Element Distinctness and Bounded Input Size in Private Set Intersection and Its Variants

Xavier Carpent[1], Seoyeon Hwang[2], and Gene Tsudik[2]

[1] University of Nottingham, Nottingham, UK
[2] University of California, Irvine, Irvine, USA

Abstract. This paper considers a Private Set Intersection (PSI) protocol variant in which a lower bound on one party's input set size is enforced while keeping that size secret. This variant is suitable for settings where one party (server) imposes a minimum size on the other party's (client's) input to obtain the computation result, and the latter wants to keep its input size private. In this context, two possible types of client misbehavior are: (1) generating and using fake set elements, and/or (2) duplicating genuine set elements. The former can be addressed via pre-authorizing all client elements by a trusted party, which translates into the so-called Authorized PSI (APSI). However, the latter is not easy to tackle.
To this end, we construct a protocol for *Proof of Element-Distinctness (PoED)* whereby one party convinces the other that all of its input elements are distinct, without revealing any information about them. Using it as a building block, we then construct a PSI variant, called *All-Distinct Private Set Intersection (AD-PSI)*, that outputs the set intersection only when the client inputs all distinct elements. We also present some AD-PSI variants in which using duplicates can cause unexpected information leakage. Combining the AD-PSI with previous work for upper-bounded PSI, we construct a *Bounded-Size-Hiding-PSI (B-SH-PSI)* that outputs the intersection only if the client's input size satisfies the server's requirement on *both* lower and upper bounds while keeping that size private. Finally, we present a protocol that prevents both types of misbehavior, called *All-Distinct Authorized PSI (AD-APSI)*.

Keywords: Private Set Intersection, Input Correctness, Element Distinctness, Bounded Input, Size-Hiding

1 Introduction

Private Set Intersection (PSI) is a cryptographic primitive that allows two parties to compute the intersection of their private input sets, without revealing any information about the set elements outside the intersection to each other. It has garnered a lot of attention from various privacy-preserving applications, such as contact tracing [25,75], online targeted advertising [45], genomic testing [51], botnet detection [59], TV program history matching [47], private contact discovery [23,38], and private matchmaking [79].

Due to its functionalities applicable to numerous real-world applications, there has been a long line of work in PSI and its variants (details in Section 2), starting from the earliest forms in 1980s [56,73]. While most PSI protocols reveal set (input) sizes as part of the protocol, [3] constructed the first PSI variant – *Size-Hiding PSI (SH-PSI)* – that allows one party (*Client*) to learn the intersection while keeping its input set size private against the other party (*Server*). Building upon this size-hiding property, Bradley et. al. [6] and Cerulli et. al. [9] suggested upper-bounding *Client*'s input set size to prevent it from learning too much information about *Server*'s input set.

This paper focuses on a related question: lower-bounding *Client*'s input set size while keeping it private. That is, suppose that *Server* requires *Client* to have at least l elements in its input set to run the PSI protocol with *Server*'s set. This requirement might be useful in social network settings, such as Facebook and LinkedIn, where a popular/prominent user would agree to connect to another user only if the latter has at least l genuine friends/followers to e.g., block the stalkers who keep creating bogus accounts and requesting to connect.

If we relax the size-hiding property, lower-bounding *Client* input size is straightforward: *Server* simply checks whether the *Client* set size (revealed as part of the PSI interaction) is $\geq l$, and if not, aborts the protocol. However, this only works if *Client* is honest. A dishonest *Client* can bypass this requirement by (1) generating and using fake set elements, and/or (2) duplicating its genuine set elements. Then, since PSI protocols typically obfuscate (often by blinding) *Client* set elements, *Server* cannot distinguish between the genuine and fake input elements.

One intuitive way to mitigate this misbehavior is via auditing: a trusted third party (TTP) regularly verifies the *Client* input by examining the transcripts of PSI protocol and looking for duplicate or spurious elements. However, this would be too late since the dishonest *Client* already obtained the intersection.

To deal with the type-(1) misbehavior, so-called *Authorized PSI* (APSI) techniques [19,20,77] have been proposed. This is achieved by having an offline TTP that pre-authorizes *Client* input by signing each element. Later, during PSI interaction, *Server* (implicitly or explicitly) verifies these signatures without learning *Client* input. This way, *Client* cannot obtain signatures of spurious elements, and thus, cannot learn the intersection using fake elements. However, APSI protocols cannot cope with the type-(2) misbehavior, i.e., *Client* can still bypass the requirement by using duplicated (TTP-authorized) signed elements. This prompts a natural question:

Can Client prove that each of its private input elements is not duplicated, i.e., all input elements are distinct while keeping them private?

To answer this question, we first investigate if current PSI protocols can detect duplicates (see Section 2.4). A few prior results [5,49] proposed protocols for *Private Multiset*[3] *Intersection (PMI)* which allows *multiset* inputs. However, we note that their goal is different because it outputs the intersection *multiset*, not

[3] Recall that, a multiset allows duplicate elements, while a set does not.

the intersection *set*, which yields more information than PSI, e.g., the number of occurrences (i.e., multiplicities) of common elements.

Next, we show how to prove *element distinctness* in two-party settings, whereby one party convinces the other that its input elements are all distinct, without revealing any information about them. We use the term *element distinctness* (a.k.a. *element uniqueness*) problem from the computational complexity theory: given n numbers $x_1, ..., x_n$, return "Yes" if all x_i's are distinct, and "No" otherwise. To the best of our knowledge, there is no prior work in the two-party settings where one party *proves element distinctness of its private input* to the other party. We call this ***P**roofs **o**f **E**lement-**D**istinctness (**PoED**)*.

We propose a concrete PoED construction by generalizing the two billiard balls problem, which can be an independent interest. Using this PoED construction as a building block, we propose a new PSI variant, called *All-Distinct Private Set Intersection (AD-PSI)*, and its construction. Informally speaking, AD-PSI allows *Client* to learn the intersection only if all of its input elements are distinct. It additionally guarantees that *Client* learns no information, not even *Server* input size, if it uses any duplicates as input.

Then, we extend AD-PSI to three PSI variants where using duplicates can be more problematic: (1) *AD-PSI-Cardinality (AD-PSI-CA)* that outputs the cardinality of the intersection; (2) *Existential AD-PSI (AD-PSI-X)* that outputs whether the intersection is non-empty; and (3) *AD-PSI with Data Transfer (AD-PSI-DT)* that transfers associated data along with the intersection; only when *Client* inputs all distinct elements. We also show a *Bounded-Size-Hiding-PSI (B-SH-PSI)* construction with both upper and lower bound on *Client* input, combining our AD-PSI with prior work on upper-bounded size-hiding PSI (U-SH-PSI) [6,9], which shows the applicability of PoED and AD-PSI.

Note that the protocols above work in the case where *Client* cannot generate fake elements, and to expand *Client*'s capabilities to both type-(1) and type-(2) misbehavior, including a TTP is unavoidable. To fill this gap, we finally present an *All-Distinct Authorized PSI (AD-APSI)* protocol that prevents both duplicate and spurious elements by ensuring the validity of *Client* input. We specify desired security properties for AD-APSI and prove that the proposed protocol satisfies them.

To summarize, the contributions of this work are:

- A PoED protocol with security analysis;
- Definition of AD-PSI and concrete construction with security proofs;
- Three AD-PSI variants: AD-PSI-CA, AD-PSI-X, AD-PSI-DT;
- Extension of U-SH-PSI to B-SH-PSI with both upper and lower bounds on *Client* input set size; and
- Definition of AD-APSI and concrete construction with security proofs;

Organization: After overviewing related work and preliminaries in Section 2, Section 3 presents a PoED construction and its analysis. Then, Section 4 defines AD-PSI and proposes a concrete protocol, followed by some variants in Section 5. Section 6 constructs a B-SH-PSI protocol atop U-SH-PSI, and Section 7 demonstrates an AD-APSI protocol and its security proofs.

2 Related Work & Background

2.1 Private Set Intersection (PSI)

Private Set Intersection (PSI) in two-party computation is an interaction between *Client* and *Server* that computes the intersection of their private input sets. A long line of work on PSI can be classified according to the underlying cryptographic techniques: (1) Diffie-Hellman key agreement [6,19,44,56,73]; (2) RSA [18,20,21]; (3) cryptographic accumulators [3,20]; (4) oblivious transfer (or oblivious pseudorandom function) [10,40,42,46,50,60,63,65,66,67,68,69,70]; (5) Bloom filter [22,24,64,69]; (6) oblivious polynomial evaluation [16,30,32,41,49]; and (7) generic multiparty computation [24,43,69,13,55]. This paper considers the *one-sided* PSI where *Client* learns the result. Most efficient protocols incur $O(n)$ computation/communication costs, where n is the input set size.

2.2 PSI Variants

Some PSI variants reveal less information than the actual intersection. For example, PSI-CA [18,22,25,72,76] outputs only the cardinality of the intersection, and PSI-X [8] outputs a one-bit value reflecting whether the intersection is non-empty. On the other hand, some reveal more information, such as associated data for each intersecting element [20,78] or additional private computation results (e.g., sum or average) along with the intersection [45,57,52,55]. The latter is more interesting because of their realistic applications, such as statistical analysis for, e.g., advertisement conversion rate [45], of intersecting data.

2.3 PSI with Restrictions

Certain PSI variants place conditions for *Client* to obtain the result. For example, *threshold PSI* (t-PSI) reveals the intersection only if the cardinality of the intersection meets a *Server*-set threshold value [32,33,39,68,78,79], and its variants, such as t-PSI-CA or t-PSI-DT (also called, *threshold secret transfer)* [78], reveals the intersection or additional data only when the threshold restriction is met or reveals only the cardinality, otherwise. Zhao and Chow [78] extend this to PSI with a generic access structure so that *Client* can learn the result only when the intersection satisfies a certain structure. Also, they build the below/over t-PSI [79] such that *Client* can reconstruct the secret key used by *Server* only when the threshold condition is met, which inspires some steps in our protocols.

On the other hand, Bradley et. al. [6] first suggest the *Bounded-Size-Hiding-PSI* which restricts the *Client input*, i.e., *Client* learns the intersection only if the size of its *input* does not exceed a *Server*-set upper bound in the random oracle model, and later Cerulli et. al. [9] improve it to be secure in the standard model. Compared to the other PSI literature which naturally reveals the input set sizes during the computation, [6] and [9] additionally hide that cardinality information from each other. We note that there has been no PSI variant that places a lower bound (or both lower and upper bounds) on *Client* input.

2.4 PSI with Multiset Input

We now consider how current PSI protocols handle multisets. Note that adversary models in PSI literature do include malicious *input*. Loosely speaking, Honest-but-Curious (HbC) (a.k.a. semi-honest) adversaries try to learn as much as possible while honestly following the protocol, while malicious adversaries arbitrarily deviate from the protocol. However, according to Lindell and Pinkas [54], such adversaries can not be prevented from refusing to participate in a protocol, supplying arbitrary input, or prematurely aborting a protocol instance. Since PSI security is generally based on *sets*, *multisets* can be viewed as malicious inputs. PSI protocols do not offer security against multiset inputs. i.e., Security against malicious adversaries does not mean that multiset inputs are "automatically" handled.

It turns out that some PSI protocols are incompatible with multiset inputs because they assume set input, i.e., distinctness of all elements. For example, [5] and [43] obliviously sort elements and compare the adjacent elements to compute the intersection by checking for equality [43] or erasing each element once [5]. Thus, these protocols output incorrect results with multiset inputs. Furthermore, PSI protocols based on Cuckoo hashing [30,65,68,69] can encounter unexpected errors with multiset inputs. Cuckoo hashing maps each input element into a hash table using some hash functions such that each bin contains at most one element. Since the hash of the same input value is always the same, duplicates can cause an infinite loop (to find an available bin) or result in a waste of resources, e.g., repeating steps until a certain threshold and increasing the stash size.

There exist some PSI protocols that either enforce input element distinctness or are compatible with multiset inputs. For example, the party creates a polynomial that has roots on its input values in [16,32,49] to perform oblivious polynomial evaluation, which by nature filters the duplicates. [64] also guarantees the set input by a new data structure, called PaXoS, which disables encoding any non-distinct elements. On the other hand, security of [40,46] is unaffected by duplicates because it uses an oblivious pseudo-random function to obtain some (random-looking) numbers for its private elements, and then compare the received messages.

Our work focuses on PSI protocols that are incompatible with multiset inputs and suggests adding some simple steps to ensure element distinctness of private input so that they can work properly.

2.5 Zero-Knowledge Proofs

The notion of Zero-Knowledge Proof (ZKP) is first introduced by [35] which is the zero-knowledge interactive proof system. Informally, an *interactive proof system* for a language L is defined between a prover (Prv) and a verifier (Vrf) with a common input string x and unbiased coins, where Prv tries to convince Vrf that x is indeed in L while keeping their coin tosses private. It must be *complete*, i.e., for any $x \in L$, Vrf accepts, and *sound*, i.e., for any $x \notin L$,

Vrf rejects no matter what Prv does. The interactive proof is *zero-knowledge* if given only x, Vrf could simulate the entire protocol transcript by itself without interacting with Prv. A *proof-of-knowledge* [29,4] is an interactive proof where Prv tries to convince Vrf that it has "knowledge" tying the common input x, which requires the *completeness* and *knowledge extractibility* (stronger notion of *soundness*) properties. *Knowledge extractibility* (a.k.a.*validity*) is that for any Prv who can make Vrf accept its claim with non-negligible probability, there exists an efficient program K called *knowledge extractor*, such that K can interact with Prv and output a witness w of the statement $x \in L$. *Zero-Knowledge Proof of Knowledge (ZKPoK)* adds the *zero-knowledge* property on top of them. Compared to ZKP, ZKPoK keeps the one-bit information (whether $x \in L$ or not) private from Vrf, thus realizing "zero"-knowledge.

2.6 Homomorphic Encryption

Homomorphic encryption (HE) is a special type of encryption that allows users to perform certain arithmetic operations on encrypted data such that results are meaningfully reflected in the plaintext. It is called *Fully Homomorphic Encryption (FHE)* when a HE supports *both* unlimited addition and multiplication of ciphertexts. Whereas, a scheme that supports a limited number of operations of either type is called *Somewhat Homomorphic Encryption (SWHE)* and a scheme that supports only one operation type is called *Partially Homomorphic Encryption (PHE)*. There are many PHE schemes such as [14,15,17,26,36,62,58,61,48,71]. For example, ElGamal encryption scheme [26] is a well-known PHE supporting multiplication, and a variant of ElGamal [15] having g^m instead of m and Paillier [62] are well-known PHE schemes supporting addition.

3 Proving Element Distinctness

We first define ***P**roofs **o**f **E**lement-**D**istinctness (PoED)* in the two-party settings where Prv proves element distinctness of its private input elements to Vrf. i.e., PoED is an interactive proof system, where Prv tries to convince Vrf that its input $\mathcal{C} := [c_1, ..., c_n]$ consists of distinct elements, without revealing any other information about each c_i. As a result, Vrf *accepts* or *rejects* the Prv's claim. Following the notation for ZKPoK introduced by [7], PoED is denoted as:

$$PK\{\mathcal{C} \mid e_i = f(c_i) \text{ for each } c_i \in \mathcal{C}, \text{ and } c_i \neq c_j \text{ for } \forall c_i, c_j \in \mathcal{C} \text{ such that } i \neq j\},$$

where f is a function that "hides" c_i so that Vrf does not learn any information about c_i from e_i, while it "binds" c_i to e_i so that Prv cannot change c_i once e_i is computed, e.g., via randomized encryption or cryptographic commitments.

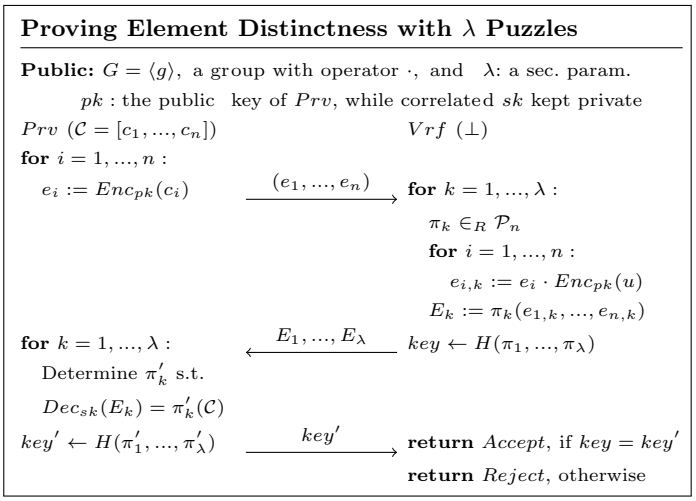

Fig. 1. The PoED-puzzle Protocol. (Enc, Dec) can be any PHE over G. $\mathcal{P}_n$ is a set of random permutations for n elements. H is a cryptographic hash function that maps the arbitrary-length messages to κ-bit values. u is the unit element of the message space.

3.1 Puzzle-Based PoED Construction

The main idea starts from the well-known *two billiard balls* problem where Prv has two billiard balls and (honest) Vrf is color-blind. To convince Vrf that two balls have different colors, the following "*puzzle*" is repeated k times:

1. Prv puts a ball in each hand of Vrf
2. Vrf puts both hands behind its back and decides (at random) whether to switch the balls
3. Vrf shows the balls to Prv
4. Prv declares whether a switch occurred
5. If Prv answers incorrectly, Vrf concludes that Prv cheated

If Prv answers correctly k times, Vrf concludes that, with probability 2^{-k}, the balls have different colors.

Extending this problem to many balls, we construct a PoED protocol and call it *PoED-puzzle* protocol. Instead of the color-blind Vrf, Prv encrypts each element with its public key under an encryption scheme satisfying the ciphertext indistinguishability (IND) property. Since all IND-secure encryption schemes are non-deterministic, Prv can hide the information about the input elements.

To form the puzzles such that Vrf can generate while Prv can solve only when all input elements are distinct, PoED-puzzle needs a PHE scheme over a cyclic group G[4] of prime order with a generator g. i.e., Assume that each of Prv input values is a group element in G, or we can assume a deterministic map that maps each input value c_i to a group element in G. Since any PHE allows Vrf

[4] We sometimes denote G as a subgroup of $\mathbb{Z}_p^*$ whose prime order is known, which will be explicitly indicated in such case.

to re-randomize received ciphertexts by multiplying the encryption of the unit element u (under Prv's public key), this computation gives a new ciphertext of the same plaintext without learning/requiring anything about the plaintext.

Finally, Vrf chooses a random permutation π from $\mathcal{P}_n$, the set of all permutations of length n, and shuffles the re-randomized ciphertexts with π, as if it "switches or not" in the two billiard balls problem. Once it receives a puzzle, Prv decrypts each ciphertext with its private key, determines the permutation π' that shuffles original elements to received elements, and forwards π' to Vrf. Vrf accepts if $\pi' = \pi$.

There is a probability that Prv can solve the puzzle without having all distinct elements. In the worst case when Prv uses only one duplicate, Prv can correctly solve the puzzle with 50% probability. To make the cheating probability low, the puzzle should be repeated λ times, such that $1/2^{\lambda}$ becomes negligible.

Since each puzzle is independent, Vrf can hash the puzzles (using a suitable cryptographic hash function H) and check this hash value, instead of repeating this three-message exchange multiple times for each puzzle. This reduces the number of communication rounds and associated delays. Fig. 1 presents the PoED-puzzle protocol described above.

3.2 Analysis of PoED-puzzle Protocol

Theorem 1. *Assuming an IND-secure PHE scheme (Enc, Dec) over a cyclic group G, a secure cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^{\kappa}$, and the statistical security parameter λ, the PoED-puzzle protocol described in Section 3.1 is a secure PoED protocol.*

(*Sketch Proof*) *Completeness* is straightforward because only one correct permutation π exists for distinct elements, and honest Prv can easily determine π after decrypting the ciphertexts. For the *knowledge extractability*, the private key of the underlying encryption scheme can be seen as the witness. Suppose Prv does not know the private key. Then, by the IND and homomorphic property, re-randomized and shuffled ciphertexts from Vrf are indistinguishable from random strings in the ciphertext space. Furthermore, after decryption, the probability of having duplicates and solving the puzzle correctly is *at most* $2^{-\lambda}$ which is set to be negligible by the security parameter λ. Lastly, *zero-knowledgeness* naturally follows from the IND property, since all Vrf can observe are the ciphertexts encrypted by an IND secure PHE. □

Table 1 summarizes the computation and communication complexities of the PoED-puzzle protocol with λ puzzles. C is denoted by the ciphertext space of Enc and H generates a κ-bit hash result. Overall, both complexities are $O(\lambda n)$, where n is the Prv input size.

4 PSI with Element Distinctness Check

Using the PoED as a building block, now we propose a new variant of PSI that requires all the input elements to be distinct, which we call *All-Distinct Private Set Intersection (AD-PSI).*

Table 1. Cost Analysis of the PoED-puzzle Protocol

Computation Cost			
Operation \ Entity	Prv		Vrf
	Offline	Online	Online
Encryption	n	0	λn
Decryption	0	λn	0
Modular multiplication	0	0	λn
Random permutations (of length n)	0	0	λ
Cryptographic hash (input length)	0	λn	λn
Equality check	0	0	1

Group	**Communication Cost**
C	$(\lambda+1)n$
$\{0,1\}^\kappa$	1

4.1 Adversary Model

Among the two parties participating in the computation, *Client* and *Server*, we consider *Server* to be HbC while *Client* can be malicious. This assumption is reasonable in real-life scenarios because *Server* is the one who provides the service to *Client* and multiple barriers (e.g., law/regulation, security systems for their data, and loss of trust deriving loss of customers) exist for them to be malicious. However, *Client* typically maintains less secure systems and much less data than *Server* so they may be eager to learn more from *Server*'s large dataset. Note that we consider a stronger guarantee than normal malicious security in PSI literature, as now we aim to enforce the input correctness for *Client*.

4.2 Definition of AD-PSI

We define AD-PSI directly, instead of defining PSI and adding features. We follow the definitions of client and server privacy in related work [31,32,34,40]. Let $View^{\Pi}_{\mathcal{A}^*}(\mathcal{C}, \mathcal{S}, \lambda)$ denotes a random variable representing the view of adversary $\mathcal{A}^*$ (acting as either *Client* or *Server*) during an execution of Π on inputs $\mathcal{C}$ and $\mathcal{S}$ and the security parameter λ.

Definition 1 (All-Distinct Private Set Intersection (AD-PSI)). *consists of two algorithms: {Setup, Interaction}, where:*

- *Setup: an algorithm selecting global/public parameters;*
- *Interaction: a protocol between Client and Server on respective inputs: a multiset $\mathcal{C} = [c_1, ..., c_n]$ and a set $\mathcal{S} = \{s_1, ..., s_m\}$, resulting in Client obtaining the intersection of the two inputs;*

An AD-PSI scheme satisfies the following properties:

- *Correctness: At the end of Interaction, Client outputs the exact intersection of two inputs, only when the elements in $\mathcal{C}$ are all distinct. It outputs $\perp$, o.w.*
- *Server Privacy: For every PPT adversary $\mathcal{A}^*$ acting as Client, we say that a AD-PSI scheme Π guarantees the server privacy if there exists a PPT algorithm P_C such that*

$$\{P_C(\mathcal{C}, \mathcal{C} \cap \mathcal{S}, \lambda)\}_{(\mathcal{C},\mathcal{S},\lambda)} \overset{c}{\approx} \{View^{\Pi}_{\mathcal{A}^*}(\mathcal{C}, \mathcal{S}, \lambda)\}_{(\mathcal{C},\mathcal{S},\lambda)}$$

i.e., on each possible pair of inputs $(\mathcal{C}, \mathcal{S}, \lambda)$*, Client's view can be efficiently simulated by* P_C *on input* $(\mathcal{C}, \mathcal{C} \cap \mathcal{S}, \lambda)$.

- *Client Privacy: For every PPT adversary* $\mathcal{A}^*$ *acting as Server, we say that a AD-PSI scheme* Π *guarantees the client privacy if there exists a PPT algorithm* P_S *such that*

$$\{P_S(\mathcal{S}, \lambda)\}_{(\mathcal{C},\mathcal{S},\lambda)} \overset{c}{\approx} \{View^{\Pi}_{\mathcal{A}^*}(\mathcal{C}, \mathcal{S}, \lambda)\}_{(\mathcal{C},\mathcal{S},\lambda)}$$

i.e., on each possible pair of inputs $(\mathcal{C}, \mathcal{S}, \lambda)$*, Server's view can be efficiently simulated by* P_S *on input* $(\mathcal{S}, \lambda)$.

We note that the security definition above is equivalent to the generic "real-vs-ideal" world simulation definition in the semi-honest model, as shown in [34], with the ideal functionality $\mathcal{F}$ below:

1. Wait for an input multiset $\mathcal{C} = [c_1, .., c_n]$ from *Client*.
2. Wait for an input set $\mathcal{S} = \{s_1, ..., s_m\}$ from *Server*.
3. Give output $(|\mathcal{S}|, \mathcal{C} \cap \mathcal{S})$ to *Client* if $\mathcal{C}$ includes all distinct elements, or output $(|\mathcal{S}|)$, otherwise.
4. Give output $(|\mathcal{C}|)$ to *Server*.

Fig. 2. Ideal Functionality $\mathcal{F}$ for AD-PSI

According to the definition above, we propose a construction using PoED-puzzle protocol, so-called *AD-PSI-puzzle*, in the following section.

4.3 A Construction for AD-PSI based on PoED-puzzle

AD-PSI-puzzle protocol starts with the PoED-puzzle protocol, i.e., *Client* encrypts each input element in G (or the mapped values for each input element to G with a public map) under a PHE and sends the ciphertexts to *Server*, and *Server* generates a secret key *key*, derived from multiple puzzles that shuffle re-randomized ciphertexts with random permutations. Note that now the underlying PHE scheme needs to be multiplicatively homomorphic (instead of any PHE) for the correct computation below.

For computing the intersection without revealing the other elements, *Server* hides *Client*'s ciphertexts and its own input values with the same random element $R \in \mathbb{Z}_p^*$. i.e., *Server* first homomorphically exponentiates *Client* elements with R by e_i^R, which is defined by R homomorphic operations, for each $e_i = Enc(c_i), \forall i$ (by multiplying e_i R times or directly exponentiating R[5]). For its own input values, *Server* computes s_j^R for each $s_j \in \mathcal{S}$ so that if some c_i and

[5] Usually, exponentiation of the underlying plaintext can be done more efficiently than multiplying the ciphertext R times. For example, in ElGamal, encryption of x is $Enc(x) = (g^r, xh^r)$ for some random r, and exponentiating to c can be done either $Enc(x) \cdot ... \cdot Enc(x) = (g^R, x^c h^R) = Enc(x^c)$ or $Enc(x)^c = (g^{cr}, x^c h^{cr}) = Enc(x^c)$.

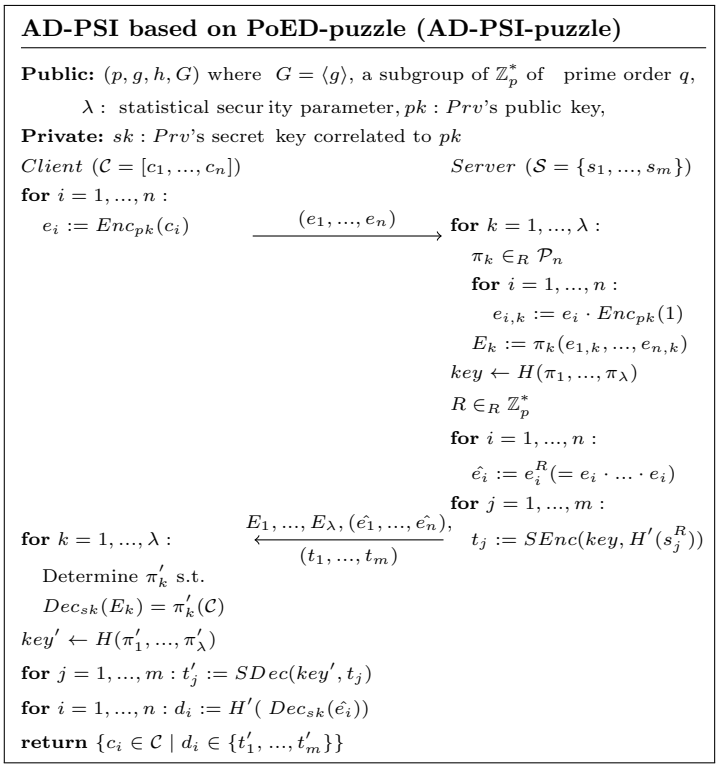

Fig. 3. AD-PSI-puzzle Protocol. (Enc, Dec) is a multiplicative PHE over G and $(SEnc, SDec)$ can be any symmetric encryption scheme over a key space $\{0,1\}^\kappa$. $\mathcal{P}_n$ and H are same as Fig. 1

s_j match, then the randomized c_i^R and s_j^R can also be matched. Then, it hashes each s_j^R using a cryptographic hash function H' and encrypts them under a symmetric encryption scheme with a key key, i.e., $t_j := SEnc(key, H'(s_j^R))$. Thus, unless $Client$ can derive the right key, it cannot decrypt/learn any information about $Server$ elements.

When receiving the messages from $Server$, $Client$ first derives the symmetric key key' by solving all the puzzles, as in PoED-puzzle. Then using the derived key, $Client$ decrypt t_j's, obtains $\{t'_j := H'(s_j^R)\}_j$, and compares them with $d'_i s$, the hash values of the decryption of re-randomized ciphertexts, i.e., $d_i := H'(c_i^R)$ for all i. Finally, $Client$ outputs all c_i's such that d_i matches for some t_j.

The protocol described above is depicted in Fig. 3. Due to the space limit, we show the full security proofs of the following theorem in Appendix A.

Theorem 2. *Assuming the hardness of the decisional Diffie-Hellman problem, the protocol described in Fig. 3 is a secure AD-PSI scheme, satisfying the Definition 1 in ROM.*

Though we define AD-PSI such that it does not reveal whether $\mathcal{C}$ satisfies the element distinctness or not to *Server*, this one-bit information may be favored by *Server* to save its computing resources. We discuss this alternative definition and an idea of modifying the AD-PSI-puzzle protocol in the following section.

4.4 Alternative AD-PSI and Modified Construction

Checking the distinctness of $\mathcal{C}$ before proceeding to the next steps may be preferable by *Server* with a large set $\mathcal{S}$ because the rest computation cost is linear to $|\mathcal{S}|$. Whereas, *Client* may be reluctant as it reveals whether *Client* used all distinct elements to *Server*, i.e., a trade-off between client privacy and server efficiency. For this alternative design, AD-PSI *Correctness* can be defined with *Server* outputs $(|\mathcal{C}|, b)$ in Definition 1 instead, where b is a boolean result of whether $\mathcal{C}$ satisfies the element distinctness. Likewise, $\mathcal{F}$ is modified as below:

1. Wait for an input multiset $\mathcal{C} = [c_1, .., c_n]$ from *Client*.
   Abort if $\mathcal{C}$ includes any duplicates.
2. Wait for an input set $\mathcal{S} = \{s_1, ..., s_m\}$ from *Server*.
3. Give output $(|\mathcal{S}|, \mathcal{C} \cap \mathcal{S})$ to *Client*.
4. Give output $(|\mathcal{C}|)$ to *Server*.

Fig. 4. Ideal Functionality $\mathcal{F}$ for Alternative AD-PSI

To meet this definition, the AD-PSI-puzzle protocol (in Fig. 3) can be modified as in Fig. 5. i.e., Before the intersection computation phase, *Server* first sends all the puzzles to *Client* and proceeds to the next phase only if *Client* corrects all puzzles. Although this modification increases the number of communication rounds, *Server* can save its computation resources for the clients who do not cheat and have enough elements (by size checking) and use this one-bit information in another application (See Section 6).

Table 2 summarizes the computation and communication complexities of the AD-PSI-puzzle protocols with λ puzzles. We denote the cost of the alternative protocol in parentheses only when it has a different cost from the original one. `HE` denotes the partial homomorphic encryption scheme and `SE` denotes the symmetric encryption scheme used in the protocol(s). C_{Π} represents the ciphertext space of a scheme Π and cryptographic hash functions H and H' generate a κ-bit and κ'-bit hash result, respectively. Overall, both complexities are $O(\lambda n + m)$, where n is the *Client* input size (including duplicates, if any) and m is the *Server* input size.

5 AD-PSI Variants

As mentioned earlier in Sections 1 and 2.4, duplication can be more problematic in PSI variants that give additional/restricted information. In this section,

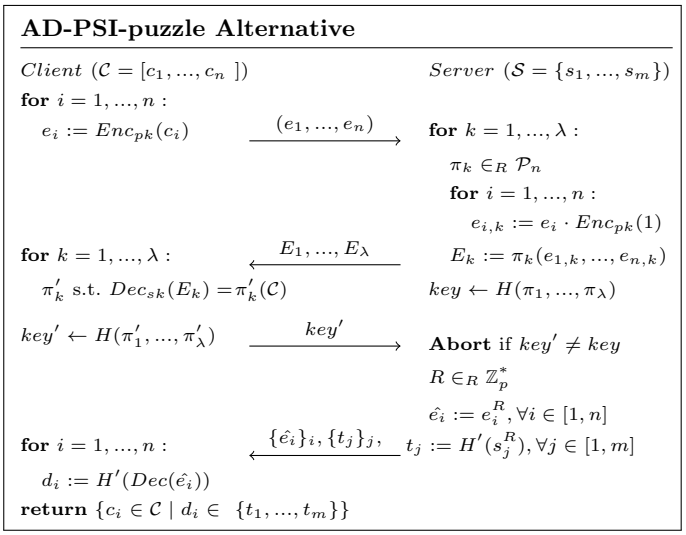

Fig. 5. Alternative AD-PSI Protocol

we further discuss how duplication can leak more information, and propose a solution for each variant using AD-PSI. Although the solutions are simple, we provide the figures for each protocol in Appendix B for better presentation.

Note that we follow the convention in PSI literature and do not consider the information leakage after multiple executions which will naturally reveal more than the one they are supposed to reveal in a single execution. For example, when *Client* deliberately adjusts its input elements to PSI-X and the protocol outputs 'No' in the previous rounds and 'Yes' in the next round, then *Client* learns that the exact element added in the last round is in the *Server* set. Though this is interesting, we consider it as a future work.

5.1 PSI-CA with Element Distinctness (AD-PSI-CA)

Recall that PSI-CA outputs only the cardinality of the intersection set. Suppose *Client* uses a single element as input to PSI-CA. In that case, although it is not malicious behavior, *Client* can learn if that exact element is in $\mathcal{S}$, which is more information than it is supposed to learn. Furthermore, by repeating PSI-CA with different single elements, it can eventually learn the intersection set, or the entire $\mathcal{S}$ if the message space is small enough. To prevent this, *Server* may want to restrict the minimum input set size as l and check if $|\mathcal{C}| > l$ during the computation phase.

However, *Client* still can bypass this simple check by duplicating a single element n times where n is greater than l. Although *Server* does not abort as the *Client* set size n is larger than l, the PSI-CA result with this input will be either '0' or '1' which reveals if the single element is in $\mathcal{S}$ or not. Thus, the

Table 2. Cost Analysis of AD-PSI-puzzle Protocols. We present the cost of the alternative protocol in $(\cdot)$ only when it is different from the original cost.

Computation Cost of AD-PSI-puzzle (and its alternative)				
Operation Entity		*Client*		*Server*
		Offline	Online	Online
HE.Encryption		n	0	λn
HE.Decryption		0	$(\lambda+1)n$	0
Modular Multiplication		0	0	$(\lambda+R)n$
Random number generation (in $\mathbb{Z}_p^*$)		0	0	1
Random permutations (of length n)		0	0	λ
Cryptographic hash	of input length λn	0	1	1
	of input length $\lvert\mathcal{M}\rvert$	0	n	m
Equality check		0	0	0 (1)
Involvement check (i.e., if $a \in A$)		0	n	0
SE.Encryption		0	0	m (0)
SE.Decryption		0	m (0)	0

Group	Communication Cost
$C_{\mathtt{HE}}$	$(\lambda+2)n$
$C_{\mathtt{SE}}$	m
$\{0,1\}^{\kappa}$	0 (1)
$\{0,1\}^{\kappa'}$	0 (m)
#(rounds)	1 (2)

simple size check is not enough, and *Server* needs a way to check the element distinctness of $\mathcal{C}$, which we call *AD-PSI-Cardinality (AD-PSI-CA)*.

The definition of AD-PSI-CA is similar to the one of AD-PSI, except that $(|\mathcal{S}|, |\mathcal{S}\cap\mathcal{C}|)$ is the *Client* output for correctness, and what the ideal functionality gives to *Client* as output. Adding this feature can be simply done by modifying the AD-PSI-puzzle protocol: *Server* additionally chooses a random permutation π and sends the permuted ciphertexts $\hat{e_i} := e^R_{\pi(i)}$ instead of $\hat{e_i} := e^R_i$. Since the ciphertexts are randomized with R by *Server*, and *Client* does not know π, now *Client* cannot match the d_i's to the original c_i's. Furthermore, AD-PSI-puzzle guarantees that *Client* cannot solve the puzzle correctly with overwhelming probability when using duplicated inputs. Therefore, *Client* learns $|\mathcal{C}\cap\mathcal{S}|$, only when it uses all distinct input elements.

5.2 PSI-X with Element Distinctness (AD-PSI-X)

PSI-X outputs very limited information, only the boolean result of whether the intersection of two private input sets is non-empty. Likewise, although *Server* decides on a lower-bound restriction on the size of $\mathcal{C}$, *Client* can obtain more information than the boolean result by using a small input set, because if the result is '1' (i.e., intersection exists), each element is in $\mathcal{S}$ with the probability of $1/|\mathcal{C}|$. *Server*, thus, may have more motivation to restrict the size of $\mathcal{C}$ to reduce this probability.

One way to construct a AD-PSI-Existence (AD-PSI-X) protocol is to add our PoED phase to the FHE-based PSI-X protocol. The basic idea of the FHE-based PSI-X protocol is to encrypt each element under an FHE, compute the subtraction of every pair of $\mathcal{C}$ and $\mathcal{S}$, and multiply all subtractions (with a random number) so that the decryption result can be zero if any of the pairs match. i.e., It computes the encryption of $R \cdot \Pi_{i,j}(c_i - s_j)$ for a random R, which becomes the encryption of zero if any pair of c_i and s_j matches. The recent benchmark [37] on FHE libraries shows that the addition can be done within 100 ms while multiplication requires about 1 second over the integer encoding in many libraries, such as Lattigo [2], PALISADE [1], SEAL [53], and TFHE [12]. The PoED phase can be easily added: *Server* can add the shuffling phase before the PSI-X steps, and just encrypt the final message with the key derived from the puzzles as in the PoED-puzzle protocol.

5.3 PSI-DT with Element Distinctness (AD-PSI-DT)

PSI-DT transfers additional data associated with the intersecting elements. Since this gives more data other than the intersection, when *Server* restricts the *Client* input size, *Client* without enough elements may have more motivation to cheat and bypass the restriction to obtain them. AD-PSI with data transfer (AD-PSI-DT) is defined similarly to AD-PSI, except it outputs $(|\mathcal{S}|, I := \mathcal{S} \cap \mathcal{C}, \{D_j\}_{s_j \in I})$ for *Client*.

An AD-PSI-DT protocol can be constructed as follows: It is the same as AD-PSI-puzzle protocol until randomizing *Client* ciphertexts. Then, for *Server* input elements, *Server* computes one more hash (or a one-way function) H'' and encrypts them under the key derived from the puzzles, i.e., $t_j := SEnc(key, H''(s'_j))$, where $s'_j := H'(s_j^R)$. For the associated data to transfer, *Server* encrypts each D_j using the pre-image of H'', i.e., $D'_j := SEnc(s'_j, D_j)$, and sends them along with the other messages. This prevents *Client* from trying all decryption results as key to decrypt the associated data.

Receiving the messages from *Server*, *Client* performs the same steps to learn the intersection as AD-PSI. To obtain the associated data, *Client* uses the matching d_i's for its own (randomized) values to decrypt and get the data. Security for the non-intersecting elements follows the security of AD-PSI, and the one-way property of H'' and the security of the underlying symmetric encryption scheme guarantee the security of the associated data.

6 Completing Bounded-Size-Hiding-PSI

As mentioned in Section 2.3, *Bounded-Size-Hiding-PSI* was introduced in [6], extending the concept of *Size-Hiding-PSI (SH-PSI)* from [3] by adding an upper bound on the size of *Client* input set $\mathcal{C}$, $|\mathcal{C}|$. For clarification, we denote this primitive by *Upper-bounded-SH-PSI (U-SH-PSI)*. Now we propose a *Bounded-Size-Hiding-PSI (B-SH-PSI) protocol* with **complete, both lower and upper, bounds on** $|\mathcal{C}|$.

In B-SH-PSI, *Server* publishes its restriction rules, L for lower bound and U for upper bound, for $|\mathcal{C}|$. i.e., *Server* wants *Client* to obtain the intersection only when $L \leq |\mathcal{C}| \leq U$. On the other hand, *Client* wants to hide its input size as well as any information about its elements from *Server*. Fig. 6 shows the ideal functionality $\mathcal{F}_B$ for B-SH-PSI described above.

1. Wait for input $\mathcal{C} = [c_1, ..., c_n]$ from *Client*.
2. Wait for input $\mathcal{S} = \{s_1, ..., s_m\}$ from *Server*.
3. Abort if $\mathcal{C}$ does not contain at least L distinct elements, or $|\mathcal{C}| > U$.
   Give the output $(|\mathcal{S}|, \mathcal{C} \cap \mathcal{S})$ to *Client* only if $L \leq |\mathcal{C}| \leq U$.
4. Give output b to *Server*, where b is the boolean value of whether $|\mathcal{C}| \geq L$.

Fig. 6. Ideal Functionality $\mathcal{F}_B$ for B-SH-PSI

We construct a B-SH-PSI protocol using U-SH-PSI and the AD-PSI-puzzle protocols as building blocks, and briefly present it in Fig. 7. To enforce that *Client* cannot learn any information about the intersection without satisfying both upper- and lower-bound requirements, we need the alternative AD-PSI-puzzle protocol (in Section 4.4) that reveals the one-bit information if $\mathcal{C}$ satisfies the lower-bound or not.

Recall that *Client* cannot obtain the next message from *Server* with overwhelming probability if $\mathcal{C}_L$ includes any duplicates. Also, since *Server* can see the size of $\mathcal{C}_L$ during the AD-PSI phase, it can just abort (or send an error message to *Client*) if $\mathcal{C}_L$ does not satisfy the lower bound L. Otherwise, *Server* stores this size $|\mathcal{C}_L|$ and sends some puzzles for AD-PSI to *Client*. The honest *Client* can enclose the first message (the accumulator for the rest of the elements in $\mathcal{C}$, i.e., $\mathcal{C}^* := \mathcal{C} \setminus \mathcal{C}_L$), $\mathtt{msg}_1$, along with the key' derived from the given puzzles. If key' is correct, *Server* proceeds to the steps for U-SH-PSI using $\mathtt{msg}_1$ and the upper bound, $U' := (U - |\mathcal{C}_L|)$, or aborts, otherwise. *Client* obtains $I_1 := \mathcal{C}_L \cap \mathcal{S}$ from the response for AD-PSI (denoted by $\mathtt{msg}_2$ in Fig. 7), and $I_2 := \mathcal{C}^* \cap \mathcal{S}$ from the one for U-SH-PSI (denoted by $\mathtt{msg}_3$ in Fig. 7), which are combined to the final result, $I := I_1 \cup I_2$.

The security and efficiency of the idea above rely on the ones of underlying AD-PSI and U-SH-PSI protocols. The AD-PSI phase guarantees that $\mathcal{C}$ satisfies the lower bound L. Although there is no duplicate check in the U-SH-PSI phase, *Client* does not have the motivation for duplicating the elements because *Client* can learn the result only when $|\mathcal{C}^*| \leq U'$ (i.e., duplicates limit *Client* more, especially when $|\mathcal{C}|$ is close to U).

7 Authorized PSI with Element Distinctness

So far, we have seen multiple PSI and its variant protocols that check the duplicity of input values. However, as noted in Section 1, malicious *Client* can still bypass these duplicity checks by generating random inputs instead of duplicating valid inputs. And what is the meaning of "valid" inputs? To examine

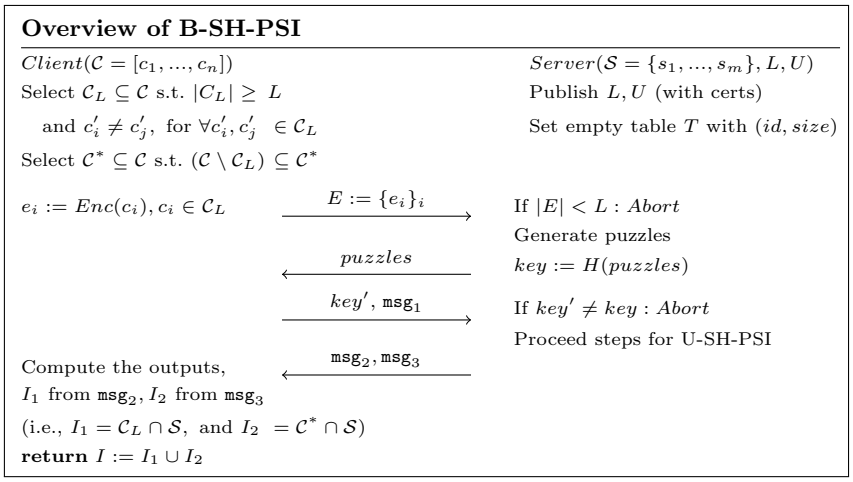

Fig. 7. Idea of B-SH-PSI with input bound $[L, U]$. $\texttt{msg}_1$ and $\texttt{msg}_3$ denote the first and responding messages for the U-SH-PSI protocol, whereas the others denote the messages for the alternative AD-PSI-puzzle protocol in Fig. 5

if *Client* uses valid inputs, including a trusted third party (TTP) who signs on valid inputs and later audits and punishes any invalid inputs is inevitable. i.e., Authorized PSI (APSI) that not only checks the element distinctness but also the validity of the input values. This section presents two versions of APSI: (v1) *stateful APSI*, where TTP tracks *Client* input values, and (v2) *stateless APSI*, where TTP does not save/track any information about *Client* input values.

7.1 AD-APSI Definition

Adopting the definitions of APSI from the related work [19,20,74,77] and referring to the definitions of general two-party computation from [28,34], secure AD-APSI can be defined as below. Let $REAL^{\Pi}_{\mathcal{A}(z),P}(C, S, \lambda)$ be the output of honest party and the adversary $\mathcal{A}$ corrupting P (either *Client* or *Server*) after a real execution of an AD-APSI protocol Π, where *Client* has input (potentially multi)set $\mathcal{C}$, *Server* has input set $\mathcal{S}$, $\mathcal{A}$ has auxiliary input z, and the security parameter is λ. Let $IDEAL^{\mathcal{F}}_{Sim(z),P}(\mathcal{C}, \mathcal{S}, \lambda)$ be the analogous distribution in an ideal execution with a trusted party who computes the ideal functionality $\mathcal{F}$ defined below.

Definition 2 (All-Distinct Authorized PSI (AD-APSI)). *is a tuple of three algorithms: {Setup, Authorize, Interaction}, where*

- *Setup: an algorithm selecting global/public parameters;*
- *Authorize: a protocol between Client and TTP resulting in Client committing to its input, $\mathcal{C} = [c_1, ..., c_n]$, and TTP issuing authorizations, one for each element of $\mathcal{C}$; and*

– *Interaction: a protocol between Client and Server on respective inputs: a (multi)set $\mathcal{C}$ and a set $\mathcal{S}$, resulting in Client obtaining the intersection of two inputs;*

An AD-APSI scheme satisfies the following properties:

– *Correctness: At the end of Interaction, Client outputs the exact intersection of two inputs, only when the elements in $\mathcal{C}$ are all distinct and authorized by TTP. Otherwise, Client outputs $\perp$;*
– *Server Privacy: Client learns no information about the subset of $\mathcal{S}$ that is not in the intersection, except its size. More formally, an AD-APSI scheme securely realizes the server privacy in the presence of malicious adversaries corrupting Client if for every real-world adversary $\mathcal{A}$, there exists a simulator Sim such that, for every $\mathcal{C}$, $\mathcal{S}$, and auxiliary input z,*

$$\{REAL^{\Pi}_{\mathcal{A}(z),Client}(\mathcal{C},\mathcal{S},\lambda)\}_{\lambda} \overset{c}{\approx} \{IDEAL^{\mathcal{F}}_{Sim(z),Client}(\mathcal{C},\mathcal{S},\lambda)\}_{\lambda}$$

– *Client Privacy: Server learns no information about Client input elements, except its size, authorization status, and element distinctness. More formally, an AD-APSI scheme securely realizes the client privacy in the presence of malicious adversaries corrupting Server if for every real-world adversary $\mathcal{A}$, there exists a simulator Sim such that, for every $\mathcal{C}$, $\mathcal{S}$, and z,*

$$\{REAL^{\Pi}_{\mathcal{A}(z),Server}(\mathcal{C},\mathcal{S},\lambda)\}_{\lambda} \overset{c}{\approx} \{IDEAL^{\mathcal{F}}_{Sim(z),Server}(\mathcal{C},\mathcal{S},\lambda)\}_{\lambda}$$

where the ideal functionality $\mathcal{F}$ is defined as follows:

– **`Authorize`** *: ($\mathcal{F}$ forwards the messages between Client and TTP and remembers the authorized elements for Client)*
  1. *Wait for an authorization request from Client, requesting TTP to authorize an element c*
  2. *Forward the request to TTP who either accepts or rejects it*
  3. *If TTP accepts, it forwards the messages from TTP to Client and remembers that TTP has authorized c for Client. Otherwise, it replies* **`abort`** *to Client*
– **`Interaction`** *: ($\mathcal{F}$ receives input elements from Client and Server and outputs the intersection to Client, only when Client inputs are all distinct and authorized, while giving Client input size and verification result (for authorization and duplication) to Server)*
  1. *Wait for an input (multi)set $\mathcal{C} = [c_1, .., c_n]$ from Client*
  2. *Wait for an input set $\mathcal{S} = \{s_1, ..., s_m\}$ from Server*
  3. *While sending $|\mathcal{C}|$ to Server, send* **`abort`** *to Client if $\mathcal{C}$ includes (1) any unauthorized element, or (2) duplicated elements. Otherwise, compute the intersection of $\mathcal{C}$ and $\mathcal{S}$ and send $(|\mathcal{S}|, \mathcal{C} \cap \mathcal{S})$ to Client. It also sends b to Server, where b is the result(s) for verifying the existence of (1) (and (2) in stateless version) above with their cardinality(ies).*

For clear notation, we denote the functionalities above as $\mathcal{F}_{\texttt{Auth}}$ and $\mathcal{F}_{\cap}$.

7.2 AD-APSI Construction

The main idea is from the double spending detection in [11]. i.e., TTP first divides each input value into two factors, where these factors are not revealed to anyone except *Client*. For the stateful TTP, the factors can be computed by choosing a random value in $\mathbb{Z}_p^*$ as the first factor and calculating the rest. For the stateless TTP, the first factor is computed so that it is unique per element value, e.g., with a pseudo-random function (under TTP's secret key) for each element in $\mathcal{C}$, and the second factor is calculated by dividing the element with the first factor. Then, the TTP signs a message such that it can be easily re-computed by a third party while not revealing each factor so that anyone with the message can verify the signature with the TTP's public key.

In the online phase, *Client* sends $G_{\mathcal{C}}$, the pre-computed values that effectively hide two factors for each input value, and Σ, all the signatures given by TTP. Then, *Server* first verifies each signature with a newly-computed message with $G_{\mathcal{C}}$ and aborts if any signature verification fails. In the stateless TTP version, *Server* additionally checks if there are any same elements in $G_{\mathcal{C}}$ and aborts if so. If all passed, *Server* now proceeds to the intersection computation phase, similar to the other PSI protocols. i.e., It first chooses a random number R to hide its elements, and computes t_j, which can be also pre-computed. Then, *Server* exponentiates each $g^{e_{i,1}}$ to the same R so that *Client* can compute the same form, compare, and obtain the intersection result. Fig. 8 shows the aforementioned offline and online phases with stateful and stateless TTP options, with an example form of message, $m_i := H(g^{e_{i,1}}, g^{e_{i,2}})$ for each $c_i = e_{i,1} * e_{i,2} \pmod p$ in $\mathcal{C}$. In the offline phase, *Client* can pre-compute $G_{\mathcal{C}}$ once it receives all the factors from TTP, or TTP can also send $G_{\mathcal{C}}$ along with the others, which is the trade-off between communication cost and *Client*'s computation cost.

7.3 Security Analysis

Theorem 3. *The protocol described in Section 7.2 is a secure AD-APSI scheme, satisfying Definition 2 in ROM.*

(*Sketch proof*) Duplicated elements get caught by either the stateful TTP or *Server* (when TTP is stateless) because the $G_{\mathcal{C}}$ element is always the same for an input value. Also, *Server* detects if any unauthorized elements are used via signature verification. For the honest $\mathcal{C}$ (i.e., with all authorized and distinct elements), *Client* outputs the exact intersection of $\mathcal{C}$ and $\mathcal{S}$ because, for $c_i = s_j$,

$$d_i := H(\hat{e_i}^{e_{i,2}}) = H((g^{e_{i,1}R})^{e_{i,2}}) = H(g^{c_i R}) = H(g^{s_j R}) = t_j$$

Server responds only when all signature verification and duplication checks are passed, and t_j's do not reveal any information about s_j by randomizing with R and hashing with a cryptographic hash function. Lastly, client privacy depends on both computational Diffie-Hellman problem and the secure signature scheme. □

We provide the full security analysis of Theorem 3 in Appendix C.

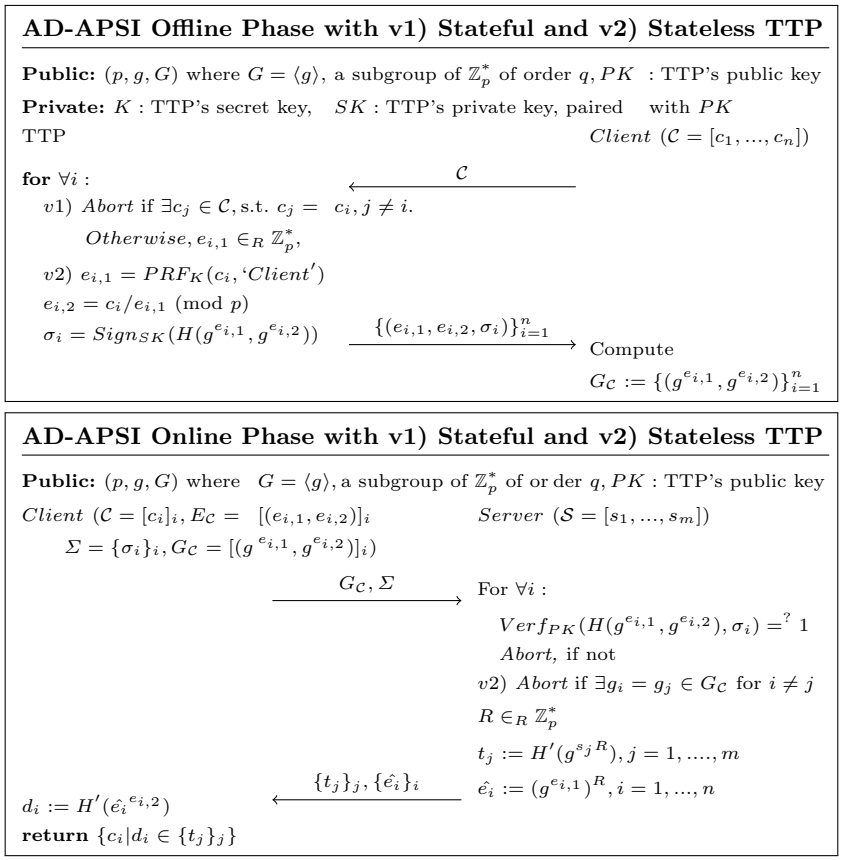

Fig. 8. All-Distinct Authorized PSI (AD-APSI) scheme. PRF is a pseudo-random function, $(Sign, Verf)$ is a digital signature scheme over $\{0,1\}^\kappa$, and H, H' are cryptographic hash functions.

8 Conclusion

This paper investigated two malicious behaviors for private input – using duplicated and spurious elements – and suggested checking the input validity in PSI and its variants. We proposed a PoED construction, PoED-puzzle, using PHE and a generalized version of the two billiard balls problem, and using it as a building block, we introduced a new PSI variant, AD-PSI, with a formal definition. We presented an AD-PSI protocol based on PoED-puzzle and analyzed its security according to the definition. We also provided ideas of three AD-PSI variants, AD-PSI-CA, AD-PSI-X, and AD-PSI-DT, where duplicates cause more information leakage without PoED, and proposed a B-SH-PSI scheme with both upper and lower bounds on the client input size, using AD-PSI and U-SH-PSI. Lastly, we formalized the definition of AD-APSI that assesses both misbehaviors on client input and suggested a construction with its security analysis.

References

1. Palisade homomorphic encryption software library. Online: `https://palisade-crypto.org/` (2017)
2. Lattigo v4. Online: `https://github.com/tuneinsight/lattigo` (Aug 2022), ePFL-LDS, Tune Insight SA
3. Ateniese, G., De Cristofaro, E., Tsudik, G.: (if) size matters: size-hiding private set intersection. In: International Workshop on Public Key Cryptography. pp. 156–173. Springer Berlin Heidelberg (2011)
4. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Advances in Cryptology — CRYPTO' 92. pp. 390–420 (1993)
5. Blanton, M., Aguiar, E.: Private and oblivious set and multiset operations. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS) (2012)
6. Bradley, T., Faber, S., Tsudik, G.: Bounded size-hiding private set intersection. In: International Conference on Security and Cryptography for Networks. pp. 449–467. Springer, Springer International Publishing (2016)
7. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Advances in Cryptology — CRYPTO '97. pp. 410–424 (1997)
8. Carpent, X., Faber, S., Sander, T., Tsudik, G.: Private set projections & variants. In: Proceedings of the 2017 on Workshop on Privacy in the Electronic Society (WPES '17). Association for Computing Machinery (2017)
9. Cerulli, A., De Cristofaro, E., Soriente, C.: Nothing refreshes like a repsi: Reactive private set intersection. In: Applied Cryptography and Network Security. pp. 280–300 (2018)
10. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious prf. In: Advances in Cryptology – CRYPTO 2020. pp. 34–63. Springer International Publishing (2020)
11. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Advances in Cryptology – CRYPTO' 88. pp. 319–327. Springer New York (1990)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: Fast fully homomorphic encryption library over the torus. `https://github.com/tfhe/tfhe.(2017)`, accessed: 2022-01-31
13. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. In: Security and Cryptography for Networks (2018)
14. Clarkson, J.B.: Dense probabilistic encryption. In: Proceedings of the Workshop on Selected Areas of Cryptography. pp. 120–128 (1994)
15. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Advances in Cryptology – EUROCRYPT '97. pp. 103–118. Springer Berlin Heidelberg (1997)
16. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Applied Cryptography and Network Security. pp. 125–142 (2009)
17. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: Public Key Cryptography. pp. 119–136 (2001)
18. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: Cryptology and Network Security, pp. 218–231. Springer (2012)
19. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Advances in Cryptology - ASIACRYPT 2010. pp. 213–231 (2010)

20. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Financial Cryptography and Data Security. pp. 143–159 (2010)
21. De Cristofaro, E., Tsudik, G.: Experimenting with fast private set intersection. In: Trust and Trustworthy Computing. pp. 55–73 (2012)
22. Debnath, S.K., Dutta, R.: Secure and efficient private set intersection cardinality using bloom filter. In: Information Security. pp. 209–226. Springer International Publishing (2015)
23. Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: PIR-PSI: Scaling private contact discovery. Proceedings on Privacy Enhancing Technologies pp. 159–178 (2018)
24. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: CCS (2013)
25. Duong, T., Phan, D.H., Trieu, N.: Catalic: Delegated psi cardinality with applications to contact tracing. In: Advances in Cryptology – ASIACRYPT 2020. pp. 870–899. Springer International Publishing (2020)
26. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
27. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Advances in Cryptology. pp. 10–18 (1985)
28. Evans, D., Kolesnikov, V., Rosulek, M.: A Pragmatic Introduction to Secure Multi-Party Computation (2018)
29. Fiege, U., Fiat, A., Shamir, A.: Zero knowledge proofs of identity. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. pp. 210–217. STOC '87, Association for Computing Machinery (1987)
30. Freedman, M.J., Hazay, C., Nissim, K., Pinkas, B.: Efficient set intersection with simulation-based security. Journal of Cryptology **29**(1), 115–155 (2016)
31. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Theory of Cryptography. pp. 303–324. Springer Berlin Heidelberg (2005)
32. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Advances in Cryptology - EUROCRYPT 2004. pp. 1–19 (2004)
33. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: Advances in Cryptology – EUROCRYPT 2019. pp. 154–185. Springer International Publishing (2019)
34. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press (2004)
35. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. pp. 291–304. STOC '85 (1985)
36. Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing. pp. 365–377. STOC '82 (1982)
37. Gouert, C., Mouris, D., Tsoutsos, N.G.: Sok: New insights into fully homomorphic encryption libraries via standardized benchmarks. Proc. Priv. Enhancing Technol. **2023**(3), 154–172 (2023)
38. Hagen, C., Weinert, C., Sendner, C., Dmitrienko, A., Schneider, T.: All the numbers are us: Large-scale abuse of contact discovery in mobile messengers. IACR Cryptology ePrint Archive p. 1119 (2020)
39. Hallgren, P., Orlandi, C., Sabelfeld, A.: Privatepool: Privacy-preserving ridesharing. In: IEEE 30th Computer Security Foundations Symposium (CSF) (2017)

40. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Theory of Cryptography. pp. 155–175 (2008)
41. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Public Key Cryptography – PKC 2010. pp. 312–331 (2010)
42. Hemenway Falk, B., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions. In: Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society. pp. 14–25. WPES'19, Association for Computing Machinery (2019)
43. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS. ISOC (2012)
44. Huberman, B., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: ACM Conference on Electronic Commerce (1999)
45. Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Saxena, S., Seth, K., Raykova, M., Shanahan, D., Yung, M.: On deploying secure computing: Private intersection-sum-with-cardinality. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 370–389 (2020)
46. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In: Theory of Cryptography. pp. 577–594 (2009)
47. Kajita, K., Ohtake, G.: Private set intersection for viewing history with efficient data matching. In: HCI International 2022 Posters. pp. 498–505. Springer International Publishing (2022)
48. Kawachi, A., Tanaka, K., Xagawa, K.: Multi-bit cryptosystems based on lattice problems. In: Public Key Cryptography – PKC 2007. pp. 315–329 (2007)
49. Kissner, L., Song, D.: Privacy-preserving set operations. In: Advances in Cryptology – CRYPTO 2005. pp. 241–257 (2005)
50. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 818–829. CCS, Association for Computing Machinery (2016)
51. Kolesnikov, V., Rosulek, M., Trieu, N.: SWiM: Secure wildcard pattern matching from ot extension. In: Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 – March 2, 2018, Revised Selected Papers. pp. 222–240. Springer-Verlag (2018)
52. Kulshrestha, A., Mayer, J.: Estimating incidental collection in foreign intelligence surveillance: Large-Scale multiparty private set intersection with union and sum. In: 31st USENIX Security Symposium. pp. 1705–1722. USENIX Association (2022)
53. Laine, K., Chen, H., Player, R.: Simple encrypted arithmetic library (seal). `https://github.com/microsoft/SEAL.(2017)`, accessed: 2022-01-31
54. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. Journal of Cryptology **28**(2), 312–350 (Apr 2015)
55. Ma, J.P.K., Chow, S.S.M.: Secure-computation-friendly private set intersection from oblivious compact graph evaluation. ASIA CCS '22 (2022)
56. Meadows, C.A.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. 1986 IEEE Symposium on Security and Privacy pp. 134–134 (1986)
57. Miao, P., Patel, S., Raykova, M., Seth, K., Yung, M.: Two-sided malicious security for private intersection-sum with cardinality. In: Advances in Cryptology – CRYPTO 2020. pp. 3–33. Springer International Publishing (2020)

58. Naccache, D., Stern, J.: A new public key cryptosystem based on higher residues. In: Proceedings of the 5th ACM Conference on Computer and Communications Security. pp. 59–66. CCS '98 (1998)
59. Nagaraja, S., Mittal, P., Hong, C., Caesar, M., Borisov, N.: BotGrep: Finding bots with structured graph analysis. In: Usenix Security (2010)
60. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: CCS '21: ACM SIGSAC Conference on Computer and Communications Security. pp. 1151–1165 (2021)
61. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Advances in Cryptology — EUROCRYPT'98. pp. 308–318 (1998)
62. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology — EUROCRYPT '99. pp. 223–238. Springer Berlin Heidelberg (1999)
63. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse ot extension. In: Advances in Cryptology – CRYPTO 2019. pp. 401–431. Springer International Publishing (2019)
64. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Psi from paxos: Fast, malicious private set intersection. In: Advances in Cryptology – EUROCRYPT 2020. pp. 739–767. Springer International Publishing (2020)
65. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Transactions on Privacy and Security (TOPS) **21**, 1–35 (2016)
66. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: Proceedings of the 24th USENIX Conference on Security Symposium. pp. 515–530. SEC'15, USENIX Association (2015)
67. Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based psi with linear communication. In: Advances in Cryptology – EUROCRYPT 2019. pp. 122–153. Springer International Publishing (2019)
68. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based psi via cuckoo hashing. In: Advances in Cryptology – EUROCRYPT 2018. pp. 125–157. Springer International Publishing (2018)
69. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 797–812. USENIX Association (2014)
70. Rindal, P., Schoppmann, P.: Vole-psi: Fast oprf and circuit-psi from vector-ole. In: Advances in Cryptology – EUROCRYPT 2021. pp. 901–930. Springer International Publishing (2021)
71. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
72. Sathya Narayanan, G., Aishwarya, T., Agrawal, A., Patra, A., Choudhary, A., Pandu Rangan, C.: Multi Party Distributed Private Matching, Set Disjointness and Cardinality of Set Intersection with Information Theoretic Security. In: CANS (2009)
73. Shamir, A.: On the power of commutativity in cryptography. In: Automata, Languages and Programming. pp. 582–595 (1980)
74. Stefanov, E., Shi, E., Song, D.: Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In: Public Key Cryptography – PKC (2012)
75. Takeshita, J., Karl, R., Mohammed, A., Striegel, A., Jung, T.: Provably secure contact tracing with conditional private set intersection. In: Security and Privacy in Communication Networks. pp. 352–373. Springer International Publishing (2021)

76. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security **13**(4) (2005)
77. Wen, Y., Gong, Z., Huang, Z., Qiu, W.: A new efficient authorized private set intersection protocol from Schnorr signature and its applications. Cluster Computing **21**(1), 287–297 (Mar 2018)
78. Zhao, Y., Chow, S.: Are you the one to share? secret transfer with access structure. Proceedings on Privacy Enhancing Technologies – PETS'17 (2017)
79. Zhao, Y., Chow, S.S.: Can you find the one for me? In: Proceedings of the 2018 Workshop on Privacy in the Electronic Society. pp. 54–65. WPES'18, Association for Computing Machinery (2018)

Appendix A Security Proof for AD-PSI-puzzle

Theorem 2. *The protocol described in Figure 3 is a secure AD-PSI scheme, satisfying the Definition 1 in ROM.*

Proof. **Correctness:** For an honest *Client* with distinct input elements, there exists only one permutation π_k such that $\pi_k(\mathcal{C}) = Dec(E_k)$. This is because the decryption results remain the same after the re-randomization due to the homomorphic property of the ElGamal scheme on multiplication. Thus, honest *Client* derives the same permutations as the ones *Server* used and the derived key', the hash of these permutations, is equal to key. *Client* gets the *Server*'s tags, $\{t'_j = H'(s_j^R)\}_j$ by symmetric-decrypting each of them. Since $d_i = H'(Dec(\hat{e}_i)) = H'(Dec(e_i^R)) = H'(c_i^R)$, with overwhelming probability (due to the collision resistance of the cryptographic hash functions), we have $t'_j = d_i \Leftrightarrow s_j^R = c_i^R \Leftrightarrow s_j = c_i$. Therefore, *Client* obtains correct intersection $\{c_i\}_{i\in I}$, with $I := \{i \mid d_i \in \{t'_1, ..., t'_w\}\}$ with distinct input elements.

On the other hand, we show that any clients with duplicated elements in their input cannot obtain the intersection with overwhelming probability. Let's look at the case where a corrupted *Client* has the highest probability of successfully cheating, i.e., with $\mathcal{C} = [c_1, ..., c_n]$ with $(n-1)$ distinct items and one duplicate. Without loss of generality, let's say $c_1 = c_2$, and the others are all distinct. In this case, the probability that *Client* obtains the intersection is the same as the probability that *Client* guesses λ correct permutations, so $2^{-\lambda}$, which is negligible with a sufficiently large λ.

Client Privacy: Assume that *Server* is corrupted. Showing the client privacy is relatively easy: it only sends to *Server* the encryption of the element in its set. Assuming two input sets with the same sizes, if the adversary corrupting *Server* can distinguish whether *Client* used which set as an input, then it can be used for IND-CPA of the ElGamal encryption system. Since it is well-known that the ElGamal encryption system is semantically secure [27] assuming the hardness of the decisional Diffie-Hellman problem which is reduced to DLP, the adversary cannot distinguish which set is used as well as learn anything about the *Client*'s set elements.

Server Privacy: Assume that *Client* is corrupted, denoted by *Client**. To claim server privacy, we need to show that the *Client*'s view can be efficiently

simulated by a PPT algorithm $\texttt{Sim}_{\mathcal{C}}$. The simulator $\texttt{Sim}_{\mathcal{C}}$ can be constructed as follows:

1. $\texttt{Sim}_{\mathcal{C}}$ builds two tables $T = ((\pi_1, ..., \pi_\lambda), k)$ and $T' = (m, h')$ to answer the H and H' queries, respectively.
2. After getting the message $(\mathbb{G}, p, g, h)$ and $\{e_i\}_{i=1}^n$ of a corrupted real-world client $Client^*$, $\texttt{Sim}_{\mathcal{C}}$ picks λ random permutations from $\mathcal{P}_n$ and n random numbers $r_{i,j}$ from $\mathbb{Z}$ where $i = 1, ..., n$ for each $j = 1, ..., \lambda$. Then, $\texttt{Sim}_{\mathcal{C}}$ re-randomizes and shuffles $\{e_i\}_{i=1}^n$ by multiplying $(g^{r_{i,j}}, h^{r_{i,j}})$ to each e_i's for $i = 1, ..., n$, say $e_{i,j}$, and applying the permutation π_j to $\{e_{i,j}\}_{i=1}^n$, for each j, say $E_j := \pi_j(e_{1,j}, ..., e_{n,j})$.
3. Also, $\texttt{Sim}_{\mathcal{C}}$ picks random $R \in \mathbb{Z}$, and exponentiates each component of e_i's, i.e., $\hat{e_i} := e_i^R = (e_{i,1}^R, e_{i,2}^R)$ for $i = 1, ..., n$. $\texttt{Sim}_{\mathcal{C}}$ also picks m random elements from $\mathcal{M}$, say $u_1, ..., u_m$.
4. $\texttt{Sim}_{\mathcal{C}}$ encrypts each u_j using $SymE$ with the key, $key := H(\pi_1, ..., \pi_\lambda)$, i.e., $t_j := SymE(key, u_j)$, and replies $\{E_k\}_{k=1}^\lambda$, $\{\hat{e_i}\}_{i=1}^n$, $\{t_j\}_{j=1}^m$ to $Client^*$.
5. Then, $\texttt{Sim}_{\mathcal{C}}$ answers the H, H' queries as follows:
   - For each query $(\pi_1, ..., \pi_\lambda)$ to H, $\texttt{Sim}_{\mathcal{C}}$ checks if $\exists\ ((\pi_1, ..., \pi_\lambda), key) \in T$ and returns key if so. Otherwise, $\texttt{Sim}_{\mathcal{C}}$ picks a random $key \in_R \mathcal{K}$ and checks if $\exists((\pi'_1, ..., \pi'_\lambda), key') \in T$ such that $key' = key$. If so, output $\texttt{fail}_1$ and aborts. Otherwise, it adds $((\pi_1, ..., \pi_\lambda), key)$ to T and returns key to $Client^*$ as $H(\pi_1, ..., \pi_\lambda)$.
   - For each query m to H', $\texttt{Sim}_{\mathcal{C}}$ checks if $(m, h') \in T'$. If so, $\texttt{Sim}_{\mathcal{C}}$ returns h'. Otherwise, $\texttt{Sim}_{\mathcal{C}}$ picks a random $h' \in_R \mathcal{M}$, and checks if $\exists(m'', h'')$ in T' where $h'' = h'$ and $m'' \neq m$. If so, $\texttt{Sim}_{\mathcal{C}}$ outputs $\texttt{fail}_2$ and aborts. Otherwise, $\texttt{Sim}_{\mathcal{C}}$ adds (m, h') to T' and returns h' to $Client^*$ as $H'(m)$.

This finishes the construction $\texttt{Sim}_{\mathcal{C}}$. The ideal-world server $\overline{Server}$ that interacts with the ideal function f, which answers the queries from $\texttt{Sim}_{\mathcal{C}}$ as the ideal-world client $\overline{Client}$, gets $\perp$ from f, and the real-world server $Server$ which interacts with $Client^*$ in the real protocol also outputs $\perp$. We now argue that $Client^*$'s view in the interaction with $Server$ and with $\texttt{Sim}_{\mathcal{C}}$ constructed as above are indistinguishable. The $Client^*$'s view is different only if one of the following happens:

- $\texttt{fail}_1$ **occurs:** This happens if $\exists(Q' := (\pi'_1, ..., \pi'_\lambda), key')$ such that $key' = key$ but $Q' \neq Q$ existing in T, for a randomly chosen key from $\mathcal{K}$ for the query $Q = (\pi_1, ..., \pi_\lambda)$ to H. This means a collision of H is found, i.e., $H(Q) = H(Q')$ where $Q \neq Q'$. This occurs with negligible probability by the collision resistance of H.
- $\texttt{fail}_2$ **occurs:** This happens if there exists the entry (m'', h'') such that $h'' = h'$ but $m'' \neq m$ existing in T', for a randomly chosen h' from $\mathcal{M}$ for the query m to H'. This means a collision of H' is found, i.e., $H'(m'') = H'(m)$ where $m'' \neq m$. This happens with negligible probability due to the collision resistance of H'.

Since all events above happen with negligible probability, $Client^*$'s views in the real protocol with the real-world server $Server$ can be efficiently simulated by $\texttt{Sim}_{\mathcal{C}}$ in the ideal world. □

Appendix B AD-PSI Variants

This section presents the figures of the protocols described in Section 5, AD-PSI-CA in Fig. 9, AD-PSI-X in Fig. 10, and AD-PSI-DT in Fig. 11, respectively.

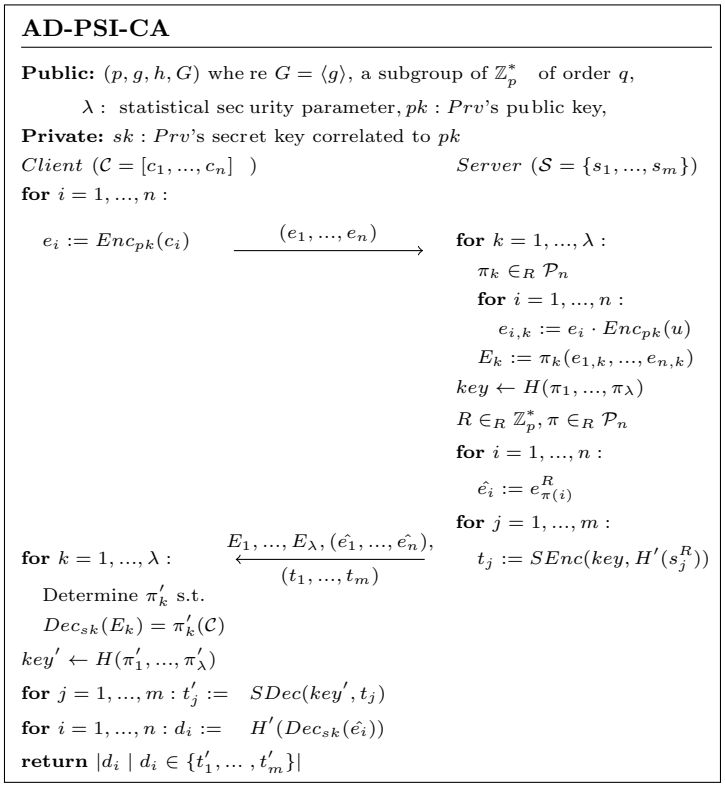

Fig. 9. AD-PSI-Cardinality (AD-PSI-CA) Protocol with same notation as Fig. 3

Appendix C Security Proof for AD-APSI

Theorem 3. *The protocol described in Section 7.2 is a secure AD-APSI scheme, satisfying Definition 2 in ROM.*

Proof. **Correctness:** For an honest *Client* with all authorized and distinct elements, the stateful TTP generates authentic signatures for each element so that *Server* can verify the signatures correctly. For the stateless TTP, instead of tracking all the input values of *Client*, TTP generates unique and deterministic factors of the input. Thus, *Server* can tell when *Client* uses duplicated elements as the corresponding elements in $G_{\mathcal{C}}$ are the same. When *Server*

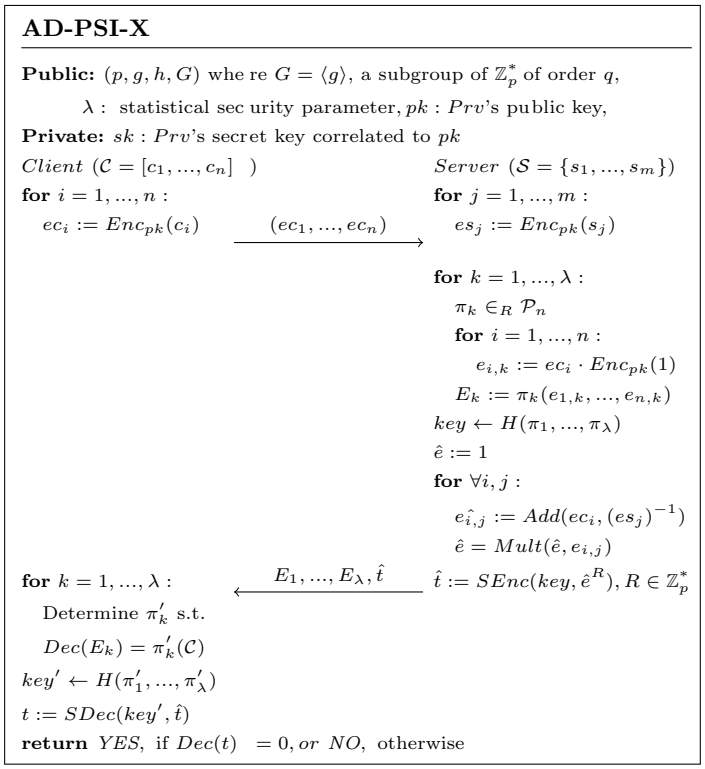

Fig. 10. AD-PSI-Existence (AD-PSI-X) Protocol with same notation as Fig. 3 except that (Enc, Dec) should be a FHE over G satisfying $Add(Enc(a), Enc(b)) = Enc(a+b)$ and $Mult(Enc(a), Enc(b)) = Enc(a * b)$.

replies, *Client* outputs the exact intersection of $\mathcal{C}$ and $\mathcal{S}$ because, for $c_i = s_j$, $d_i := H'(\hat{e_i}^{e_{i,2}}) = H'((g^{e_{i,1}R})^{e_{i,2}}) = H'(g^{c_iR}) = H'(g^{s_jR}) = t_j$. Therefore, duplicated elements in $\mathcal{C}$ are caught by either the stateful TTP or *Server* (when TTP is stateless), unauthorized (i.e., not signed by TTP) elements are caught by *Server*, and honest *Client* obtains the exact intersection of the two input sets.

For server and client privacy, we show that the distribution of protocol execution in the real world is computationally indistinguishable from the output from interaction with $\mathcal{F}$ in the ideal world, assuming the same corrupted party (either *Client* or *Server*). Since the interaction between *Server* and *Client* is during the online phase for `Interaction`, it is compared with $\mathcal{F}_\cap$ (recall Definition 2), assuming $\mathcal{C}$ is authorized with $\mathcal{F}_{\texttt{Auth}}$.

Server Privacy: Assume that *Client* is corrupted, denoted by *Client**. We show that the distribution of *Client** outputs in the real world can be efficiently simulated by a PPT $\texttt{Sim}_\mathcal{C}$ constructed as below.

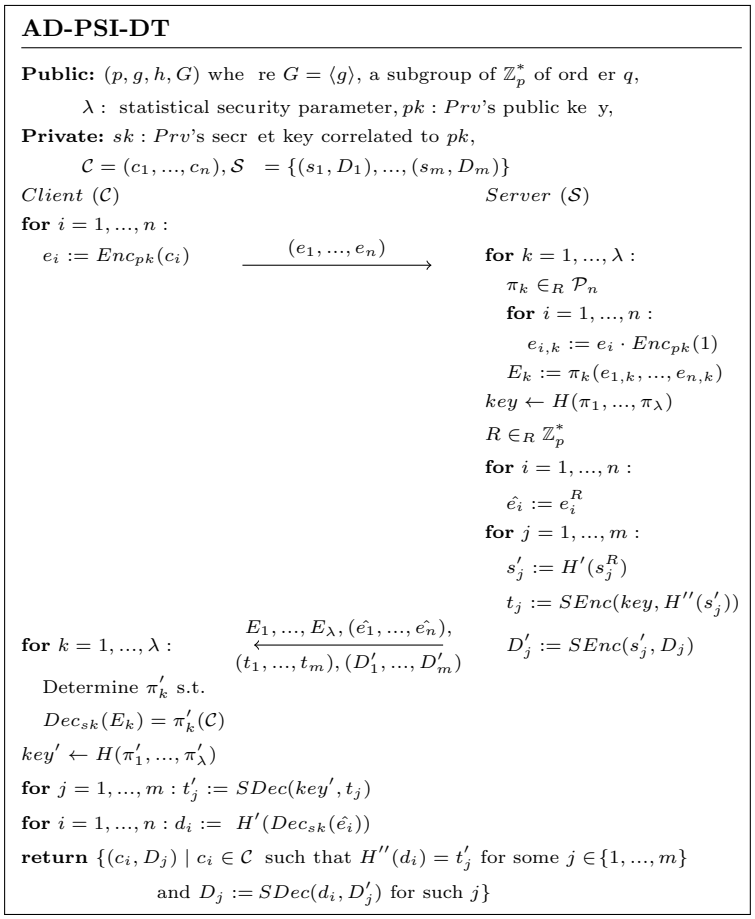

Fig. 11. AD-PSI-Data Transfer (AD-PSI-DT) Protocol with same notation as Fig. 3. Additionally, H'' is a one-way function that maps k-bit messages to k-bit messages.

1. $\mathtt{Sim}_{\mathcal{C}}$ builds two tables $T_1 = ((m_1, m_2), h)$ and $T_2 = (m, h')$ to answer the H and H' queries, respectively.
2. After getting the messages $G_{\mathcal{C}} := \{g_{i,1}, g_{i,2}\}_i$ and Σ of a corrupted real-world client, $Client^*$, $\mathtt{Sim}_{\mathcal{C}}$ verifies the received signatures with respect to each $H(g_{i,1}, g_{i,2})$ via $Verf$ and TTP's public key. If any of those fails, it aborts. (Likewise, for the stateless version, $\mathtt{Sim}_{\mathcal{C}}$ also checks the duplicates in $G_{\mathcal{C}}$ and aborts if any.
3. Otherwise, $\mathtt{Sim}_{\mathcal{C}}$ picks m random elements, $u_1, ..., u_m$, in G and computes $t_j := H'(u_j)$ for $j = 1, ..., m$. It also picks a random R, computes $\{\hat{e_i} = g_{i,1}^R\}_i$, and replies $\{\hat{e_i}\}_i$ and $\{t_j\}_m$ to $Client^*$.
4. For each query to H and H', $\mathtt{Sim}_{\mathcal{C}}$ answers as follows:

- For each query (m_1, m_2) to H, $\texttt{Sim}_{\mathcal{C}}$ checks if $exists((m_1, m_2), h) \in T_1$ and returns h if so. Otherwise, $\texttt{Sim}_{\mathcal{C}}$ picks a random h (from the same space as other values) and checks if $exists((\tilde{m_1}, \tilde{m_2}), \tilde{h}) \in T_1$ such that $h = \tilde{h}$. If so, output $\texttt{fail}_1$ and abort. Otherwise, it adds $((m_1, m_2), h)$ to T_1 and returns h to $Client^*$ as $H((m_1, m_2))$.
- For each query m to H', $\texttt{Sim}_{\mathcal{C}}$ checks if $exists(m, h') \in T_2$ and returns h' if so. Otherwise, $\texttt{Sim}_{\mathcal{C}}$ picks a random h' (from the same space as other values) and checks if $exists(\tilde{m}, \tilde{h}) \in T_2$ such that $h' = \tilde{h}$. If so, output $\texttt{fail}_2$ and abort. Otherwise, it adds (m, h') to T_2 and returns h' to $Client^*$ as $H'(m)$.

This finishes the $\texttt{Sim}_{\mathcal{C}}$ construction. The $Client^*$'s view in the interaction with $\texttt{Sim}_{\mathcal{C}}$ above is different from the view in the real-world interaction with the real server, $Server$, only if $\texttt{fail}_1$ or $\texttt{fail}_2$ happen. However, due to the collision resistance property of cryptographic hash functions H, H', they occur with negligible probability. Thus, $Client^*$'s view when interacting with $Server$ can be efficiently simulated by $\texttt{Sim}_{\mathcal{C}}$ in the ideal world. For the outputs, the ideal-world server $\overline{Server}$ that interacts with $\mathcal{F}_{\cap}$, which answers the queries from $\texttt{Sim}_{\mathcal{C}}$ as the ideal-world $Client$, $\overline{Client}$, receives $(|\mathcal{C}, b)$ from $\mathcal{F}_{\cap}$. On the other hand, the real-world (honest) server $Server$ that interacts with $Client^*$ in the real protocol also outputs (learns) $(|\mathcal{C}, b)$. i.e., $\overline{Server}$ interacting with $\texttt{Sim}_{\mathcal{C}}$ and $Server$ interacting with $Client^*$ yield the identical outputs.

Client Privacy: Similarly, now we assume a corrupted server, $Server^*$, and show that $Server^*$'s view in the real world can be efficiently simulated by a PPT simulator, $\texttt{Sim}_{\mathcal{S}}$, constructed as below. Intuitively, $\texttt{Sim}_{\mathcal{S}}$ sits between $\mathcal{F}_{\cap}$ and $Server^*$, and interacts with both in such a way that $Server^*$ is unable to distinguish protocol runs with $\texttt{Sim}_{\mathcal{S}}$ from real-world protocol runs with $Client$. First, $\texttt{Sim}_{\mathcal{S}}$ builds tables T_1 and T_2, and answers similarly to $\texttt{Sim}_{\mathcal{C}}$ above for H and H' queries. Then for inputs, since $Client$ and TTP communicate in the offline phase before the online phase, the authorized elements for $Client$ are made available to $\texttt{Sim}_{\mathcal{S}}$. $\texttt{Sim}_{\mathcal{S}}$ uses a subset of authorized elements during the simulation to emulate $Client$'s behavior. If $Server^*$ does not abort and reply $(\{t_j\}_j, \{\hat{e_i}\}_i)$, $\texttt{Sim}_{\mathcal{S}}$ checks if $\hat{e_i}^{e_{i,2}} \in \{t_j\}_j$. If so, $\texttt{Sim}_{\mathcal{S}}$ adds $s_i := e_{i,1}e_{i,2} \pmod p$ in $\mathcal{S}$, and otherwise, adds a dummy element in $\mathbb{Z}_p^*$ in $\mathcal{S}$. Then, $\texttt{Sim}_{\mathcal{S}}$ plays the role of the ideal-world server, $\overline{Server}$, using $\mathcal{S}$ to respond to the queries from the ideal client ($\overline{Client}$). Since $\texttt{Sim}_{\mathcal{S}}$ uses the authorized inputs, $Server^*$'s view in the interaction with $\texttt{Sim}_{\mathcal{S}}$ is identical to the view in the interaction with honest $Client$ in the real world. Also, the output of the ideal-world client $\overline{Client}$ that interacts with $\mathcal{F}_{\cap}$, which answers the queries from $\texttt{Sim}_{\mathcal{S}}$ as the ideal-world $Server$, $\overline{Server}$, is identical to the output of the real-world $Client$ interacting with $Server^*$ as $(|\mathcal{S}, \mathcal{C} \cap \mathcal{S})$, only when all inputs in $\mathcal{C}$ are authorized and distinct. □