Re-visiting Authorized Private Set Intersection: A New Privacy-Preserving Variant and Two Protocols

Francesca Falzon
ffalzon@ethz.ch
ETH Zürich
Zürich, Switzerland

Evangelia Anna Markatou
e.a.markatou@tudelft.nl
TU Delft
Delft, Netherlands

Abstract

We revisit the problem of Authorized Private Set Intersection (APSI), which allows mutually untrusting parties to authorize their items using a trusted third-party judge before privately computing the intersection. We also initiate the study of Partial-APSI, a novel privacy-preserving generalization of APSI in which the client only reveals a subset of their items to a third-party semi-honest judge for authorization. Partial-APSI allows for partial verification of the set, preserving the privacy of the party whose items are being verified. Both APSI and Partial-APSI have a number of applications, including genome matching, ad conversion, and compliance with privacy policies such as the GDPR.

We present two protocols based on bilinear pairings with linear communication. The first realizes the APSI functionality, is secure against a malicious client, and requires only one round of communication during the online phase. Our second protocol realizes the Partial-APSI functionality and is secure against a client that may maliciously inject elements into its input set, but who follows the protocol semi-honestly otherwise. We formally prove correctness and security of these protocols and provide an experimental evaluation to demonstrate their practicality. Our protocols can be efficiently run on commodity hardware. We also show that our protocols are massively parallelizable by running our experiments on a compute grid across 50 cores.

Keywords

Private Set Intersection, 2PC, Secure Multiparty Computation

1 Introduction

Private set intersection (PSI) enables two parties – a client and a server, for example – to securely compute the intersection of their respective sets, $\mathcal{X}$ and $\mathcal{Y}$. One or both parties learn the intersection $\mathcal{X} \cap \mathcal{Y}$, but neither party should learn anything about the other party's elements not contained in the intersection. This problem naturally arises in many domains, including proximity testing [35], the testing of sequenced human genomes [5, 36], botnet detection [32], and Apple AirDrop [24]. PSI has also been proposed by Facebook [8, 9] and Google [26] to measure ad conversion rates by comparing the list of people who have seen an advert with those who have completed a transaction.

Proceedings on Privacy Enhancing Technologies 2025(1), 792–807

https://doi.org/10.56553/popets-2025-0041

However, even if a PSI protocol is secure in the malicious model, parties can still inject elements into their set in order to mislead the other party or to learn more information about the other party's set. If the universe is enumerable, a malicious party could go so far as to add every element in the universe to their set. **Authorized private set intersection** (APSI) mitigates this attack by requiring the items in the input set to be authorized by a trusted third party, or **judge**, before the intersection is computed [10, 12]. The judge is only involved in authorization and then goes offline.

While APSI offers good guarantees over PSI for preventing such attacks, the party authorizing their elements must reveal their entire set to the judge, which may be unreasonable. For example, revealing the entire set is unnecessary if the client undergoes an audit in which only a strict subset of the input is checked; revealing elements beyond those requested only harms the privacy of the audited client. We propose a new privacy-preserving variant of APSI, which we call **partial authorized private set intersection** (Partial-APSI). Partial-APSI requires the client to pre-commit to their set and send this commitment to the judge. The client later reveals a subset of their items – of the judge's choosing – to the judge. Importantly, the judge picks the subset to prevent the client from hand-picking elements that it knows would be approved. Partially revealing the input is useful in scenarios where revealing the entire data set (e.g., an entire human genome) is undesirable, but revealing a few elements (e.g., genes) may still result in sufficient privacy.

Below we highlight some important applications.

Application 1: Ad conversion. A social media company and a store wish to compute the intersection of the set of people who purchased an item from the store and the set of people who viewed the store's ad. The social media company may require the store to have an auditing firm authorize the store's input data via an APSI protocol. If the client's set is sufficiently large, the auditing firm may instead only randomly sample the data and sign the set if the selected elements are valid. In the latter case, a Partial-APSI protocol can be used to ensure that the client list is not fully revealed.

Application 2: Privacy policy compliance. Data minimization is central to privacy policies such as the European Union's General Data Protection Regulation (GDPR). GDPR Article 5(1)(c) states that personal data must be "adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed" [1]. Partial-APSI supports compliance by allowing parties to only reveal necessary elements during routine audits before intersection computations, thereby minimizing data sharing.

Application 3: Genome matching. PSI on DNA sequences is useful in a number of settings including clinical (e.g., comparing patient genomes), criminal (e.g., matching DNA of a suspect to that found on a crime scene), personal (e.g., ancestry testing), and

Protocol	Security	Assumptions	Multi-runs	Authorize		Intersect	
				# Rounds	Comm.	# Rounds	Comm.
DT10 [14]	Input-Malicious	RSA & ROM	✓	2	$2n+2$	2	$2+2n+m$
DKT10 [13]	Malicious	RSA, DDH, & ROM	✓	2	$2n+2$	2	$1+5n+m$
SSS12 [38]	Malicious	CBDH & ROM	✓	2	$2n+3$	–	–
DD15 [15]	Malicious	Quadratic Residuosity	✓	2	$2n+2$	2	$2n+m$
Ours: APSI	Malicious Client	DBDH & ROM	✓	2	$2n+2$	1	$m+1$
Ours: Partial-APSI	Input-Malicious	DBDH & ROM & DDH	✓	4	$2(p+1)n+2$	3	$3m+1$

Table 1: A comparison with prior APSI work. Here, n and m denote the size of the client's and server's set, respectively. Input-Malicious refers to a client that may maliciously insert elements to its set prior to Intersect, but who then executes the protocol semi-honestly. Stefanov et al. [38] assumes that both parties authorize their elements and that the protocol has black-box use of a PSI protocol; for comparability, we report the communication complexity needed to authorize a single party's set.

commercial (e.g., patent disputes) settings. Genomes are immutable and private, and parties participating in PSI involving such sensitive data may wish to carry out the computation with a guarantee that the other party's data is valid; in such a case, APSI would be an ideal solution. Other parties may wish to minimize the data revealed to third parties, in which case, Partial-APSI may be the better alternative. For example, bioengineering companies have been discussing barcoding cell lines, by adding unique DNA sequences to genomes [27]. Genomes are relatively large, (e.g., mice genomes are $\approx$ 2.5GB) and there are locations to place barcodes that don't seem to affect the organism. Auditing the entire genome is expensive and the company might not wish to disclose the entire proprietary genome. Partial-APSI would minimize access to private information and discourage misuse of patented information. Recently, DNA-based genotyping has been used as an alternative to serological antibody-based methods to match blood donors to blood recipients [40]. DNA-based genotyping is able to test for antigens for which there are no serologic reagents. Both APSI and Partial-APSI could serve to match blood donors to patients needing transfusions.

We consider protocols between three parties: a judge, a client, and a server. The judge is assumed to be trusted, in the case of APSI, and semi-honest, in the case of Partial-APSI. The client and server hold sets $\mathcal{X}$ and $\mathcal{Y}$, respectively, and we consider one-sided protocols in which only the client learns the intersection $\mathcal{X} \cap \mathcal{Y}$. APSI and Partial-APSI can be broken down into two phases: (1) the initial Authorize phase in which the client is required to (partially or fully) authorize its elements and (2) the Intersect phase in which the client can jointly compute the intersection with the server. Throughout this paper, we assume that the client is **input-malicious** i.e., it may inject unauthorized elements into its input before authorization or before computing the intersection.

Table 2 describes the notation used throughout this paper.

n	size of the sets $\mathcal{X}$
m	size of the sets $\mathcal{Y}$
p	fraction of elements to be revealed to judge
$[n]$	the set $\{1, \ldots, n\}$ for some integer n
$\perp$	empty string
$x \leftarrow\!\!\$\, \mathcal{X}$	x is chosen uniformly at random from a set $\mathcal{X}$
$T \subseteq_{\$} S$	randomly chosen subset T of a set S
$\overset{c}{\approx}$	computational indistinguishability

Table 2: Notation.

1.1 Our Contributions

We revisit APSI and propose a new protocol based on bilinear pairings that is bandwidth efficient and round-optimal, requiring fewer interactions than previous proposed solutions. The server encodes its elements using some randomness, and sends the encoded elements to the client along with a commitment to the randomness. The client can then recover the intersection from these elements. Our APSI protocol is secure against a malicious client and a semi-honest server, assuming the Decisional Bilinear Diffie-Hellman assumption and in the random oracle model. Our threat model is inspired by the work of Hazay and Lindell [22, 23]. Since our APSI functionality is one-sided (i.e., only the client learns the output), we consider a relaxed notion of security in which only one party – the client authorizing its elements and receiving the intersection – is fully simulatable. Hazay and Lindell note that such a relaxation is often sufficient in one-sided functionalities and that this notion has been considered in previous works [18, 34].

We introduce Partial-APSI, a generalization of APSI that addresses the loss of privacy when the client's entire input is presented to a semi-honest judge. Unlike in APSI where the client is required to reveal all of its elements to the judge, Partial-APSI only requires the client to present a partial, but computationally binding view of its elements to the judge. A Partial-APSI protocol is parameterized by a value $p \in (0, 1]$ which specifies the fraction of elements that the client must reveal to the judge. The client commits to its items and sends the commitments to the judge. The judge then requests openings for a p fraction of the set. If the revealed elements are valid, then the judge authorizes the entire set (e.g., by signing the committed values).

In Partial-APSI, the parameter p denotes the privacy afforded to the client. We model the judge as a semi-honest party (not fully trusted) that aims to learn as much information about the client's set. The server can therefore choose to interact with clients who have authorized their elements using a mutually approved judge and whose parameter p meets the server's preferred threshold. Note that when $p = 1$, the client must reveal its entire set to the judge, and thus Partial-APSI reduces to APSI. For this reason, we consider Partial-APSI to be a generalization of APSI.

We present a second protocol that realizes the Partial-APSI functionality. This protocol extends our APSI protocol to the Partial-APSI setting and is also based on bilinear pairings. However, in the authorize phase, the client must commit to its elements using a secret r to hide them from the judge. The intersection phase thus requires an additional two rounds of communication over the APSI protocol so that the client can blind the server's set with the same r using an oblivious PRF.

The performance of our two protocols is summarized in Table 1. Both our protocols achieve **multi-runs**: the client can authorize its elements once and then execute Intersect without revealing its signatures or the fact that its input set is the same. We assume that in the Authorize phase, the client authorizes their elements resulting in $2n$ communication (n elements to be sent to the judge and n signatures returned to the client). We also include an additive factor of 2 to account for sending the judge's public key to the client and server. The communication of Intersect in both of our protocols is independent of n; this results in an at least $2n$ multiplicative factor decrease in communication compared with prior work, whilst still achieving security against a malicious client.

Our contributions can be summarized as follows:

(1) We **describe a new practical APSI protocol** from bilinear pairings that outperforms prior work with respect to number of rounds and asymptotic communication costs. (Section 4)
(2) We **introduce and formalize** a privacy-preserving generalization of APSI, which we coin partial authorized private set intersection (Partial-APSI). (Section 2)
(3) We **extend our APSI protocol** to realize the Partial-APSI functionality. (Section 5)
(4) We support our protocols with a **formal analysis** of their correctness and security. We show that our APSI protocol is secure against a malicious client and semi-honest server, and our Partial-APSI protocol is secure against an input-malicious client, and semi-honest judge and server. (Sections 4 and 5)
(5) We **implement our two protocols** and demonstrate their efficiency and highly parallelizable nature. (Section 6)

1.2 Related Work

PSI can be realized using general secure multi-party computation [41], but specialized protocols are generally more efficient. While the problem of PSI has been well-studied, the work done on APSI is more limited. For a general treatment of PSI refer to [31].

PSI for certified sets was first proposed by Camenisch and Zaverucha [10] as a way for a trusted third-party to authenticate and bind the input sets to each party. De Cristofaro et al. [12] proposed *privacy-preserving policy-based information transfer (PPIT)*, in which a third-party authorizes a client to retrieve a single piece of information from a server. The protocols of [10] and [12] require quadratic communication and complexity.

De Cristofaro and Tsudik [14] introduced *Authorized PSI (APSI)* as a generalization of PPIT. Their protocol is secure in the input-malicious model, with linear communication and complexity. This work was later extended to be secure in the malicious model [13].

Kerschbaum [28] presented an APSI protocol that relies on Bloom filters and homomorphic encryption. Debnath et al. [15] expanded on this work and presented a more efficient Bloom-filter-based construction for both APSI and *authorized private set intersection cardinality* (where parties only learn the size of the intersection). Faber et al. [16] presented a protocol for *authorized two-way private set intersection*; both parties are required to obtain authorizations on their input sets and both parties learn the intersection. Their work relies on a Diffie-Hellman key-exchange approach and is proven secure in the malicious model.

Policy-enhanced private set intersection (PPSI) was introduced as a variant of APSI in which parties must obtain authorizations from a set of certificate authorities, and can encode elements with different "policies" that certificates must satisfy [38]. Nagy et al.[33] proposed *Common Friends*, which allows parties in a social network to determine if they are friends or share common friends. The construction enforces access control in a privacy-preserving manner and uses Bloom filters for efficiency. D'Arco et al. [11] studied impossibility results of size-hiding PSI constructions; they demonstrated that unconditionally secure size-hiding PSI is only possible in the APSI setting. Their scheme requires exponential time and space.

The notion of authentication also appears in the outsourced PSI literature, albeit with a different security goal. In Outsourced-PSI [3], two clients outsource the intersection computation to an untrusted server; the protocol includes an authorization mechanism that ensures that the intersection is only computed with the permission of all the clients and that the result is hidden from the server. *Verifiable delegated PSI* [2, 39] considers the problem of how mutually-distrusting clients can verifiably delegate computation to a server and verify that the computation was performed correctly.

Most recently, Ghosh et al. [21] introduced *private certifier intersection*, the goal of which is to compute the set of common certifiers between two parties or more parties and validate the certificates.

2 Problem Definition

Both APSI and Partial-APSI proceed in two phases: (1) Authorize in which the client **C** authorizes its elements, and (2) Intersect in which the client **C** and the server **S** jointly compute the intersection of their sets. At the end of the protocol, the client **C** learns the intersection of its (authorized) elements with those of the server's and the server **S** learns nothing.

APSI supports authorization of a party's elements via a trusted third-party judge **J** [13–15]. Partial-APSI generalizes APSI by formalizing the judge as an honest-but-curious party, instead of a trusted third-party. In Partial-APSI, the judge only sees a p fraction of $\mathcal{X}$ and its goal is to infer as much information about $\mathcal{X}$ beyond the revealed elements.

2.1 Parties

Both APSI and Partial-APSI involve three parties:

- **Client C**: This party holds a set $\mathcal{X}$ and wishes to compute the intersection of its set with the set held by a server. **C** must authorize its elements before computing an intersection.
- **Server S**: This party holds a set $\mathcal{Y}$ and wishes to participate in computing the intersection of their set with that of **C**.
- **Judge J**: Prior to an intersection computation, the judge **J** either sees all elements in set $\mathcal{X}$ (APSI) or a p fraction of $\mathcal{X}$ of the judge's choosing (Partial-APSI); if the judge accepts the observed (sub)set, then it authorizes all the elements in $\mathcal{X}$.

Parties and their inputs: Client **C** with input sets $\mathcal{X}$ and $\mathcal{X}_1, \ldots, \mathcal{X}_q$; the server **S** with input sets $\mathcal{Y}_1, \ldots, \mathcal{Y}_q$; the judge **J** with input $\perp$.
Parameters: The set sizes $|\mathcal{X}|$, and $|\mathcal{X}_k|$ and $|\mathcal{Y}_k|$, for $k \in [q]$; the probability $p \in (0, 1]$.
Functionality:

- Authorize: The trusted third party **T** receives an authorization request from the client **C**, requesting **J** to authorize $\mathcal{X}$.
  - **T** forwards $\mathcal{X}$ to **J**, who can either accept or reject the items.
  - **T** samples $\mathcal{I} \subseteq_{\$} [n]$ such that $|\mathcal{I}| = \lceil pn \rceil$ and forwards $\{x_i \in \mathcal{X} : i \in \mathcal{I}\}$ to **J** , who can either accept or reject the request.
  - If **J** accepts, **T** replies `Accept` to **C** and remembers that **J** has authorized **C** on $\mathcal{X}$. Otherwise, **T** replies `Reject` to **C**.
- Intersect: **for** $k \in [q]$ **do**
  - **T** receives a request from party **C** to perform a set intersection with the server **S**. **T** forwards the request to **S**, and **C** and **S** run the ideal protocol below.
    - **C** sends a set $\mathcal{X}_k$ to **T**; or **C** aborts.
    - **S** sends a set $\mathcal{Y}_k$ to **T**; or **S** aborts.
    - **T** computes the intersection $(\mathcal{X}_k \cap \mathcal{X}) \cap \mathcal{Y}_k$ and sends the resulting intersection and the set size $|\mathcal{Y}_k|$ to **C**.

Figure 1: The Ideal Functionalities $\mathcal{F}_{\text{APSI}}$ and $\mathcal{F}_{\text{Partial-APSI}}$. Text in the dashed box denotes steps unique to APSI and text in a solid box denotes steps unique to Partial-APSI.

2.2 Security and Threat Model

We prove security for both APSI and Partial-APSI using the real-ideal paradigm. In the real world, we execute our protocol and the parties interact with each other directly. In the ideal world, the parties **C**, **S**, and **J** interact with a trusted third party **T** who executes the ideal functionality $\mathcal{F}_{\text{APSI}}$ or $\mathcal{F}_{\text{Partial-APSI}}$ (Figure 1) on inputs chosen by the parties **C** and **S**.

In the Authorize phase, **C** submits its set $\mathcal{X}$ to **T**. In the case of $\mathcal{F}_{\text{APSI}}$, **T** forwards the entire set to the judge, whereas, in the case $\mathcal{F}_{\text{Partial-APSI}}$, **T** randomly selects a subset of size $\lceil pn \rceil$ and forwards this subset to the judge **J**. If the judge approves the elements it sees, then **T** forwards `Accept` to **C** and remembers the set $\mathcal{X}$. Otherwise **T** forwards `Reject` to **C**.

In the Intersect phase, **C** may initiate an intersection computation with the server **S** up to a polynomial q number of times. The intersection is mediated by **T**. **C** and **S** send their input sets $\mathcal{X}_k$ and $\mathcal{Y}_k$ to **T**, respectively (here $\mathcal{X}_k$ and $\mathcal{Y}_k$ denote the input sets of the k-th invocation of Intersect). **T** then computes $(\mathcal{X}_k \cap \mathcal{X}) \cap \mathcal{Y}_k$ and sends the result along with $|\mathcal{Y}_k|$ to **C**. When $\mathcal{X}_k = \mathcal{X}$, the client precisely learns the intersection $\mathcal{X} \cap \mathcal{Y}$. However, if **C** maliciously adds an element x to $\mathcal{X}_k$ such that $x \notin \mathcal{X}$, then x will still not appear in the k-th intersection, even if $x \in \mathcal{Y}_k$.

Throughout this work, we assume that the client might not have authorizations for all elements in its set and may try to inject additional elements into the input set at the start of either the authorization or intersection phase. In APSI, we assume that the judge is fully trusted and, in Partial-APSI, we assume that the judge is semi-honest (honest-but-curious) and may try to infer as much as it can about the client's set beyond the elements revealed.

In the **semi-honest setting**, the parties follow the protocol and may only try to passively infer information about the other party's input. In the **malicious setting**, the parties may deviate arbitrarily from the protocol; they may refuse to participate, abort prematurely, or modify their input. Even PSI protocols that are secure against malicious adversaries may be susceptible to parties adding additional elements to their set to try and learn additional information about the other party's set. Authorization thus ensures that the intersection is only computed on authorized elements. We assume that the judge and server do not collude.

We prove our APSI protocol is secure against a **malicious client** and **semi-honest server**. Since only the client learns the intersection, they have the most to gain by cheating. This relaxed notion of security for one-sided PSI in which only the party receiving the output is fully simulatable was formalized in [22, 23]. We also prove our Partial-APSI protocol secure against an **input-malicious client** and semi-honest judge and server. An input-malicious agent acts exactly like a semi-honest adversary, but may choose their input maliciously. The simulator uses the parties' inputs and outputs to create a view that is computationally indistinguishable from the real world. We define security of APSI formally as follows.

Definition 1. *Let $E = (E_1, \ldots, E_m)$ be a sequence of events where each E_i is either of the form $\langle \text{Authorize}, \mathbf{C}, \mathcal{X}, \mathbf{J}, \perp \rangle$ or $\langle \text{Intersect}, \mathbf{C}, (\mathcal{X}_1, \ldots, \mathcal{X}_q), \mathbf{S}, (\mathcal{Y}_1, \ldots, \mathcal{Y}_q) \rangle$. Let $\mathbf{Ideal}_{\mathcal{S},E}$ denote the joint output distribution of all parties and a simulator $\mathcal{S}$ in the ideal world under E, and $\mathbf{Real}_{\mathcal{A},E}$ denote the joint output distribution of all parties and the adversary $\mathcal{A}$ in the real world under E. A protocol Π **securely realizes** $\mathcal{F}_{\text{APSI}}$ if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there is a probabilistic polynomial time simulator $\mathcal{S}$ such that*

$$\mathbf{Ideal}_{\mathcal{S},E} \overset{c}{\approx} \mathbf{Real}_{\mathcal{A},E}.$$

In other words, the execution of our protocol (real world) should be indistinguishable from realizing the functionality with a trusted third-party carrying out the intersection (ideal world). Security of Partial-APSI is defined analogously, with the only change being that the APSI ideal functionality $\mathcal{F}_{\text{APSI}}$ is replaced by the Partial-APSI ideal functionality $\mathcal{F}_{\text{Partial-APSI}}$.

On Multi-Run Security. One important property of PSI protocols is **multi-run** security or **unlinkability**: the parties should be able to execute the intersect protocol multiple times without needing to re-authorize their elements. Consider the naive APSI protocol in which both parties authorize their elements and the intersection is carried out on the signatures. If this protocol is executed multiple times, the client must re-authorize its set each time in order to avoid leaking differences in its input between invocations. In the

Partial-APSI setting, this is problematic since, by doing so, the client would have to reveal additional elements from $\mathcal{X}$ each time.

Our protocols achieve multi-run security, and this notion is captured by our functionality. In the ideal functionality (Figure 1), no information is ever sent to the server and the client only ever learns the intersection $(\mathcal{X}_k \cap \mathcal{X}) \cap \mathcal{Y}_k$ and $|\mathcal{Y}_k|$, even when Intersect is invoked multiple times. In other words, neither the server nor the client can tell if two instances of Intersect are related. In the functionality, the trusted third party must remember $\mathcal{X}$ to ensure that only items that are authorized can appear in the intersection.

3 Cryptographic Building Blocks

We now describe the required cryptographic building blocks.

3.1 Bilinear Group

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order q. Our protocols make use of an efficient, non-degenerate **bilinear mapping** $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that for all group elements $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and all $a, b \in \mathbb{Z}$ we have that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.

In this work, we concern ourselves with **Type-3 pairings**, which are asymmetric pairings ($\mathbb{G}_1 \neq \mathbb{G}_2$) for which there are no efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$. Such pairings also admit a hash function $H_1 : \{0,1\} \to \mathbb{G}_1$. Type-3 pairings are efficient and representative of pairings libraries such as Mcl [37] and Miracl [30]. See [19] for a primer on pairings for cryptography.

3.2 Computational Assumptions

The **Decisional Diffie-Hellman (DDH)** assumption states:

Definition 2. *Let $\mathbb{G}$ be a cyclic group of order q and let g be a generator of $\mathbb{G}$. Let $a, b, c \leftarrow\$ \mathbb{Z}_q$ be sampled uniformly at random. Then, (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are computationally indistinguishable.*

Note that DDH is trivially solvable using Type-1 pairings ($\mathbb{G}_1 = \mathbb{G}_2$), since given g^a, g^b, and g^x where $x = ab$ or $x = c$ and $g \in \mathbb{G}_1$, one can easily check if $e(g^a, g^b) = e(g^x, g)$. DDH may also be solved using Type-2 pairings, in which an isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ is known. In contrast, DDH can still be assumed computationally hard in $\mathbb{G}_1$ and $\mathbb{G}_2$ when using Type-3 pairings.

We also make use of the **Decisional Bilinear Diffie-Hellman (DBDH) in Type-3** assumption, which is defined as follows:

Definition 3. *Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be cyclic groups of prime order q and let g_1 and g_2 generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Let $a, b, c \leftarrow\$ \mathbb{Z}_q$ and $g_T' \leftarrow\$ \mathbb{G}_T$ be sampled uniformly at random. Then, $(g_1^a, g_1^b, g_1^c, g_2^b, g_2^c, g_T')$ and $(g_1^a, g_1^b, g_1^c, g_2^b, g_2^c, e(g_1, g_2)^{abc})$ are computationally indistinguishable.*

By Theorem A.2 of [6], we see that the DDH and DBDH assumptions may be assumed to hold true in Type-3 pairings.

3.3 Random Oracle Model

Our work is secure in the **random oracle model**; we assume the existence of a random oracle O that behaves as follows.

Definition 4 (Random Oracle.). *Random Oracle $O : \{0,1\}^* \to \mathbb{G}$ is a public function. On input x, O returns a random value from the codomain. If input x is used again, the output is the same.*

3.4 Digital Signature Scheme

Definition 5. *A **digital signature scheme** consists of a tuple of algorithms* $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *with the following syntax:*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *takes as input a security parameter λ and outputs a public-private key pair* $(\mathsf{pk}, \mathsf{sk})$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ *takes as input a secret key* sk *and message m, and outputs a signature σ.*
- $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \sigma, m)$ *takes as input a signature σ, a message m, and public key* pk *and outputs a bit $b \in \{1, 0\}$.*

Security. We require that the digital signature scheme be **unforgeable**. That is, an adversary without the secret key should not be able to generate a valid signature on a message m not previously signed, with more than negligible probability.

The BLS Signature Scheme. Our protocols use the signature scheme by Boneh–Lynn–Shacham (BLS) [7] to authorize the client's elements. We describe the BLS signature scheme in detail below.

Let $H_1 : \{0,1\}^* \to \mathbb{G}_1$ denote a hash function modelled as a random oracle and let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be generators of their respective groups. To generate the key pair, the judge samples a secret key $\mathsf{sk} \leftarrow\$ \mathbb{Z}_q$ and sets the public key to $\mathsf{pk} \leftarrow g_2^{\mathsf{sk}}$ and shares it with the client.

To authorize an element $x \in \{0,1\}^*$ for client $\mathbf{C}$, the judge computes $\sigma \leftarrow H_1(x \| \mathtt{ID}_C)^{\mathsf{sk}}$ where $\mathtt{ID}_C$ denotes a unique identifier of the client. As in [38], we concatenate x with the client's ID to bind the authorization of x to the party.

To verify, that σ is a signature on x for $\mathbf{C}$, one can check that the following equality holds:

$$e(\sigma, g_2) = e(H_1(x \| \mathtt{ID}_C), \mathsf{pk}).$$

3.5 Arguments of Knowledge

In this work, we utilize a Sigma protocol. Intuitively, we can use a sigma protocol to convince a judge that a set of PRF values has been correctly computed under the same key.

Definition 6 (Effective relation). *An effective relation is a binary relation $\mathcal{R}$ that is the Cartesian product of 2 efficiently recognizable finite sets. If $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, then $\mathbf{w}$ is a witness for $\mathbf{x}$.*

Definition 7 (Interactive Arguments of Knowledge). *Let the triple of algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be defined as follows.*

- Setup *takes as input security parameter λ and a binary relation $\mathcal{R}$ and outputs public parameters* p.
- Prove *and* Verify *are interactive algorithms.* Prove *takes as input the public parameters* p *and a pair* $(\mathbf{x}, \mathbf{w})$*, and* Verify *takes as input* p *and* $\mathbf{x}$*. Let* $\mathsf{tr} \leftarrow \langle \mathsf{Prove}(\mathsf{p}, (\mathbf{x}, \mathbf{w})), \mathsf{Verify}(\mathsf{p}, \mathbf{x}) \rangle$ *denote the transcript of the interaction. At the end of the interaction,* Verify *outputs a bit $b = \langle \mathsf{Prove}(\mathsf{p}, (\mathbf{x}, \mathbf{w})), \mathsf{Verify}(\mathsf{p}, \mathbf{x}) \rangle$. If the transcript is accepted, then $b = 1$; otherwise, $b = 0$.*

The triple $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is an interactive argument of knowledge if the following two properties hold.*

- **Completeness:** *For honest prover $\mathbf{P}$ and verifier $\mathbf{V}$, $\forall (\mathbf{x}, \mathbf{w}) \in \mathcal{R}$:*

$$\Pr[\langle \mathsf{Prove}(\mathsf{p}, (\mathbf{x}, \mathbf{w})), \mathsf{Verify}(\mathsf{p}, \mathbf{x}) \rangle = 1] = 1$$

- **Honest-verifier zero knowledge (HVZK):** *For all $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, there exists an efficient algorithm $\mathcal{S}$, such that the transcript from*

an honest prover, and verifier is computationally indistinguishable from $\mathcal{S}(\mathsf{p}, \mathbf{x})$*:*

$$\{\langle \mathsf{Prove}(\mathsf{p}, (\mathbf{x}, \mathbf{w})), \mathsf{Verify}(\mathsf{p}, \mathbf{x})\rangle\} \overset{c}{\approx} \{\mathcal{S}(\mathsf{p}, \mathbf{x})\}$$

- **Knowledge Soundness:** *There exists an efficient extractor* $\mathcal{E}$ *such that given two accepting transcripts* T_0, T_1 *for* $\mathbf{w}$*, it can extract* $\mathbf{x}$*, where* $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$*.*

$$\mathbf{x} \leftarrow \mathcal{E}(T_0, T_1, \mathbf{w})$$

Using the Fiat-Shamir heuristic [17], we can turn a public coin interactive protocol into a non-interactive protocol in the random oracle model (ROM). In this case, the Setup algorithm also outputs a description of the hash function.

Definition 8. *An* ***exponent equality argument*** $\mathsf{EEA} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *is a zero-knowledge argument of knowledge for the following relation:*

$$\mathbf{x} = (\{w_i\}_{i\in[n]}\{h_i\}_{i\in[n]});\ \mathbf{w} = r:\ \ \forall i \in [n],\ h_i = H(w_i)^r$$

Details of the EEA protocol used can be found in Appendix A.1.

3.6 Oblivious PRFs

An oblivious pseudo-random function (OPRF) is a 2-party protocol in which the sender inputs a secret key k and the receiver inputs values $x_1, \ldots, x_m$. The receiver learns the output $f(k, x_1), \ldots, f(k, x_m)$ for some function f.

Definition 9 ([18]). *An* ***oblivious pseudo-random function*** *(OPRF) is a 2-party protocol* OPRF *with the following API:*

- $t, \mathcal{B} \leftarrow \mathsf{Request}(\mathcal{M})$ *takes as input a set* $\mathcal{M}$ *of messages, and outputs a blinding element* t *and a set of blinded elements* $\mathcal{B}$*.*
- $C \leftarrow \mathsf{Eval}(k, \mathcal{B})$ *takes as input a key* k*, blinded set* $\mathcal{B}$*, and outputs the set* C*.*
- $\mathcal{D} \leftarrow \mathsf{Recover}(C, t)$ *takes as input set* C *and blinding factor* t*, and outputs the set of PRF evaluations* $\mathcal{D}$*.*

Details of the OPRF instantiation can be found in Appendix A.2.

4 Our APSI Protocol

In this section we present our APSI protocol based on bilinear pairings and which is inspired by [38]. We emphasize, however, that the protocol in [38] encodes the elements and their signatures, and then uses these encodings as input to any black-box PSI protocol. This results in 3 rounds of communication beyond what is required by the underlying PSI protocol; in addition to signing the client's elements, fresh randomness must be exchanged by both **C** and **S** each time prior to running the black-box PSI protocol. In contrast, we are able to both authorize the client's elements and compute the intersection in 3 rounds, whilst still achieving malicious security against the client and better communication complexity.

4.1 Protocol Overview

The protocol proceeds in two phases. In the authorization phase, the client interacts with the judge to authorize its elements. The client sends its set $\mathcal{X}$ to the trusted third-party judge. If the judge approves $\mathcal{X}$, then it issues a signature on each element which the client can later use during the intersection phase.

Public parameters. Set sizes $n = |\mathcal{X}|$ and $m = |\mathcal{Y}|$; Bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order q with generators g_1, g_2, and g_T, respectively, and pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; Hash function $H_1 : \{0,1\}^* \to \mathbb{G}_1$ modeled as a random oracle.

Authorize. Authorization proceeds in 2 rounds. The client **C** sends its set $\mathcal{X} = \{x_i\}_{i\in[n]}$ to the judge **J**. If **J** approves $\mathcal{X}$, then for all $i \in [n]$, it computes the BLS signature $\sigma_i \leftarrow H_1(x_i \| \mathtt{ID}_C)^{\mathsf{sk}}$ and returns the signatures $\{\sigma_i\}_{i\in[n]}$ to **C**.

Intersect. The intersect phase is carried out in 1 round and can be carried out on input of any subset of $\mathcal{X}$. Upon receiving an intersection request from the client, **S** samples a random element $s \leftarrow\!\!\$\, \mathbb{Z}_q$ and computes the group element $S \leftarrow g_2^s$; this will be used to blind the server's elements. The server then encodes its elements as $\hat{y}_j \leftarrow e(H_1(y_j \| \mathtt{ID}_C), \mathsf{pk}^s)$ for each $j \in [m]$ and sends S and $\{\hat{y}_j\}_{j\in[m]}$ to the client.

Importantly, since the client does not know s or sk, it cannot generate the correct encoding of an element for which it does not have a valid signature. The client uses S to encode its elements as $\hat{x}_i \leftarrow e(\sigma_i, S)$ for each $i \in \mathcal{N} \subseteq [n]$. It recovers the intersection by outputting the x_i's such that $\hat{x}_i = \hat{y}_j$ for some $i \in \mathcal{N}$ and $j \in [m]$.

The detailed APSI protocol can be found in Figure 2.

4.2 Correctness and Security

Theorem 1. *Let* $\mathcal{X}$ *and* $\mathcal{Y}$ *be the inputs from the client* **C** *and server* **S***, respectively. Let* $\mathcal{X}_{AUTH}$ *be the client's elements authorized by the judge. Then, the APSI protocol (Figure 2) outputs* $(\mathcal{X}_{AUTH} \cap \mathcal{X}) \cap \mathcal{Y}$*, assuming unforgeability of the BLS signature scheme in the ROM.*

The proof can be found in Appendix B.1.

Theorem 2. *The APSI protocol (Figure 2) is secure against a malicious client* **C***, if the Decisional Bilinear Diffie-Hellman (DBDH) assumption holds and the hash function* H *is a random oracle.*

Proof. We start by describing our simulator, the first part of which simulates the judge **J**. The simulator starts by choosing a signing key pair (sk, pk) for the judge and gives pk to the adversary $\mathcal{A}$. The simulator proceeds as follows.

- **Hash query.** Honestly play the role of random oracle H_1 and construct a table T. Upon receiving a hash query q from the adversary $\mathcal{A}$, check to see if the query q has been previously issued. If yes, return the same answer stored in the table, $T[q]$. If not, then sample a group element $h \leftarrow\!\!\$\, \mathbb{G}_1$, return h, and update the table $T[q] \leftarrow h$.
- **Authorize.** When the adversary $\mathcal{A}$ requests the authorization of x for a corrupt client **C** with identifier $\mathtt{ID}_C$, make a hash query to the random oracle on $x \| \mathtt{ID}_C$ and update table T. After determining the hash output, compute the signature using sk and return it to $\mathcal{A}$.

The remainder of the simulator, simulates the server **S** for iteration $k \in [q]$ and on inputs $\mathcal{X}_k$ and $\mathcal{Y}_k$ from the client and server, respectively.

- **Set Intersection.** For each $x \in \mathcal{X}_k$, that $\mathcal{A}$ sent **J** and its corresponding signature σ do the following:
  - Sample $s \leftarrow\!\!\$\, \mathbb{Z}_q$ and compute $S = g_2^s$.

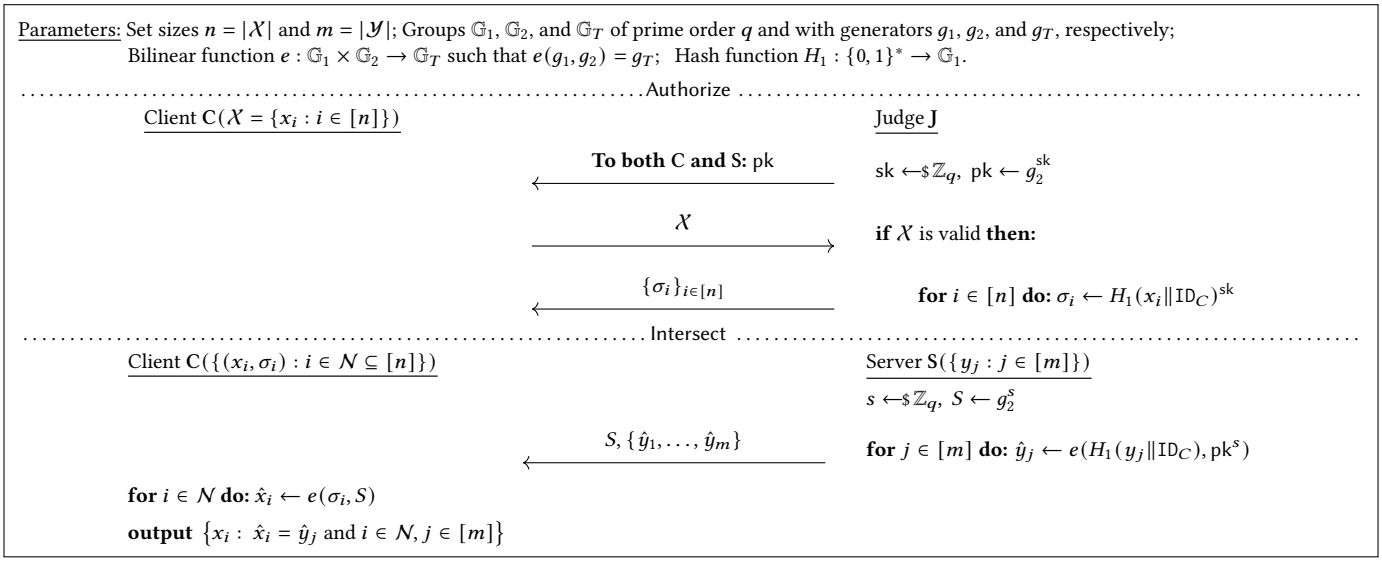


Figure 2: The APSI Protocol.

- Submit the set of elements in $\mathcal{X}_k$ that were received in the clear during authorization, i.e., $\mathcal{X} \cap \mathcal{X}_k$, to the ideal functionality (the functionality also receives $\mathcal{Y}_k$ from the honest server).
- Upon receiving the intersection $\mathcal{Z} = (\mathcal{X} \cap \mathcal{X}_k) \cap \mathcal{Y}_k$ from the functionality, compute $\hat{z} \leftarrow e(H_1(z\|\mathtt{ID}_C), \mathsf{pk}^s)$. Then pad $\mathcal{Z}$ up to size m with randomly sampled values from $\mathbb{G}_T$.
- Send $\mathcal{Z}$ to the client.

We now describe a list of hybrids that we will use to demonstrate indistinguishably between the real and ideal worlds.

Hybrid 0: The real interaction, where all parties run the protocol honestly on input $\mathcal{X}$ in Authorize and on $\mathcal{X}, \mathcal{X}_k$ and $\mathcal{Y}_k$ for all $k \in [q]$ in Intersect.

Hybrid 1: Compute $\mathsf{sk} \leftarrow\$ \mathbb{Z}_q$ and $\mathsf{pk} \leftarrow g_2^{\mathsf{sk}}$ to simulate a public/private key-pair (pk, sk). For each $x \in \mathcal{X}$, compute the signature σ of x using sk. Since sk is sampled uniformly at random, then the simulated key pair is distributed identically to those generated by the judge and thus Hybrids 0 and 1 are computationally indistinguishable.

Hybrid 2: Same as Hybrid 1, except the interaction aborts before the signatures are sent, if there are $x, x' \in \mathcal{X}$ such that $x\|\mathtt{ID}_C \notin T$ and $x'\|\mathtt{ID}_C \in T$ and yet querying $x\|\mathtt{ID}_C$ to H_1 results in the output $T[x'\|\mathtt{ID}_C]$. For some fixed item $x \neq x'$, the probability of such a collision is $1/|\mathbb{G}_1|$. Union bounding over all possible x we get a total collision probability of $n^2/|\mathbb{G}_1|$, which is negligible. Thus, Hybrids 1 and 2 are computationally indistinguishable.

Hybrid (3, k): Same as Hybrid 2 for $k = 0$ and the same as Hybrid $(3, k-1)$ otherwise, except we replace s_k with a randomly sampled value from $\mathbb{Z}_q$ and compute $S_k \leftarrow g_2^{s_k}$. Values s_k and S_k are identically distributed to those in the previous hybrid, and thus this hybrid and the previous one are indistinguishable.

Hybrid (4, k) for $k \in [q]$: Same as previous hybrid, except for $y \in \mathcal{Y}_k$ such that $y \notin \mathcal{X} \cap \mathcal{X}_k$, replace the value $\hat{y} = e(H_1(y\|\mathtt{ID}_C), \mathsf{pk}^s)$ with a random element $\hat{z} \leftarrow\$ \mathbb{G}_T$. We define the DBDH tuple

$$(H_1(y\|\mathtt{ID}_C),\ g_1^{\mathsf{sk}},\ g_1^s,\ g_2^{\mathsf{sk}},\ g_2^s,\ \hat{y} = e(H_1(y\|\mathtt{ID}_C), \mathsf{pk}^s)).$$

Replacing the final entry of the tuple with a random $\hat{z}$ is computationally indistinguishable by the DBDH assumption. Thus, Hybrid $(4, k)$ is indistinguishable from the previous hybrid.

Hybrid $(4, q)$ defines the simulator and the theorem follows. □

Theorem 3. *The APSI protocol (Figure 2) is secure against a semi-honest server* S.

Since the server only receives a public key from an honest judge and receives no input from the client, simulation of the interaction with the server is trivial. It is sufficient to argue correctness, which follows from Theorem 1.

5 Our Partial-APSI Protocol

We now present our Partial-APSI protocol, which extends our APSI protocol to support partially revealing the client's set to the judge. The main difference between the APSI and Partial-APSI problems is that the judge can only see a subset of $\mathcal{X}$, yet, must sign all the elements. In our Partial-APSI protocol, the client first blinds each element by raising its hash to a random field element r and sends the blinded values to the judge. The judge then requests for a fraction of the blinded elements to be revealed. If these elements are valid, then the judge signs all the blinded values. During the intersection phase, the server's elements must be intersected with the client's blinded values. The client and server engage in an OPRF protocol so that the client can "blind" the server's elements using the same r. Importantly, the client uses an OPRF to avoid sharing r.

5.1 Protocol Overview

Partial-authorization requires that the client commits to the elements in $\mathcal{X}$ and sends them to the semi-honest judge, who then challenges the client to reveal a subset of its elements. The client sends over the requested items in plaintext, together with an EEA proof that the requested items correspond to their respective commitments. If the judge approves the revealed items and the EEA

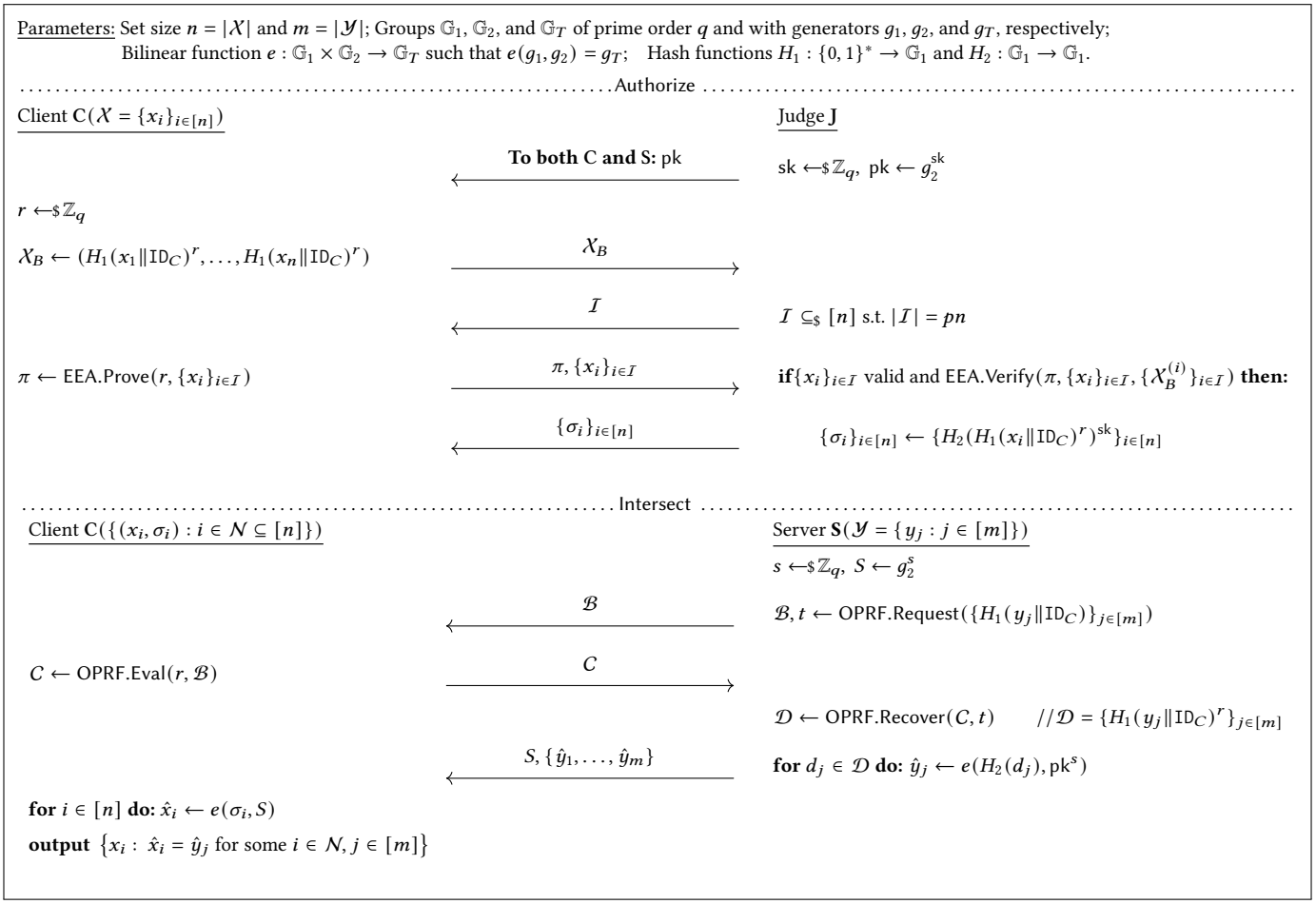


Figure 3: The Partial-APSI Protocol.

proof is valid, then **J** computes a set of BLS signatures on all the blinded items and returns them to the client. The client can then use these signatures during the intersection phase. At the end of the k-th iteration of Intersect, the client learns $(\mathcal{X} \cap \mathcal{X}_k) \cap \mathcal{Y}$. As in APSI, the server learns nothing.

Public parameters. Set sizes $n = |\mathcal{X}|$ and $m = |\mathcal{Y}|$; Bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order q with generators g_1, g_2, and g_T, respectively, and pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; Hash functions $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \mathbb{G}_1 \to \mathbb{G}_1$ modeled as a random oracles.

Authorize. The client starts by first sampling randomness r and then committing to each element x_i as $H_1(x_i\|\mathtt{ID}_C)^r$; we denote the set of blinded $\mathcal{X}$ values with $\mathcal{X}_B$. Blinding the terms with r ensures that the judge learns nothing about the client's values until they are revealed. The judge samples a set of indices $\mathcal{I} \subseteq_\$ [n]$ and sends it to the client; $\mathcal{I}$ specifies the indices of the client's elements that the client is challenged on and which should be revealed to the judge. The client computes an EEA proof π relating the plaintexts $\{x_i\}_{i\in\mathcal{I}}$ with the blinded commitments $\{H_1(x_i\|\mathtt{ID}_C)^r\}_{i\in\mathcal{I}}$, and sends π and $\{x_i\}_{i\in\mathcal{I}}$ to the judge. If the plaintext values are approved and the proof is valid, then the judge computes the BLS signature $\sigma_i \leftarrow H_2(H_1(x_i\|\mathtt{ID}_C))^{\mathsf{sk}}$ for all $i \in [n]$. Finally the judge returns all the signatures to the client, which completes partial-authorization.

Intersect. As before, the server starts by sampling randomness s. From here, the protocol diverges from the APSI protocol. In order for the server **S** to compute $H_1(y_j\|\mathtt{ID}_C)^r$ for $j \in [m]$ using the client's secret r, **S** must engage in an OPRF protocol with **C**. The client sends the OPRF evaluation C to the server, from which the server can recover $\mathcal{D} = \{d_j = H_1(y_j\|\mathtt{ID}_C)^r\}_{j\in[m]}$. The server encodes its elements using the randomness s as $\hat{y}_j \leftarrow e(H_2(d_j), \mathsf{pk}^s)$ for all $d_j \in \mathcal{D}$. Finally it sends the value $S \leftarrow g_2^s$ and the $\hat{y}_j$'s to the client. The client finally computes $\hat{x}_i \leftarrow e(\sigma_i, S)$ for $i \in \mathcal{N} \subseteq [n]$ and outputs the set $\{x_i : \hat{x}_i = \hat{y}_j \text{ for } i \in \mathcal{N}, j \in [m]\}$.

The OPRF is run on the server's elements (to exponentiate them using the secret r without the server learning r and the client learning anything about the server's set). If the server samples a new s for every run, then the $\hat{y}_j$'s received by the client look random, thus ensuring multi-run security. The server never receives the client's set and cannot infer anything about the client's elements.

The pseudocode of the protocol can be found in Figure 2.

5.2 Correctness and Security

Theorem 4. *Let $\mathcal{X}$ and $\mathcal{Y}$ be the inputs from the client* C *and server* S*, respectively. Let $\mathcal{X}_{AUTH}$ be the client's elements authorized by the judge. Then, the Partial-APSI protocol (Figure 3) outputs $(\mathcal{X} \cap \mathcal{X}_{AUTH}) \cap \mathcal{Y}$, assuming unforgeability of the BLS signature scheme in the ROM.*

The proof can be found in Appendix B.2.

Theorem 5. *The Partial-APSI protocol (Figure 3) is secure against a semi-honest Judge, if the DDH assumption holds in the bilinear group $\mathbb{G}_1$ and the hash function H_1 is a random oracle.*

The proofs can be found in Appendix B.3.

Theorem 6. *The Partial-APSI protocol (Figure 3) is secure against an input-malicious client* C*, if the Decisional Bilinear Diffie-Hellman (DBDH) holds, the Decisional Diffie-Hellman (DDH) assumption holds in group $\mathbb{G}_1$, and hash functions H_1 and H_2 are random oracles.*

Proof. Once again, we prove the security through a series of hybrid arguments. Upon input of $\mathcal{Y}$ and $\mathcal{Y} \cap \mathcal{X}$, the simulator chooses a signing key pair (sk, pk) for the judge, gives pk to the adversary $\mathcal{A}$, and does the following:

- **Hash queries:** Model the hash functions H_1 and H_2 as random oracles and store the results in tables T_1 and T_2, respectively. In particular, upon each query q to H_1, check if there exists hash value h such that $h = T_1[q]$ and return h. If not, then sample a random h, return h, and record $T_1[x] \leftarrow h$. Similarly for H_2.
- **Authorize.** To simulate the judge **J** it does the following.
  - Upon receiving an authorization request from the client and the committed values $(x'_1, \ldots, x'_n)$, sample a set of indices $\mathcal{I}' \subseteq_\$ [n]$ and send it to **C**.
  - Check that π and $\{x_i\}_{i \in \mathcal{I}}$ are valid. If not, then abort. If yes, then make a hash query to the random oracle H_2 for each x'_i, $i \in [n]$ (T_2 is updated accordingly). If for any query q to H_2 and its corresponding hash value h, there exists a $q' \neq q$ such that $T_2[q'] = h$, then abort. As before, this happens with negligible probability. If all hash queries are successful, then compute the signatures and return them to **C**.

On iteration $k \in [q]$ and inputs $\mathcal{X}_k$ and $\mathcal{Y}_k$ from the client and server, respectively, do the following:

- **Set intersection.** To simulate the server S it does the following.
  - Upon receiving an intersection request from **C**, sample a new $s^{(k)} \leftarrow_\$ \mathbb{Z}_q$.
  - Run the OPRF protocol with **C**. In particular, for each $y \in (\mathcal{X} \cap \mathcal{X}_k) \cap \mathcal{Y}_k$, query $y \| \mathtt{ID}_C$ to H_1 and run the OPRF on the value $H_1(y \| \mathtt{ID}_C)$. If for any hash query q to H_1 and its corresponding value h, there exists a $q' \neq q$ such that $T_2[q'] = h$, then abort. This happens with negligible probability.
  - For the remaining $|\mathcal{Y}_k \setminus (\mathcal{X} \cap \mathcal{X}_k) \cap \mathcal{Y}_k|$ items, sample a random value $y' \in \{0,1\}^*$ and run the OPRF on y'.
  - For each $c_j \in C$ where C is the output of the OPRF evaluation, make a hash query to H_2 on c_j and compute $\hat{y}_j \leftarrow e(H_2(c_j), \mathrm{pk}^{s^{(k)}})$. If for any hash query q to H_2 and its corresponding value h, there exists a $q' \neq q$ such that $T_2[q'] = h$, then abort. This happens with negligible probability. Finally return $S = g_2^{s^{(k)}}$ and the $\hat{y}_j$'s to the **C**.

We now consider the following series of hybrids:

Hybrid 0: The real interaction, where all parties run the protocol honestly on input $\mathcal{X}$ in Authorize and on $\mathcal{X}, \mathcal{X}_k$ and $\mathcal{Y}_k$ for all $k \in [q]$ in Intersect.

Hybrid 1: Same as Hybrid 0, but $\mathcal{I}$ is chosen randomly by the simulator and then sent to **C**. Hybrid 1 is indistinguishable from Hybrid 0, since $\mathcal{I}$ is sampled from the same distribution.

Hybrid (2, k) for $k \in [q]$: Same as previous hybrid, except we abort if there exist $y^* \in \mathcal{Y}_k$ and $x^* \in \mathcal{X}_k \setminus \mathcal{Y}_k$ such that $H_1(x^* \| \mathtt{ID}_C) = H_1(y^* \| \mathtt{ID}_C)$. This event happens with probability $nm/|\mathbb{G}_1|$ which is negligible.

Hybrid $(3, k)$ **for** $k \in [q]$**:** Same as previous hybrid, except we abort if there exist elements $y^* \in \mathcal{Y}_k$ and $x^* \in \mathcal{X}_k \setminus \mathcal{Y}_k$ such that $H_2(H_1(x^* \| \mathtt{ID}_C)^r) = H_2(H_1(y^* \| \mathtt{ID}_C)^r)$ and $H_1(x^* \| \mathtt{ID}_C) \neq H_1(y^* \| \mathtt{ID}_C)$. This happens with probability $nm/|\mathbb{G}_1|$ which is negligible.

Hybrid $(4, k)$ **for** $k \in [q]$**:** Same as previous hybrid, but we replace the $H_1(y_j || \mathtt{ID}_C)^{t_k} \in \mathcal{B}$, where $y_j \in \mathcal{Y}_k \setminus (\mathcal{X} \cap \mathcal{X}_k)$, with random values from $\mathbb{G}_1$. Under the DDH assumption, the OPRF request $\mathcal{B}$ in Hybrid 3 is indistinguishable from the request in Hybrid 4 (Theorem 9). In particular, this can be carried out with only knowledge of $(\mathcal{X} \cap \mathcal{X}_k) \cap \mathcal{Y}_k$, $\mathcal{Y}_k$ and $|\mathcal{Y} \setminus (\mathcal{X} \cap \mathcal{X}_k)|$.

Hybrid $(5, k)$ **for** $k \in [q]$**:** Same as previous hybrid, but we replace the elements $\hat{y_j} \leftarrow e(H_2(H_1(y_j || ID_C)^r), pk^{s^{(k)}})$, where $y_j \in \mathcal{Y}_k \setminus (\mathcal{X} \cap \mathcal{X}_k)$ with random values $\hat{z_j} \leftarrow_\$ \mathbb{G}_T$. We define the DBDH tuple

$$(H_2(H_1(y_j || ID_C)^r), g_1^{\mathsf{sk}}, g_1^{s^{(k)}}, g_2^{\mathsf{sk}}, g_2^{s^{(k)}}, \hat{y} = e(H_2(H_1(y_j || ID_C)^r), \mathrm{pk}^{s^{(k)}})).$$

Replacing the final entry with a random $\hat{z_j}$ is computationally indistinguishable by the DBDH assumption. Thus, this hybrid is indistinguishable from the previous one.

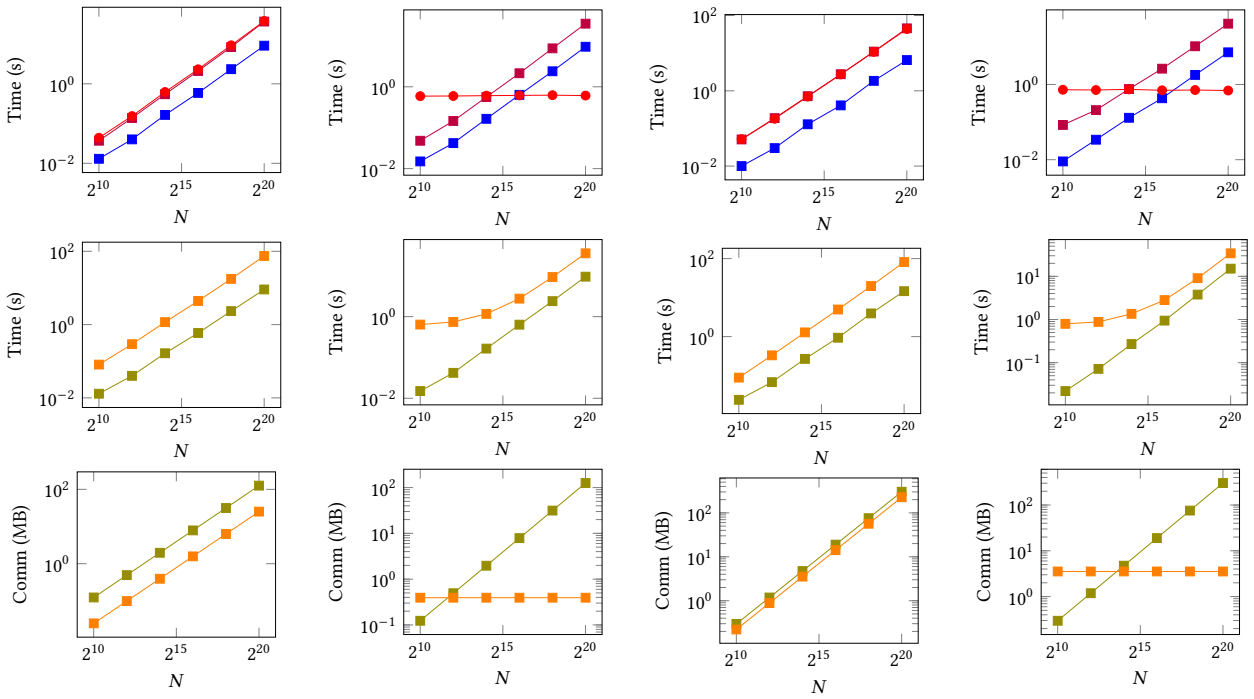
Hybrid $(5, q)$ is identical to our simulator and thus the Partial-APSI protocol is secure against an input-malicious client **C**. □

Theorem 7. *The Partial-APSI protocol (Figure 3) is secure against a semi-honest server* S*, if the DDH assumption holds in group $\mathbb{G}_1$ and hash functions H_1 and H_2 are random oracles.*

Proof. We prove the security through a series of hybrid arguments. Upon input of $\mathcal{X}$ the simulator does the following:

- **Hash queries:** Model the hash functions H_1 and H_2 using tables T_1 and T_2 as before.
- **Authorize.** To simulate **C** and **J**, $\mathcal{S}$ does the following.
  - Upon receiving an intersection request from S, sample $r \leftarrow_\$ \mathbb{Z}_q$.
  - Compute vector $\mathcal{X}_B$ as follows. For each $x_i \in \mathcal{X} \cap \mathcal{Y}$, make a hash query $x_i \| \mathtt{ID}_C$ (updating T_1 as needed) to H_1 and compute $\mathcal{X}_B^{(i)} \leftarrow H_1(x_i \| \mathtt{ID}_C)^t$; for all other indices, sample a random $x'_i \leftarrow_\$ \{0,1\}^*$ and compute $\mathcal{X}_B^{(i)} \leftarrow_\$ H_1(x'_i)^t$. If there is a collision for distinct q, q' to H_1 in table T_1, then abort.
  - For each $\mathcal{X}_B^{(i)}$, $i \in [n]$, make a hash query to random oracle H_2 (updating T_2 as needed) and compute the signatures (recall that the simulator knows the private-public key pair of the judge). If there is a collision between queries to H_2 in T_2, abort.

On iteration $k \in [q]$ and inputs $\mathcal{X}_k$ and $\mathcal{Y}_k$ from the client and server, respectively, do the following:



(a) **APSI:** $|\mathcal{X}| = |\mathcal{Y}| = N$.

(b) **APSI:** $|\mathcal{X}| = N, |\mathcal{Y}| = 2^{14}$

(c) **Partial-APSI:** $|\mathcal{X}| = |\mathcal{Y}| = N$ and $p = 0.20$

(d) **Partial-APSI:** $|\mathcal{X}| = N, |\mathcal{Y}| = 2^{14}$, and $p = 0.20$

Figure 4: Timing and communication results of our protocols run on commodity hardware: In the above graphs, — refers to the judge, — refers to the client and — refers to the server. For the phases, authorize is — and intersect is —.

- **Intersect.** Upon input of $\mathcal{X}_k$ and $(\mathcal{X} \cap \mathcal{X}_k) \cap \mathcal{Y}_k$ the simulator does the following.
  - Upon receipt of set $\mathcal{B}$, run the OPRF evaluation on r and $\mathcal{B}$ to obtain C, and send C to the server S.

We now consider the following series of hybrids:

Hybrid 0: The real interaction, where all parties run the protocol honestly on input $\mathcal{X}$ in Authorize and on $\mathcal{X}, \mathcal{X}_k$ and $\mathcal{Y}_k$ for all $k \in [q]$ in Intersect.

Hybrid 1: Compute $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_q$ and $\mathsf{pk} \leftarrow g_2^{\mathsf{sk}}$ to simulate a public/private key-pair $(\mathsf{pk}, \mathsf{sk})$. For each $x_i \in \mathcal{X}$, compute the signature σ_i of x_i using sk. Since sk is sampled uniformly at random, then the simulated key pair is indistinguishable from those generated by the judge, and Hybrids 0 and 1 are thus indistinguishable.

Hybrid $(2, k)$ for all $k \in [q]$: Same as previous hybrid, but r is replaced by $r' \leftarrow\!\!\$\, \mathbb{Z}_q$. By the DDH assumption, $H_1(y_j||\mathtt{ID}_C)^{tr'}, y_j \in \mathcal{Y}_k$, from $H_1(y_j||\mathtt{ID}_C)^{tr}$. Thus, the two sequential hybrids are indistinguishable.

Hybrid $(3, k)$ for all $k \in [q]$: Same as previous hybrid, except we abort if there exist $y^* \in \mathcal{Y}_k$ and $x^* \in \mathcal{X}_k \setminus \mathcal{Y}_k$ such that $H_1(x^*||\mathtt{ID}_C) = H_1(y^*||\mathtt{ID}_C)$. This happens with negligible probability, $nm/|\mathbb{G}_1|$, and thus the two hybrids are indistinguishable.

Hybrid $(4, k)$ for all $k \in [q]$: Same as previous hybrid, except we abort if there exist elements $y^* \in \mathcal{Y}_k$ and $x^* \in \mathcal{X}_k \setminus \mathcal{Y}_k$ such that $H_2(H_1(x^*||\mathtt{ID}_C)^r) = H_2(H_1(y^*||\mathtt{ID}_C)^r)$ and $H_1(x^*||\mathtt{ID}_C) \neq H_1(y^*||\mathtt{ID}_C)$. This happens with negligible probability, $nm/|\mathbb{G}_1|$, and so the hybrids are indistinguishable.

Hybrid $(4, q)$ is identical to our simulator. Since the simulator is indistinguishable from the real world, we conclude that the Partial-APSI protocol is secure against a semi-honest server S. □

One important property that is required to upgrade the scheme to malicious security is **key-consistency**, i.e., ensuring that the same key is used to blind/encrypt elements across both phases of the protocol. In the Partial-APSI protocol (Figure 3), we would need to ensure that the r used to blind its elements from the judge in Authorize is the same r used in the OPRF evaluation in the Intersect phase. We could achieve this by having the judge sign an additional element d during Authorize, which is then sent to S during Intersect. The server would then run the EEA protocol in Intersect with the client on the set $\mathcal{Y} \cup \{d\}$ so that it can be sure that the r in Authorize is consistent with the r used to exponentiate its elements in Intersect. To achieve malicious security, one must also take care to use a maliciously secure OPRF and we leave this for future work.

6 Evaluation

We implemented our two protocols in C++ and ran our experiments on an M2 MacBook Pro with 3.49 GHz CPU, 16GB RAM and 10 cores (results in Sections 6.1 and 6.2). We also ran our experiments on a compute cluster with a 2 x 28 Core Intel Xeon Gold 6258R 2.7GHz Processor (Turbo up to 4GHz / AVX512 Support), and 384GB DDR4 2933MHz ECC Memory, parallelized across 50

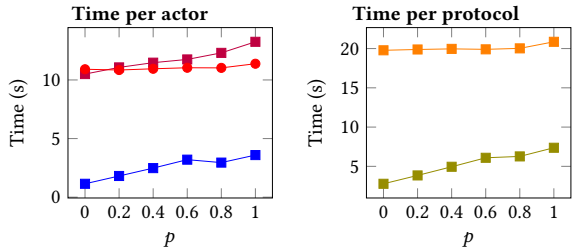


Figure 5: PAPSI: $|\mathcal{X}| = |\mathcal{Y}| = 2^{18}$ varying p; —■— refers to the judge, —■— the client and —●— the server. For the protocols, authorize is —■— and intersect is —■—.

cores to demonstrate parallelizability of our protocols (results in Section 6.3). For the pairing operations, we used the Pairing Based Cryptography Library [37] with the default Type-3 pairings.

For both the client and server set, we randomly generate credit card numbers to perform the intersection over. Our implementation can be found in https://anonymous.4open.science/r/APSI-451E.

6.1 Commodity Hardware

APSI. In Figure 4a, we plot the time and communication needed for the APSI protocol when $|\mathcal{X}| = |\mathcal{Y}|$. On the right, we see that the intersect phase takes more time than the authorize phase due to the intersection requiring the computation of a bilinear pairing for each element, which is our most expensive operation. We observe the same trend on the left, where we plot the time needed per actor. The client and the server need more time than the judge since the judge only computes one hash function evaluation and one exponentiation per element. Communication grows linearly with N, with Intersect requiring overall less bandwidth.

In Figure 4b, we plot the time and communication needed for the APSI protocol when $|\mathcal{X}|$ varies while $|\mathcal{Y}|$ is fixed to 2^{14} elements. On the left, we observe the time needed per actor. As expected, the server takes the same amount of time, regardless of the size of the client set. With regards to the different phases, the authorize time increases as the client's set grows. The time required for the intersect is initially dominated by the server's elements and stable, but when the client set grows larger than the server set, the intersect time also grows. Bandwidth of Authorize grows linearly since it is a function of $N = |\mathcal{X}|$, whereas the bandwidth of Intersect is only dependent on $|\mathcal{Y}|$, which is fixed, and hence remains constant.

Partial-APSI. In Figure 4c, we plot the time and communication needed for the Partial-APSI protocol when $|\mathcal{X}| = |\mathcal{Y}|$ and $p = 0.20$. The timing follows a similar trend as Figure 4a. There is a bigger gap between the time the server and the client need, versus the time required by the judge. This is due to the client blinding their elements in the authorize phase and running the OPRF protocol in the intersect phase. However, the bilinear pairings are the most expensive operation and the added operations (i.e., for the EEA and OPRF protocols) have minimal impact on the timing. Communication also follows a similar linear trend as in 4a; note that while authorization requires similar communication overhead to that of the APSI variant, the intersection phase requires notably more communication.

In Figure 4d, we plot the time and communication needed for the Partial-APSI protocol when $|\mathcal{X}|$ varies, $|\mathcal{Y}|$ is fixed to 2^{14} elements, and $p = 0.20$. Again, Partial-APSI follows similar trends to APSI (Figure 4b) with the server's time being independent of the client's set size, and both the client and the judge requiring time proportional to the size of $\mathcal{X}$. The communication of Authorize increases linearly with $N = |\mathcal{X}|$. In contrast, the intersection phase communication remains constant since it only relies on $|\mathcal{Y}|$ and $|\mathcal{Y}|$ is fixed. The increase in communication of Intersect of Partial-APSI over APSI is a result of the values sent for the OPRF.

In Figure 5, we fix $|\mathcal{X}| = |\mathcal{Y}| = 2^{18}$ and explore the impact of varying p, the percentage of elements that the client and judge apply EEA on. The server's time and the time it takes to run the intersect protocol are not affected by p. However, the authorize phase times (correspondingly, the judge and client times) increase as p increases.

6.2 Comparison with Related Work

In Table 3, we compare our construction with the APSI protocol DT10 [14]. Overall, we see that despite the higher communication of our protocol for most data set sizes tested (due to the larger size of pairings), our protocol out performs DT10 with respect to both computational runtime and total runtime (which takes into account both bandwidth and network latency). In addition to our protocol's individual computations being generally faster than those of DT10, our protocol is also highly parallelizable thus resulting in overall faster runtimes. In particular, all computations in Authorize and all computations in Intersect except the final intersection computation can be parallelized. In contrast, some steps of DT10's protocol cannot easily be parallelized.

DT10's protocol is most efficient for larger m and smaller n. We observe this for $n = 2^{10}$ and $m = 2^{16}$, where in the 50Mbps and 5Mbps setting, DT10 outperforms our protocol. Unfortunately, due to RAM limitations, we were not able to run DT10 for larger values of n and m. As such, we weren't able to observe the asymptotic communication savings of our protocol overtake that of DT10 for sufficiently large sets. In all other network settings we experimented with, our protocol outperformed DT10.

6.3 Compute Cluster

In Tables 4 and 5 we report the communication and computation runtime results of running our protocols on a compute cluster across 50 cores. Our protocols are both highly parallelizable, and running them across 50 cores (vs on a laptop) results in runtimes of an order of magnitude less than the experiments on commodity hardware. We also report the total runtime of our protocol. The LAN network has 20 Gbps bandwidth and 0.1 ms RTT latency. All the other network settings have 80 ms RTT.

For both protocols, we see that the Authorize phase requires substantially more communication than the Intersect phase. This is because the size of the signature (96 bytes) is much larger than the output of the pairing operation (32 bytes). We believe that, in practice, this trade-off is reasonable as both of our protocols achieve **unlinkability**: the intersection phase can be instantiated multiple times using the same set of signatures without leaking any additional information.

Set Sizes		Scheme	Comm. (KB)		Comp. Runtime (ms)			Total Runtime (s)			
n	m		Auth	Inter	J	C (Inter)	S	LAN	200Mbps	50Mbps	5Mbps
2^{10}	2^8	DT10	41.0	57.4	1	1777	175	1.95	2.12	2.13	2.27
		Ours	123	6.34	16	40	15	0.07	0.4	0.41	0.6
	2^{12}	DT10	41.0	57.4	1	1809	225	2.04	2.2	2.21	2.35
		Ours	123	98.5	11	37	151	0.2	0.53	0.55	0.87
	2^{16}	DT10	41.0	57.4	1	1810	898	2.71	2.87	2.88	3.03
		Ours	123	1573	11	71	2354	2.44	2.82	3.03	5.47
	2^{20}	DT10	-	-	-	-	-	-	-	-	-
		Ours	123	25166	12	735	38048	38.81	40.13	43.16	79.58
2^{12}	2^8	DT10	164	229	1	9707	8218	17.93	18.1	18.15	18.72
		Ours	492	6.34	48	136	11	0.2	0.53	0.59	1.31
	2^{12}	DT10	164	229	1	9844	8312	18.16	18.33	18.38	18.95
		Ours	492	98.5	41	157	1569	0.35	0.7	0.77	1.62
	2^{16}	DT10	164	229	1	9697	8941	18.64	18.81	18.86	19.43
		Ours	492	1573.056	41	195	2346	2.58	2.98	3.23	6.21
	2^{20}	DT10	-	-	-	-	-	-	-	-	-
		Ours	492	25166	39	920	38679	39.65	40.98	44.06	81.01

Table 3: Results of running our APSI protocol and DT10 [14] on commodity hardware (M2 Macbook). We were unable to run the APSI protocol from DT10 on with $n = 2^{14}$ due to memory constraints (note that although the values sent between the client and the server are small hash outputs, the values prior to hashing are large group elements).

Set Sizes		Comm. (MB)		Computation Runtime (ms)			Total Runtime (s)			
n	m	Auth	Inter	J	C (Inter)	S	LAN	200Mbps	50Mbps	5Mbps
2^{12}	2^{16}	0.525	2.10	61	117	698	0.88	1.3	1.62	5.39
	2^{20}	0.525	33.6	55	1138	11633	12.84	14.51	18.6	67.67
	2^{24}	0.525	537	69	28110	213058	241.45	263.05	327.54	1101.39
2^{16}	2^{16}	8.39	2.10	299	822	744	1.87	2.6	3.86	18.96
	2^{20}	8.39	33.6	292	1762	11655	13.73	15.71	20.74	81.14
	2^{24}	8.39	537	220	21044	182029	203.51	225.42	290.85	1076.03
2^{20}	2^{16}	134	2.10	2973	11005	779	14.81	20.53	36.89	233.18
	2^{20}	134	33.6	2764	12374	11743	26.95	33.91	54.04	295.64
	2^{24}	134	537	2741	31045	206295	240.35	267.24	347.78	1314.14
2^{24}	2^{16}	2147	2.10	43983	197836	777	243.46	328.9	586.85	3682.25
	2^{20}	2147	33.6	43308	193751	11194	249.13	335.81	597.54	3738.23
	2^{24}	2147	537	44490	204423	191963	441.95	548.57	870.69	4736.16

Table 4: Results of running APSI on a compute cluster, parallelized across 50 cores.

Set Sizes		Comm. (MB)		Computation Runtime (ms)				Total Runtime (s)			
n	m	Auth	Inter	J	C (Auth)	C (Inter)	S	LAN	200Mbps	50Mbps	5Mbps
2^{12}	2^{16}	1.18	14.7	34	75	156	962	1.23	2.5	4.4	27.25
	2^{20}	1.18	235	46	72	1566	13612	15.39	25.38	53.71	393.64
	2^{24}	1.18	3758	41	72	23322	217923	242.86	392.37	843.48	6256.84
2^{16}	2^{16}	18.9	14.7	144	335	793	907	2.19	4.16	8.19	56.55
	2^{20}	18.9	235	115	265	2188	12997	15.67	26.36	56.81	422.26
	2^{24}	18.9	3758	127	246	24559	208837	235.28	385.49	838.73	6277.61
2^{20}	2^{16}	302	14.7	1484	2866	11041	901	16.42	29.62	67.67	524.27
	2^{20}	302	235	1755	3037	12539	13253	30.8	52.72	117.19	890.89
	2^{24}	302	3758	1676	2837	34007	224301	264.45	425.88	913.14	6760.27
2^{24}	2^{16}	4839	14.7	22961	43189	185987	898	254.98	447.8	1030.19	8018.85
	2^{20}	4839	234	21823	42747	181584	12818	261.0	462.55	1071.36	8377.1
	2^{24}	4839	3758	23579	43193	209280	221013	500.5	841.57	1873.17	14252.34

Table 5: Results of running Partial-APSI ($p = 0.10$) on a compute cluster, parallelized across 50 cores.

7 Discussion

In this work, we revisit APSI and present a round-optimal APSI protocol built from bilinear pairings. We also propose a generalization of APSI, called Partial-APSI, and extend our APSI protocol to the Partial-APSI setting. We prove security and correctness, and implement both protocols to demonstrate their practicality.

Properties of (Partial-)APSI protocols. A PSI protocol is said to be **client-private** (resp. server-private) if the client (resp. server) doesn't learn anything about the other party's set other than an upper bound on the set size and – in the case of the client – the intersection. Since the client never sends anything to the server in either of our protocols, the server doesn't learn anything about $\mathcal{X}$ and thus both protocols achieve **full client privacy**.

Rational and Covert Adversaries. Modeling parties either as semi-honest or malicious does not capture the full picture of how people in the real world may act; often times people are driven by self-gain and may not deviate from the protocol if doing so offers little to no benefit. A game theoretic approach to APSI/Partial-APSI could shed light on how a **rational party** in a one-sided protocol would act. In [4], Aumann and Lindell also introduce the notion of a **covert adversary**. Such an adversary may deviate arbitrarily from the protocol, but does not wish to be "caught" cheating. The goal is to design a protocol that guarantees that any adversary deviating from the protocol can be detected by the honest parties with probability at least ϵ. It would be interesting to explore whether APSI/Partial-APSI secure against such adversaries can be designed and be made more efficient than existing constructions.

Parameter Selection. The value of p is dependent on the application and the level of risk that the server is willing to tolerate. Let n be the total number of items in $\mathcal{X}$ and let k be the number of malicious elements in $\mathcal{X}$. Then the probability that **J** samples a malicious element is $1 - \binom{n-k}{\lceil pn \rceil} / \binom{n}{\lceil pn \rceil}$. If $k = 1$, then the probability of being caught is $1 - (n - \lceil pn \rceil)/n$; this probability increases as k grows. If the server can tolerate the client learning a couple of malicious elements (elements that were never shown to the judge, but nevertheless signed), then $p \leq 1/2$ may be chosen.

In the case of a rational adversary, if the cost of getting caught is higher than the value of learning a few additional elements, then the adversary would be incentivized to not cheat even when p is small. We also note that standard PSI protocols may allow for an arbitrary number of maliciously added elements. In comparison, running Partial-APSI with $p \leq 1/4$ offers a significant deterrent to the client adding any number of malicious elements. For applications in which no error can be tolerated (e.g., such as a government organization computing the intersection of a no-fly list and the list of passengers passing through an airport), APSI may be best.

Challenges of Adapting PSI to (Partial-)APSI. As PSI protocols become more efficient, the techniques used become increasingly sophisticated. Modern PSI protocols often have multiple rounds of interaction and numerous cryptographic primitives composed in complex ways and with varying dependencies. Authorizing a subset of elements before computing the intersection is non-trivial and often places certain requirements on the protocol.

An APSI protocol needs to fulfill the following requirements: (1) only elements in $\mathcal{X}$ authorized by the judge appear in the intersection and (2) the server cannot extract anything about $\mathcal{X}$ beyond the size. If we consider Partial-APSI, we have additional requirements: (3) the elements revealed by the client during authorization indeed correspond to the indices requested by the judge and (4) no more than pn elements are revealed to the judge, either directly or by enabling the judge to extract the information (e.g., dictionary attacks). Below we outline some of the challenges of extending state-of-the-art PSI protocols to support authorization.

When a PSI protocol involves multiple rounds of interaction, it's possible that the only way to adapt it to APSI would be to involve the judge in the intersection protocol. This increases the load on the judge, requiring the judge to be online for the entire protocol.

In the case of partial authorization, a PSI protocol may utilize cryptographic building blocks such that the judge cannot guarantee the number of elements encoded, making requesting elements by index difficult or impossible. In other words, **C** could insert additional elements without the judge detecting the existence of these elements. For example, a number of PSI protocols utilize oblivious key-value stores (OKVS) [20], but many OKVSs do not necessarily leak $|\mathcal{X}|$. Some protocols make it impossible to even check if a specific element is in $\mathcal{X}$, like hashing into bins [25].

A PSI protocol may utilize cryptographic building blocks that cannot be "partially" opened. For example, PSI can be achieved using encryption. Revealing one element would require disclosing the secret key, which would compromise all the elements. In this case, the protocol would likely need to be augmented with potentially costly oblivious computation or a commitment scheme.

Future Directions. While APSI is ideal in settings in which a third party can authorize the sets (e.g., an accounting firm auditing the purchases made at a company), in some settings there is no clear answer for who the judge should be (e.g., private contact discovery). The question of how to mitigate the injection of elements to a party's set in such settings is an open question. The introduction of the Partial-APSI problem also opens up a number of future directions. One question is whether the computation time and communication complexity of Partial-APSI protocols can be improved. Another interesting direction would be to explore the intersection of APSI and data privacy law, for example, by extending the work in [38] and developing APSI schemes that allow fine-grained policy control and updates.

Acknowledgments

This work was supported by the European Commission under the Horizon Europe Programme as part of the project RECITALS (Grant Agreement no. 101168490), by Armasuisse Science and Technology, and by the U.S. National Science Foundation. This work was done in part while Francesca Falzon was at Brown University and at the University of Chicago, and while Evangelia Anna Markatou was at Brown University. The authors would like to thank Roberto Tamassia and Zachary Espiritu for introducing them to the problem and for their contributions to early stages of this work. The authors would also like to thank Peihan Miao, Anna Lysyanskaya and Kenneth G. Paterson for their helpful feedback and discussions.

References

[1] 2016. General Data Protection Regulation (GDPR). https://gdpr-info.eu/.

[2] Aydin Abadi, Sotirios Terzis, and Changyu Dong. 2016. VD-PSI: Verifiable Delegated Private Set Intersection on Outsourced Private Datasets. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 9603)*, Jens Grossklags and Bart Preneel (Eds.). Springer, 149–168. https://doi.org/10.1007/978-3-662-54970-4_9

[3] Aydin Abadi, Sotirios Terzis, Roberto Metere, and Changyu Dong. 2017. Efficient delegated private set intersection on outsourced private datasets. *IEEE Transactions on Dependable and Secure Computing* 16, 4 (2017), 608–624.

[4] Yonatan Aumann and Yehuda Lindell. 2007. Security against covert adversaries: efficient protocols for realistic adversaries. In *Proceedings of the 4th Conference on Theory of Cryptography* (Amsterdam, The Netherlands) *(TCC'07)*. Springer-Verlag, Berlin, Heidelberg, 137–156.

[5] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2011. Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (Chicago, Illinois, USA) *(CCS '11)*. Association for Computing Machinery, New York, NY, USA, 691–702.

[6] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. 2005. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3494)*, Ronald Cramer (Ed.). Springer, 440–456. https://doi.org/10.1007/11426639_26

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Advances in Cryptology — ASIACRYPT 2001*, Colin Boyd (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[8] Prasad Buddhavarapu, Benjamin M. Case, Logan Gore, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Min Xue. 2021. Multi-key Private Matching for Compute. *IACR Cryptol. ePrint Arch.* (2021), 770.

[9] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. 2020. Private Matching for Compute. *IACR Cryptol. ePrint Arch.* (2020), 599.

[10] Jan Camenisch and Gregory M. Zaverucha. 2009. Private Intersection of Certified Sets. In *Financial Cryptography and Data Security*, Roger Dingledine and Philippe Golle (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 108–127. https://doi.org/10.1007/978-3-642-03549-4_7

[11] Paolo D'Arco, María Isabel González Vasco, Angel L. Pérez del Pozo, and Claudio Soriente. 2012. Size-Hiding in Private Set Intersection: Existential Results and Constructions. In *Progress in Cryptology - AFRICACRYPT 2012*, Aikaterini Mitrokotsa and Serge Vaudenay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 378–394. https://doi.org/10.1007/978-3-642-31410-0_23

[12] Emiliano De Cristofaro, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. 2009. Privacy-Preserving Policy-Based Information Transfer. In *Privacy Enhancing Technologies*, Ian Goldberg and Mikhail J. Atallah (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 164–184.

[13] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. 2010. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 213–231. https://doi.org/10.1007/978-3-642-17373-8_13

[14] Emiliano De Cristofaro and Gene Tsudik. 2010. Practical Private Set Intersection Protocols with Linear Complexity. In *Financial Cryptography and Data Security*, Radu Sion (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 143–159. https://doi.org/10.1007/978-3-642-14577-3_13

[15] Sumit Kumar Debnath and Ratna Dutta. 2015. Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter. In *Information Security*, Javier Lopez and Chris J. Mitchell (Eds.). Springer International Publishing, Cham, 209–226. https://doi.org/10.1007/978-3-319-23318-5_12

[16] Sky Faber, Ronald Petrlic, and Gene Tsudik. 2015. UnLinked: Private Proximity-based Off-line OSN Interaction. In *WPES '15: Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. 121–131. https://doi.org/10.1145/2808138.2808149

[17] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings on Advances in Cryptology—CRYPTO '86* (Santa Barbara, California, USA). Springer-Verlag, Berlin, Heidelberg, 186–194.

[18] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *Theory of Cryptography*, Joe Kilian (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 303–324.

[19] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. 2008. Pairings for cryptographers. *Discret. Appl. Math.* 156, 16 (2008), 3113–3121. https://doi.org/10.1016/J.DAM.2007.12.010

[20] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious key-value stores and amplification for private set intersection. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*. Springer, 395–425.

[21] Bishakh Chandra Ghosh, Sikhar Patranabis, Dhinakaran Vinayagamurthy, Venkatraman Ramakrishna, Krishnasuri Narayanam, and Sandip Chakraborty. 2023. Private Certifier Intersection. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/private-certifier-intersection/

[22] Carmit Hazay and Yehuda Lindell. 2008. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008 (Lecture Notes in Computer Science, Vol. 4948)*, Ran Canetti (Ed.). Springer, 155–175. https://doi.org/10.1007/978-3-540-78524-8_10

[23] Carmit Hazay and Yehuda Lindell. 2010. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *J. Cryptol.* 23, 3 (2010), 422–456. https://doi.org/10.1007/S00145-008-9034-X

[24] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. 2021. PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3577–3594. https://www.usenix.org/conference/usenixsecurity21/presentation/heinrich

[25] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. 2019. Private set intersection with linear communication from general assumptions. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*. 14–25.

[26] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. 2017. Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions. *IACR Cryptol. ePrint Arch.* (2017), 738.

[27] Justus M Kebschull and Anthony M Zador. 2018. Cellular barcoding: lineage tracing, screening and beyond. *Nature methods* 15, 11 (2018), 871–879.

[28] Florian Kerschbaum. 2012. Outsourced private set intersection using homomorphic encryption. In *ASIACCS '12: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. 85–86. https://doi.org/10.1145/2414456.2414506

[29] Hugo Krawczyk. 2019. Oblivious Pseudorandom Functions and Some (Magical) Applications. (2019). "ObliviousPseudorandomFunctionsandSome(Magical) Applications"

[30] Miracl. 2024. Miracl. https://github.com/miracl/MIRACL/tree/master

[31] Daniel Morales, Isaac Agudo, and Javier Lopez. 2023. Private set intersection: A systematic literature review. *Computer Science Review* 49 (2023), 100567. https://doi.org/10.1016/j.cosrev.2023.100567

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

[33] Marcin Nagy, Emiliano De Cristofaro, Alexandra Dmitrienko, N. Asokan, and Ahmad-Reza Sadeghi. 2013. Do I know you?: efficient and privacy-preserving common friend-finder protocols and applications. In *Proceedings of the 29th Annual Computer Security Applications Conference — ACSAC 2013*. 159–168. https://doi.org/10.1145/2523649.2523668

[34] Moni Naor and Benny Pinkas. 2001. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (Washington, D.C., USA) *(SODA '01)*. Society for Industrial and Applied Mathematics, USA, 448–457.

[35] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. 2011. 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.

[36] Liyan Shen, Xiaojun Chen, Dakui Wang, Binxing Fang, and Ye Dong. 2018. Efficient and Private Set Intersection of Human Genomes. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 761–764.

[37] MITSUNARI Shigeo. 2024. mcl. https://github.com/herumi/mcl

[38] Emil Stefanov, Elaine Shi, and Dawn Song. 2012. Policy-Enhanced Private Set Intersection: Sharing Information While Enforcing Privacy Policies. In *Public Key Cryptography – PKC 2012*, Marc Fischlin, Johannes Buchmann, and Mark Manulis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 413–430. https://doi.org/10.1007/978-3-642-30057-8_25

[39] Shintaro Terada and Kazuki Yoneyama. 2018. Improved Verifiable Delegated Private Set Intersection. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*. 520–524. https://doi.org/10.23919/ISITA.2018.8664310

[40] Connie M Westhoff. 2019. Blood group genotyping. *Blood, The Journal of the American Society of Hematology* 133, 17 (2019), 1814–1820.

[41] Andrew C. Yao. 1982. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (FOCS 1982)*. 160–164.

A Instantiations

A.1 Exponent Equality Argument

We use an exponent equality argument EEA to prove that a set of group elements $H_1(w_1)^r, \ldots, H_1(w_n)^r \in \mathbb{G}_1$ are raised to the same random r. We use the Fiat-Shamir heuristic to make the protocol non-interactive. To generate a proof, the prover first samples a random element z and computes $t_i \leftarrow H_1(w_i)^z$ for all $i \in [n]$. The challenge is computed non-interactively as a function of the protocol transcript, $c \leftarrow H(\{t_i, H_1(w_i)\}_{i\in[n]})$. The prover then sends $\{t_i\}_{i\in[n]}$ and the value $s \leftarrow z + cr$.

To verify, the verifier takes as input the commitment $\{h_i = H_1(w_i)^r\}_{i\in[n]}$ and the elements $\{w_i\}_{i\in[n]}$, as well as $\{t_i\}_{i\in[n]}$ and s. For each $i \in [n]$, it checks whether $H_1(x_i)^s = t_i \cdot h_i^c$. If yes, then it accepts. Otherwise, it rejects the proof.

The pseudocode for these algorithms can be found in Figure 6 for interactive and Figure 7 for non-interactive.

Theorem 8. *The exponent equality argument* EEA *described in Figure 6 is complete, Honest-Verifier Zero Knowledge, and knowledge-sound.*

Proof. Below we show (i) completeness, (ii) Honest-Verifier Zero Knowledge (HZRK) and (iii) knowledge-soundness.

Completeness: For all i, we have that

$$H_1(w_i)^s = h_i^c = H_1(w_i)^{z+cr} = t_i \cdot (H_1(w_i)^r)^c.$$

HVRK: We show that for every $H_1(w_i)$, the output of our simulator is indistinguishable from the output of the transcript. We construct a simulator as follows:
(1) Sample $s \leftarrow\$ \mathbb{Z}_q^*$ and $c \leftarrow\$ \mathbb{Z}_q^*$.
(2) For all i, set $t_i = H_1(w_i)^s / h_i^c$.
(3) Output $(\{t_i\}_{i\in[n]}, c, s)$.
Note that the c is uniformly random and so are the t_i, and s is correct. Thus, the output of the simulator is random and distributed like the real transcript.

Knowledge Soundness: We construct an extractor:
(1) Run the protocol to obtain transcript (t_i, c, s) for each $i \in [n]$.
(2) Rewind and rerun the protocol to obtain a second transcript (t_i, c', s') for each $i \in [n]$.
(3) Compute

$$\frac{s - s'}{c - c'} = \frac{(z + cr) - (z + c'r)}{c - c'} \equiv r\frac{c - c'}{c - c'} = r.$$

□

A.2 OPRF

The OPRF protocol [29] is carried out between the client C with key k and the server S with its set of elements $\mathcal{Y}$. First, the server makes a Request for the PRF to be applied to its elements, then the client applies the Eval function on them, and finally the server S recovers the elements with the PRF applied on them using the Recover function (Figure 8).

Theorem 9. *The OPRF protocol described in Figure 8 is correct, hides k from* S *and hides all elements in the request from* C*, assuming DDH.*

Let $H_1 : \mathbb{Z}_q \to \mathbb{G}_1$ be a hash function.

```
1:  Prover(r, {w_i}_{i∈[n]})
2:      z ←$ Z_q^*
3:      for i ∈ [n] do
4:          t_i ← H_1(w_i)^z
5:      return {t_i}_{i∈[n]}

6:  Verifier({w_i}_{i∈[n]}, {t_i}_{i∈[n]})
7:      c ←$ Z_q^*
8:      return c

9:  Prover(r, {w_i}_{i∈[n]}, c)
10:     return s = z + cr

11: Verifier({w_i}_{i∈[n]}, {t_i}_{i∈[n]}, s)
12:     for i ∈ [n] do
13:         if H_1(w_i)^s ≠ t_i h_i^c then // h_i = H_1(w_i)^r
14:             return 0
15:     return 1
```

Figure 6: Pseudocode for the interactive Exponent Equality Argument (Schnorr's protocol).

Let $H_1 : \mathbb{Z}_q \to \mathbb{G}_1$ and $H : \{0,1\}^* \to \mathbb{G}_1$ be hash functions.

```
1:  Prove(r, {w_i}_{i∈[n]}) → π
2:      z ←$ Z_q^*
3:      for i ∈ [n] do
4:          t_i ← H_1(w_i)^z
5:      c ← H_1({t_i, w_i}_{i∈[n]})
6:      s ← z + cr
7:      return ({t_i}_{i∈[n]}, s)

8:  Verify(π, {w_i}_{i∈[n]}, {h_i}_{i∈[n]}) → b
9:      ({t_i}_{i∈[n]}, s) ← π
10:     c ← H({t_i, w_i}_{i∈[n]})
11:     for i ∈ [n] do
12:         if H_1(w_i)^s ≠ t_i · h_i^c then  // h_i = H_1(w_i)^r
13:             return 0
14:     return 1
```

Figure 7: Pseudocode for the non-interactive Exponent Equality Argument EEA = (Prove, Verify)**, after applying the Fiat-Shamir Heuristic.**

Proof. **Correctness:** For each element $a \in \mathcal{Y}$ of S, S raises them to some random value t. Then, C raises them to the k and finally S raises them to $\frac{1}{t}$. We have:

$$((a^t)^k)^{\frac{1}{t}} = a^k$$

Hiding k: S receives elements a^{tk} and can then turn them into a^k, for all $a \in \mathcal{Y}$. S cannot recover the key k, as that would break the DDH assumption.

1: // Server S initiates a request:
2: Request($\mathcal{M}$) $\to t, \mathcal{B}$
3: $t \leftarrow\$ \mathbb{Z}_q$
4: $\mathcal{B} \leftarrow \{a^t\}_{a\in\mathcal{M}}$
5: **return** $t, \mathcal{B}$
6: // Client C applies PRF:
7: Eval($\mathcal{B}, k$) $\to C$
8: **return** $\{b^k\}_{b\in\mathcal{B}}$
9: // Server S recovers elements:
10: Recover(C, t) $\to \mathcal{D}$
11: **return** $\{c^{\frac{1}{t}}\}_{c\in C}$

Figure 8: OPRF Protocol

Hiding S's input: C receives elements a^t. Since t is chosen uniformly at random, the elements a^t are indistinguishable from random. □

B Proofs

B.1 Proof of Theorem 1

Proof. Let $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ be elements of C and S, respectively, and let $\sigma = H_1(x\|\mathtt{ID}_C)^{\mathsf{sk}}$ be the signature of x issued by the judge. Suppose $x = y$ and observe the following:

$$\hat{x} = e(\sigma, S) = e(H_1(x\|\mathtt{ID}_C)^{\mathsf{sk}}, g_2^s) = e(H_1(y\|\mathtt{ID}_C), \mathsf{pk}^s) = \hat{y}.$$

Thus we have that $\hat{x} = \hat{y}$. Moreover, since σ is valid signature and by the unforgeability of the BLS signature scheme the client could not have computed it without the judge's secret key, and so $x \in \mathcal{X}_{\text{AUTH}}$ which implies that $x \in (\mathcal{X}_{\text{AUTH}} \cap \mathcal{X}) \cap \mathcal{Y}$.

Now suppose $x \neq y$. There are two cases: (1) $e(H_1(x\|\mathtt{ID}_C), \mathsf{pk}^s) \neq e(H_1(y\|\mathtt{ID}_C), \mathsf{pk}^s)$ or (2) they are equal. In case (1), the values are not equal and hence x does not appear in the intersection. In case (2), $H_1(x\|\mathtt{ID}_C) = H_1(y\|\mathtt{ID}_C)$ and $x\|\mathtt{ID}_C \neq y\|\mathtt{ID}_C$, and so we have found a collision in H_1. This case happens with negligible probability and the theorem follows. □

B.2 Proof of Theorem 4

Proof. We show that the only elements in the intersection are (i) signed by the judge and (ii) in both $\mathcal{X}$ and $\mathcal{Y}$. Let $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, and let $\sigma = H_2(H_1(x\|\mathtt{ID}_C)^r)^{\mathsf{sk}}$ be the valid signature of x issued to C by the judge. Suppose that $x = y$, then:

$$\hat{x} = e(\sigma, S) = e(H_2(H_1(y\|\mathtt{ID}_C)^r)^s, \mathsf{pk}) = \hat{y}$$

Thus, we have that $\hat{x} = \hat{y}$. Additionally, this equality holds because σ is valid signature and by the unforgeability of the BLS signature scheme the client could not have computed it without the judge's secret key sk, and so $x \in \mathcal{X}_{\text{AUTH}}$ which implies that $x \in (\mathcal{X} \cap \mathcal{X}_{\text{AUTH}}) \cap \mathcal{Y}$.

Now suppose $x \neq y$, then either (1) $e(H_2(H_1(x\|\mathtt{ID}_C)^r), g^{\mathsf{sk}\cdot s}) \neq e(H_2(H_1(y\|\mathtt{ID}_C)^r), \mathsf{pk}^s)$ or (2) they are equal. In case (1), $\hat{x} \neq \hat{y}$ and hence x does not appear in the intersection. In case (2), $H_2(H_1(x\|\mathtt{ID}_C)^r) = H_2(H_1(y\|\mathtt{ID}_C)^r)$ and so we have found a collision in H_1 or H_2. Both happen with negligible probability and the theorem follows. □

B.3 Proof of Theorem 5

Proof. We prove the security through a series of hybrid arguments. We now describe our simulator. Upon input of the random index $\mathcal{I} \subseteq_\$ [n]$ such that $|\mathcal{I}| = \lceil pn \rceil$ and the corresponding values $\{x_i\}_{i\in[n]}$, the simulator proceeds as follows:

- **Hash queries.** Model the hash function H_1 as a random oracle and store the results in a table T. For each query q to H_1, check if $q \in T$. If yes, then return $T[q] = h$. If not, then sample $h \leftarrow\$ \mathbb{G}_1$ and set $T[q] \leftarrow h$.
- **Authorize.** To simulate the client C during the authorize phase, the simulator does the following.
  - Sample value $r \leftarrow\$ \mathbb{Z}_q$.
  - For all $i \in [n] \setminus \mathcal{I}$, sample a random value $\mathcal{X}_B^{(i)} \leftarrow\$ \mathbb{G}_1$.
  - For all $i \in \mathcal{I}$, make a hash query to H_1 on $x_i\|\mathtt{ID}_C$ (table T_1 is updated accordingly) and compute $\mathcal{X}_B^{(i)} \leftarrow H_1(x_i\|\mathtt{ID}_C)^r$. If $\mathcal{X}_B^{(i)} = \mathcal{X}_B^{(j)}$ for any $i \neq j, i, j \in [n]$, abort.
  - Send the vector $\mathcal{X}_B = (\mathcal{X}_B^{(1)}, \ldots, \mathcal{X}_B^{(n)})$ to the judge.
  - Given $\mathcal{I}$, compute $\pi \leftarrow \mathsf{EEA.Prove}(r, \{x_i\}_{\mathcal{I}})$ and return $\pi, \{x_i\}_{\mathcal{I}}$.

We now describe a series of hybrids.

Hybrid 0: The real interaction, where both parties run the Authorize protocol honestly on input $\mathcal{X}$.

Hybrid 1: Same as Hybrid 0, except sample $r \leftarrow\$ \mathbb{Z}_q$. This is indistinguishable from Hybrid 0, since r is distributed equally.

Hybrid 2: Same as Hybrid 1, except we abort if there exist $x, x' \in \mathcal{X}$ such that $H_1(x\|\mathtt{ID}_C) = H_1(x'\|\mathtt{ID}_C)$. For some fixed $x \neq x'$, the probability of such a collision is $1/|\mathbb{G}_1|$. Union bounding over all x we get a total collision probability of $n^2/|\mathbb{G}_1|$ which is negligible. Thus, Hybrid 2 is indistinguishable from hybrid 1.

Hybrid 3: Same as Hybrid 2, except for $i \in [n]\setminus\mathcal{I}$, $(H_1(x_i||ID_C)^r$ in $\mathcal{X}_B$ is replaced with a random value in $\mathbb{G}_1$. By the DDH assumption holding in group $\mathbb{G}_1$, the new binding commitments are indistinguishable from those in Hybrid 2. The second message from the client $(\pi, \{x_i\}_{i\in\mathcal{I}})$ do not depend on any $x_i, i \in [n] \setminus \mathcal{I}$, so they are not affected by this change.

Hybrid 3 is identical to our simulator. Since the simulator is indistinguishable from the real world, the theorem follows. □