}^m$ (related to s_1, but masked by m), as well as two other values masked by the recipient's mask m. We assert that for an attacker aide to distinguish s_1 or s_1 from random points on the curve, it must also be able to compute $s_1 + s_2$ and $(s_1 + s_2)^m$ (both, if computed, would likely be a value that reoccurs in other row tuples, thus leading to distinguishability). The ability of the attacker to compute $s_1 + s_2$ from $s_1 + {s_2}^m$ implies the ability to compute x^{-m} for a point x in general. The ability of the attacker to compute $(s_1 + s_2)^m$ from $s_1 + {s_2}^m$ implies the ability to compute x^m from y^m (and for two points x and y in general). Both violate Diffie–Hellman assumptions for this elliptic curve, which we assume apply to the aide's computational abilities; thus, the point values sent to the aide are semantically secure. Furthermore, encryption protects the row data and the tweak t, and r is uniquely random for every row. Only $(p^{rt})^m$ and $({p_i}^r)^m$ are related to r, but these are masked by m, so computing p^{mt} and ${p_i}^m$ from these reveals nothing to the aide (because the tweak is randomly unique per row, and so is p_i, even though m is only random globally).

Finally, the aide's response to the recipient allows it to compute p^a for every row, but because a is only random globally, it can perform frequency analysis on the contributed rows through p^a and learn the intermediate intersections.[11] The recipient can compute ${p_i}^{mbt}$ (which is uniquely random per-row), but it can only reconstruct[12] it into the key which decrypts a row value if, and only if, it has every contributor's share for that row.

In case the recipient is malicious, we must rule out the existence of any adaptive attacks. Because the recipient is the only[13] party that gets an interactive response back from the other parties, we limit our analysis to adaptive attacks carried out by the recipient. Consider the aide to be an oracle that the recipient can query. Assuming the aide can cache the $H(G^{k_c})$ portion of old requests, which we later show the recipient cannot refresh in a replay attack, the recipient will not learn more after its initial query, and so we will only consider the first query. The recipient may submit a tuple of the form $(u, v, w, c_1, x, y, c_2, c_3)$ and receive back the tuple

$$\left(u^{H(w^{k_a})\cdot a}, v^{H(w^{k_a})\cdot a}, \mathrm{enc}_{x^{\mathrm{dec}_{\mathrm{Aide}}(c_3)\cdot b}}(\mathrm{dec}_{\mathrm{Aide}}(c_1)), y^{\mathrm{dec}_{\mathrm{Aide}}(c_3)\cdot b}, \mathrm{dec}_{\mathrm{Aide}}(c_2)\right).$$

It is critical we prevent replay attacks, because the last element is simply the decryption of a recipient-chosen ciphertext using the aide's asymmetric key, which the recipient could[14] use to decrypt an actual contributed row of data. With overwhelming probability, the values of $H(w^{k_a})$ and a are unique per row, therefore the recipient's ability to use the aide as an oracle to multiply two arbitrary points by $H(w^{k_a}) \cdot a$ can not be used to distinguish between pseudorandom values in any other row. Within this row, the only values that depend on $H(w^{k_a})$ or a or k_a are the intended multiplicands; thus, no adaptive attack exists using this part of the aide's computation. Using the aide as an oracle to encrypt with the value $x^{\mathrm{dec}_{\mathrm{Aide}}(c_3)\cdot b}$ as the key does not, by semantic security of encryption, feasibly reveal anything about the plaintext $\mathrm{dec}_{\mathrm{Aide}}(c_1)$ unless the recipient knows $x^{\mathrm{dec}_{\mathrm{Aide}}(c_3)\cdot b} = p^{mbt}$ (computability of p^{mbt} is discussed above in this section). Likewise, the ability of the recipient to use the aide as an oracle to multiply arbitrary points by $\mathrm{dec}_{\mathrm{Aide}}(c_3) \cdot b$ can not be used to distinguish between pseudorandom values in any other row, as b is randomly chosen by the aide for each row, and a recipient capable of computing b from $y^{\mathrm{dec}_{\mathrm{Aide}}(c_3)\cdot b}$ cannot

[11] This is the only privacy leak. Notice that the frequency of values of the different p^a values in the specialized variant is uniform. Interestingly, the p^a value could be omitted if the recipient were allowed *quadratic* time to find all reconstructable sets of shares.

[12] The security of this is determined by the secret sharing scheme used. The scheme must be implementable with only addition and scalar multiplication, and most are.

[13] Aside from contributors who learn the final output, but the recipient *can* be a contributor, also.

[14] At the expense of corrupting the final intersection result. Plus, this can be prevented by having contributors submit $\mathrm{enc}_{\mathrm{Aide}}(\mathrm{enc}_t(\mathrm{enc}_{\mathrm{Recipient}}(v))$ instead of $\mathrm{enc}_{\mathrm{Aide}}(\mathrm{enc}_{\mathrm{Recipient}}(v))$ for a row v.

exist under the discrete logarithm hardness assumption for this curve. Because we have shown the information individual parties can compute passively or adaptively does not allow them to distinguish masked values from equally-distributed random values, the messages in all parts of this protocol are semantically secure, so long as there is no collusion. Trivially, the recipient and aide can reveal all the data in the computation by colluding and sharing the recipient's mask m and the secret r known only to the aide. Thus, we require those two parties to reside in separate trust domains. Finally, because all row values are masked (the security of this masking is demonstrated above) by row-specific random values, contributors colluding with the aide cannot learn anything more about other non-colluding contributors than the aide could on its own (*i.e.*, only negligible information).

Second Secret Sharing Scheme. Parties must generate the secret share of p, denoted p_i for a contributor C_i, by using a *random* share generating algorithm. Any *deterministic* scheme allows the recipient to simply simulate each contributors' share generation process, but the recipient cannot determine which secret was shared because (as previously proven) these shares are always tweaked by either r or t, both of which are unpredictable and row-wise random. Because contributors must generate their shares of potentially jointly held secret rows in isolation, they cannot use schemes such as additive secret sharing, as in those schemes, shares cannot be generated completely in parallel (*i.e.*, shares directly depend on the value of other shares). This would require the recipient (or a stateful aide) to give contributors unique sequential indices for which to split each secret. Using Shamir's secret sharing scheme, with the secret polynomial instantiated deterministically using the hash of the secret, is a safe choice. This safety is due to the unknowable nature of r or t and that servers can compute $(s_1 + s_2)^r = {s_1}^r + {s_2}^r$ for points on a curve without knowing r or unmasking s_1 or s_2.

Verifiability. For verifiability, contributors must also provide an additional piece of information: the signature $H(c_1)^{H(G^{ak_c})}$ of row $c \in C$. This prevents the recipient from maliciously remasking and replaying the data in a future computation. Additionally, to reduce the amount of past data that the aide must remember to prevent replay attacks, the contributors could submit a timestamp/expiry (encrypted or even as plaintext) which is used (in any way) to seed H. After a period of time, the aide can clear and reject old contributions.

5.2 Scalability

Each party only ever sends and receives one message. All messages are at most linearly proportional to the total number of rows contributed because each tuple computed for each row is constant[15] in size. For local computation, each row tuple is processed (always pair-wise) a constant number of times. The only exception is that the recipient must find all corresponding secret shares (if any) in linear time, but through the use of a hash table (indexed on p^a), this cost can be achieved on average. For further illustration, if it takes s seconds of online computation to compute the n-party intersection over m total input rows, the same computation with n' additional parties will take $s \cdot \frac{(n+n')\cdot m}{nm}$ seconds. Likewise, the same computation with m additional rows of data will take $s \cdot \frac{n \cdot (m+m')}{nm}$ seconds.

[15]The representation size is bounded by the maximum possible row length (*e.g.*, 320 bytes if rows encode email addresses).

5.3 Correctness

As evident in the diagrams, each contributor sends no more than one message to the recipient, and their message is not dependent on the contributions of any other contributor. After the contributions are received, the recipient only exchanges one message with the aide. Thus, our protocol satisfies the asynchronicity requirement. We have shown that the privacy and scalability requirements are satisfied in the subsections above.

6 Conclusions and Future Work

We have presented a PSI protocol that demonstrates the feasibility of protecting intersection size information from the additional party that is introduced to enable asynchronicity.

Though we assume in this paper that the party performing the computation and the recipient of the result are the same, separating these two roles between two separate parties is possible. This can be done by having the aide apply an additional scalar mask to all returned elliptic curve points, and then having it encrypt this mask with the recipient's public key.[16] If the contributors do not trust the recipient not to give the aide its own public key (for fear of a man-in-the-middle attack), the aide could choose to have the contributors submit the recipient's public key (encrypted with the aide's public key) as input and can ensure there is no discrepancy before continuing. Either way, the contributors and separate recipient (if such exists) need not ever communicate[17] with the aide directly[18].

Depending on the secret sharing scheme used for sharing row data decryption keys, the contributors may not need to know exactly how many of them are involved (though they still need to know a lower bound), as secret sharing is what facilitates hiding the larger, intermediate intersections. We would also like to determine in the future whether there can be an asynchronous multi-party PSI protocol in which the additional party only does a constant amount of work. If so, we are interested in investigating whether the work performed by this party can be limited to helping with key management. We are also interested in investigating methods to mask the set union $\overline{S}$ in the specialized variant such that we can hide intersection sizes even when the set union is private.

References

[1] A. Abadi, S. Terzis, and C. Dong. Feather: Lightweight multi-party updatable delegated private set intersection. Cryptology ePrint Archive, Paper 2020/407, 2020. `https://eprint.iacr.org/2020/407`.

[2] A. Abadi, S. Terzis, R. Metere, and C. Dong. Efficient delegated private set intersection on outsourced private datasets. Cryptology ePrint Archive, Paper 2018/496, 2018. `https://eprint.iacr.org/2018/496`.

[3] A. Abadi, S. Terzis, R. Metere, and C. Dong. Efficient delegated private set intersection on outsourced private datasets. Cryptology ePrint Archive, Paper 2018/496, 2018. `https://eprint.iacr.org/2018/496`.

[16] The recipient must always do an amount of work proportional to the total number of contributed rows, because the party performing the computation cannot reduce the size of what it sends to the recipient without learning sensitive information. Work by Kamara *et al.* [7] describes options for size-hiding using padding.

[17] For example, using the aide's REST API.

[18] With the exception of retrieving the aide's public key, no identity management is needed.

[4] K. D. Albab, R. Issa, A. Lapets, P. Flockhart, L. Qin, and I. Globus-Harris. Tutorial: Deploying Secure Multi-Party Computation on the Web Using JIFF. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 3–3, September 2019.

[5] S. Badrinarayanan, P. Miao, and T. Xie. Updatable private set intersection. Cryptology ePrint Archive, Paper 2021/1349, 2021. `https://eprint.iacr.org/2021/1349`.

[6] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. Cryptology ePrint Archive, Paper 2013/325, 2013. `https://eprint.iacr.org/2013/325`.

[7] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In *Financial Cryptography*, pages 863–874, 2014.

[8] A. Lapets, K. D. Albab, R. Issa, L. Qin, M. Varia, A. Bestavros, and F. Jansen. Role-Based Ecosystem for the Design, Development, and Deployment of Secure Multi-Party Data Analytics Applications. In *2019 IEEE Cybersecurity Development (SecDev)*, pages 129–140, September 2019.

[9] L. Qin, P. Flockhart, A. Lapets, K. D. Albab, M. Varia, S. Roberts, and I. Globus-Harris. From usability to secure computing and back again. In *Proceedings of the Fifteenth USENIX Conference on Usable Privacy and Security*, SOUPS '19, pages 191–210, USA, August 2019. USENIX Association.