Can you find the one for me?
Privacy-Preserving Matchmaking via Threshold PSI

Yongjun Zhao and Sherman S.M. Chow

Department of Information Engineering
The Chinese University of Hong Kong, Hong Kong
{zy113, sherman}@ie.cuhk.edu.hk

Abstract. Private set-intersection (PSI) allows a client to only learn the intersection between his/her set C and the set S of another party, while this latter party learns nothing. We aim to enhance PSI in different dimensions, motivated by the use cases of increasingly popular online matchmaking — Meeting "the one" who possesses *all* desired qualities and *free from any* undesirable attributes may be a bit idealistic. In this paper, we realize *over-* (resp. *below-*) threshold PSI, such that the client learns the intersection (or other auxiliary private data) only when $|C \cap S| > t$ (resp. $\leq t$). The threshold corresponds to tunable criteria for (mis)matching, without marking all possible attributes as desired or not. In other words, the matching criteria are in a succinct form and the matching computation does not exhaust the whole universe of attributes. To the best of our knowledge, our constructions are the very first solution for these two open problems posed by Bradley *et al.* (SCN '16) and Zhao and Chow (PoPETS '17), without resorting to the asymptotically less efficient generic approach from garbled circuits.
Moreover, we consider an "outsourced" setting with a service provider coordinating the PSI execution, instead of having two strangers to be online simultaneously for running a highly-interactive PSI directly with each other. Outsourcing our protocols are arguably optimal — the two users perform $O(|C|)$ and $O(1)$ decryptions, for unlocking the private set C and the outcome of matching.

1 Introduction

In this big-data era, information is an asset. Sharing of information often leads to a win-win situation. The key issue is how to share selectively and strategically. People nowadays tend to share their information over the online social network (OSN). Usually, the sharing decision is based on whether the "subscribing user" has been admitted into a certain "circle" or not. The admission decision can be easy to make if we can rely on real-world friendship. Yet, people often reach out and expand their networks to enjoy the real benefits brought by OSN. That will be desirable if this decision can be made in a more systematic and intelligent way, *e.g.*, if a user possesses a sufficient number of common interests/attributes.

Private set intersection (PSI) is a handy cryptographic primitive which allows two parties P_1 and P_2, traditionally referred to as a client and a server, to compute the intersection of their respective private sets. For example, two companies can learn their common customers without sharing their databases. Yet, apart from hiding elements not in the set of the counter-party, PSI offers no more control. For example, if two companies do not share a high number of common customers, they may not bother to discuss any joint campaign, not to say revealing common customers to each other.

Recently, Zhao and Chow [45] initiated the study of PSI with access structure, *i.e.*, the client gets to know the intersection set only if its private set satisfies some policy specified by the server. As a special case, they consider threshold PSI which only reveals the intersection if its size is greater than a threshold agreed upon by both parties. While the vision of incorporating a sharing strategy to a vanilla PSI is great, the actual protocols realized by Zhao and Chow are a bit unsatisfactory. Without resorting to non-standard cryptographic assumptions, their design *always leaks* how many elements "contribute" to the satisfaction of the policy. Specifically, for threshold policy, they only achieved a weaker variant under the name of *threshold private set-intersection cardinality* (t-PSI-CA), which always leaks the size of the intersection to the client, regardless of whether it is greater than the threshold or not. They argue that this level of privacy protection is enough for applications such as online dating, where the matching criteria are sensitive, and revealing the degree of overlapping even in a mismatch is a nice feature. Nevertheless, threshold PSI protocol based on standard cryptographic assumptions remains open [45].

From the perspective of finding common interests, the "over threshold" policy discussed above appears to be a natural choice. Yet, considering matchmaking or access control in general, it is equally interesting to realize the complementary notion of *below-threshold private set-intersection.* To unify the notion, we rename the two functionalities above as $t^{\geq}$-PSI [45] and $t^{\leq}$-PSI [9] respectively.

The benefits of supporting both kinds of policy are apparent. For the dating application, using $t^{\geq}$-PSI alone only allows the search for desired quality. Users probably also want to match with others who do not possess a certain set of undesired attributes (*e.g.*, smoking, over-gregarious). With $t^{\leq}$-PSI, users can simply make sure that their number of occurrence is below an acceptable threshold instead of specifying every possible negated attribute (*e.g.*, non-smoking). Unfortunately, $t^{\leq}$-PSI is also posed as an open question [9].

1.1 Technical Overview

Despite the conceptual similarity, it is fair to say that $t^{\leq}$-PSI and $t^{\geq}$-PSI are two *different* problems. We have a weakened version of $t^{\geq}$-PSI (namely t-PSI-CA) [45], but the corresponding weakened form of $t^{\leq}$-PSI does not exist in the literature. To the best of our knowledge, there is no solution for these two problems (perhaps except the generic approach of using garbled circuits [30, 43]). The fundamental issue is that, $t^{\geq}$-PSI (resp. $t^{\leq}$-PSI) fall within the framework of private set-intersection with *monotone* (resp. *non-monotone*) access structure [45]. It is not clear how to construct non-monotone access structure from monotone one (and vice versa).

In this work, we unify the design of both protocols. We take an innovative approach to realize both kinds of threshold PSI protocol, without the deficiency of leaking the intersection size. To better understand the difficulty of hiding the intersection size, we briefly go over the design of the protocol of Zhao and Chow [45]. Roughly, it works by generating "secret shares" to the participant. With enough shares, the intersection set can be "unlocked". The difficulty faced by Zhao and Chow in hiding the size of the intersection appears to be the following. On one hand, there should be a way to quickly identify what shares can be used to reconstruct the unlocking key; for otherwise one needs to exhaust an exponential number of possible combinations among the shares. On the other hand, it reveals the number of matches, *i.e.*, the size of the intersection, via the counts of how many individual shares are potentially useful.

Interestingly, we got inspiration from an apparently even more restrictive variant of PSI proposed by Carpent *et al.* [11], which is known as existential PSI (or PSI-X in short). PSI-X only outputs a single bit instead of a set. The output denotes whether the two private sets have any overlapping. As minimal as it may seem, we "upgrade" our own design of PSI-X protocol ($\mathit{\Pi}_X$, see Appendix B for more details) to encode more information. Specifically, we build a protocol $\mathit{\Pi}_{\mathsf{ePSI\text{-}CA}}$ which we call *encrypted private set-intersection cardinality* (ePSI-CA). In this design, cardinality is no longer an unintended leakage but intentionally encrypted. This provides a basis useful for realizing the threshold functionality.

With ePSI-CA, we obtain a $t^{\geq}$-PSI protocol by a simple modification of our $t^{\leq}$-PSI protocol $\mathit{\Pi}_{t^{\leq}\text{-}\mathsf{PSI}}$. Underlying both designs is a technique for computation over encrypted data realized by *oblivious polynomial evaluation* (OPE) [24].

Appendix C summarizes the relations between the PSI variants.

1.2 Merits of Our Constructions

Our proposed protocols are of both practical and theoretical interest. From the efficiency perspective, the complexities of our constructions are linear in the set size n. This beats the classical garbled circuit approach that uses sort-compare-shuffle network with $O(n \log n)$ complexity [30], as well as recent advancement that achieves almost linear (namely $\omega(n)$) complexity [43].

From the design perspective, we demonstrate how to use old techniques in the PSI literature in a novel way to realize PSI functionalities for which no efficient solutions are known. Specifically, many PSI protocols use OPE to encode the private set [13, 24, 27, 28, 37, 45]. Another idea of realizing PSI, which uses Bloom filter related techniques, was first brought by Dong *et al.* [20] and has a shorter history [16, 17, 45]. To the best of our knowledge, for the first time, we combine these two techniques in a non-trivial way. It also suggests a new avenue for addressing other open problems such as private set projection (PSI-P) [11] with linear computational complexity. See Appendices A and B for more details.

Lastly, our protocols remain conceptually simple and modular. Both desiderata greatly simplify the security analysis. Future more efficient $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ instantiation will immediately result in more efficient $\mathit{\Pi}_{t^{\leq}\text{-}\mathsf{PSI}}$ and $\mathit{\Pi}_{t^{\geq}\text{-}\mathsf{PSI}}$ protocols.

1.3 Outsourced Threshold PSI Protocols

Although our protocols are asymptotically efficient, they still rely on public-key techniques, which are difficult to avoid and are not as efficient as symmetric-key primitives (say, hash functions or blockciphers) which often fail to provide algebraic structure for fancy functionalities. PSI protocols are also often highly interactive. For our motivating applications of matchmaking, the interactive nature and the heavy use of public key cryptography hinder the practical usage of PSI.

Our final contribution lies in outsourcing the heavy computations in our protocols to an oblivious cloud. Beyond simply following the trend of leveraging cloud service, we believe that most of the popular mobile applications nowadays, no matter privacy-preserving or not, are often executed with the help of some central servers operated by the service provider. As such, outsourcing PSI not only leads to better efficiency, but also better matches the business model and the usage habits of mobile applications. In the context of using threshold PSI for matchmaking, the user who is a potential match may respond to notification of the smartphone from time to time but would not permanently stay online. It is thus desirable to have the service provider which arbitrates between two users instead of having two users interact with each other directly.

While there is a server which mediates requests between clients, the privacy guarantees of PSI still carry over. In other words, with our outsourced extensions, not only we can enjoy the richer functionalities on top of the privacy provided by our PSI protocols, but also a more deployable protocol which is closer to the real-world model from the perspective of both users and the service provider.

1.4 Related Work

Freedman *et al.* [24] first proposed a PSI protocol based on oblivious polynomial evaluation. Dong *et al.* [20] proposed PSI protocol using oblivious transfer extension. Subsequently, more efficient PSI protocols using similar technique are proposed [38, 39, 42, 44].

A branch of work aims to restrict the output or the leakage of PSI. PSI-CA/PSU-CA reveals only the (approximate) cardinality of the intersection/union but not the set itself [3, 6, 15, 16, 18, 21–24, 29, 37]. Some [3, 15, 18, 22] also use Bloom filter, but none of them can be directly adapted to $\Pi_{\mathsf{ePSI\text{-}CA}}$ (see Sec. 4). Ateniese *et al.* [4] and D'Arco *et al.* [14] proposed (input-)size-hiding PSI. Bradley *et al.* [9] further enable imposing an upper bound on the input set size.

PSI is also considered in the outsourced setting [1, 33–35]. These works do not support advanced threshold set operations.

Concurrent Work. Recently, Hallgren *et al.* [26] also study over-threshold private set-intersection with application in *ride sharing*. They also consider security in the semi-honest model, but the complexity of their protocol is of order $O(n^2)$, which is worse than the garbled circuit approach. A very recent manuscript by Ciampi and Orlandi [12] studies how to perform secure post-processing of the output of PSI protocols in the semi-honest model. Combining their solution with two-party computation techniques allows computation of $f(C \cap S)$ for arbitrary function f. Our protocols are specific protocols which are more efficient ($O(n)$ vs. $O(n^2)$). Also, our design naturally allows the revelation of some auxiliary data (*e.g.*, a session key K encrypting the contact information for matchmaking) when the threshold policy is satisfied. Extending their two-party computation approach [12] in a straightforward way to also support this feature requires a more complicated function f.

Pinkas *et al.* [43] introduce new variants of Cuckoo hashing technique to reduce the number of gates from $O(n \log n)$ to $\omega(n)$ in garbled-circuit based PSI protocol. It is possible to adapt their solution to compute threshold PSI by adding an additional circuit. Therefore, the overall complexity will be at least $\omega(n)$.

2 Preliminary

For a finite set S, $|S|$ denotes its size and $s \stackrel{\$}{\leftarrow} S$ denotes picking an element uniformly at random from S. We denote $[i] = \{1, \ldots, i\}$. We write $\{s_i\}_n$ as a shorthand for the set $S = \{s_1, \ldots, s_n\}$ of n elements. We drop the subscript n if it is clear from context. We use $\mathcal{F}_\rho$ to denote an ideal functionality that implements the protocol ρ, and use $\mathit{\Pi}_\rho$ to denote a concrete construction of the protocol ρ.

A probability ensemble indexed by I is a sequence of random variables indexed by a countable index set I, namely, $X = \{X_i\}_{i \in I}$ where X_i is a random variable. Two distribution ensembles $X = \{X_n\}$ and $Y = \{Y_n\}$ are computationally indistinguishable, denoted by $X \stackrel{c}{\equiv} Y$, if for every PPT algorithm D, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $n \in \mathbb{N}$,

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| \leq \mathsf{negl}(n).$$

2.1 Additive Homomorphic Encryption and Oblivious Polynomial Evaluation

We will use CPA-secure additive homomorphic encryption (AHE) (KeyGen, Enc, Dec) such as Paillier encryption [41]. Given two ciphertexts $\mathsf{Enc}(\mathsf{pk}, m_0)$ and $\mathsf{Enc}(\mathsf{pk}, m_1)$, one can efficiently compute their addition $\mathsf{Enc}(\mathsf{pk}, m_0 + m_1) = \mathsf{Enc}(\mathsf{pk}, m_0) \oplus \mathsf{Enc}(\mathsf{pk}, m_1)$ without using private key sk. For a constant c, given one ciphertext $\mathsf{Enc}(\mathsf{pk}, m)$, one can easily compute $\mathsf{Enc}(\mathsf{pk}, c \cdot m) = c \odot \mathsf{Enc}(\mathsf{pk}, m)$.

To build an OPE protocol, a polynomial $p(x)$ can be hidden by encrypting its coefficients $a_0, \ldots, a_k$. With these encrypted coefficients, denoted by $\mathsf{Enc}(\mathsf{pk}, p(\cdot))$, anyone holding a plaintext s can compute an encryption of $p(s)$ as $\mathsf{Enc}(\mathsf{pk}, a_0) \oplus (\mathsf{Enc}(\mathsf{pk}, a_1) \odot s) \oplus \cdots \oplus (\mathsf{Enc}(\mathsf{pk}, a_k) \odot s^k)$ using addition and constant-multiplication.

2.2 Bloom Filters

A Bloom filter [5] is a compact array of m bits that represents a set S of n elements for efficient set membership testing. It consists of a set of k independent hash functions $H = (h_1, \ldots, h_k)$, h_i uniformly maps elements to index in $[m]$.

All bits in the array are initialized to 0. To insert an element $x \in S$, x is hashed by the k hash functions to get k index numbers. All the bits at these indexes in the array are set to 1, regardless of its original value. To check if an item y is in S, y is hashed by the k hash functions to get k indexes. If any of the bits at these indexes is 0, we conclude that y is certainly not in S (no false negative). Otherwise, y is probably in S. So it incurs only a small fraction of false positives. The upper bound of the false positive probability [8] is: $\epsilon = p^k \left(1 + O\left(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}\right)\right)$ where $p = 1 - (1 - \frac{1}{m})^{kn}$.

If we set the false positive rate to be less than a threshold ϵ_0, it can be shown that the length of the bit array size m should be at least $m \geq n \log_2 e \cdot \log_2 1/\epsilon_0$, where e is the base of the natural logarithm. Equality is achieved when $k = (m/n) \cdot \ln 2 = \log_2 1/\epsilon_0$. We will stick with these optimal values as follows: the false positive probability is $\epsilon = 2^{-\omega(\log \lambda)}$ so that ϵ is negligible in the security parameter λ. As a result, $k = \omega(\log \lambda)$ and $m = k \cdot n \log_2 e$.

2.3 Secure Two-Party Computation

We use the simulation-based security definition for two-party computation (2PC). More details can be referred to [25]. A 2PC protocol π computes a function that maps a pair of inputs to a pair of outputs $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. For every pair of inputs $x, y \in \{0,1\}^*$, the output-pair is a random variable $(f_1(x,y), f_2(x,y))$. The first party obtains $f_1(x,y)$ and the second party obtains $f_2(x,y)$. When $f_i(x,y) = \perp$, it means party i's output is empty.

We first consider *static semi-honest* adversaries, which can control one of the two parties and assumed to follow the protocol specification exactly. However, it may try to learn more information about the input of the other party.

In the semi-honest model, a protocol π is secure if whatever can be computed by a party in the protocol can be obtained from its input and output only. This is formalized by the simulation paradigm. We require the *view* of a party in a protocol execution to be simulatable given only its input and output. The view of the party i during an execution of π on input (x,y) is denoted by $\mathsf{View}_i^\pi(x,y) = (w, r^i, m_1^i, \ldots, m_t^i)$, where $w \in (x,y)$ is the input of i, r^i is i's internal random coin tosses, and m_j^i denotes the j-th message that it received.

Definition 1 (Semi-honest Model). *Protocol π is said to securely compute a deterministic function $f = (f_1, f_2)$ in the presence of static semi-honest adversaries if there exists PPT algorithms $\mathsf{Sim}_1, \mathsf{Sim}_2$ such that*

$$\{\mathsf{Sim}_1(x, f_1(x,y))\}_{x,y} \stackrel{c}{\equiv} \{\mathsf{View}_1^\pi(x,y)\}_{x,y},$$
$$\{\mathsf{Sim}_2(y, f_2(x,y))\}_{x,y} \stackrel{c}{\equiv} \{\mathsf{View}_2^\pi(x,y)\}_{x,y}.$$

The $\mathcal{F}$-hybrid model. We will use some secure two-party protocols as sub-protocols. We will describe our protocols in a "hybrid model" where the two parties interact with each other and use trusted help. When constructing a protocol π which uses a sub-protocol that securely computes some functionality $\mathcal{F}$, we consider that the parties run π and use "ideal calls" to a trusted party for computing $\mathcal{F}$. Upon receiving the inputs from the parties, the trusted party computes $\mathcal{F}$ and sends all parties the corresponding output. After receiving these outputs back from the trusted party, the protocol π continues. By the composition theorem [10], any protocol that securely implements $\mathcal{F}$ can replace the ideal calls to $\mathcal{F}$.

3 PSI with Threshold Policy

3.1 Definitions

We begin with the formal definitions of two private set-intersection with threshold policy functionalities in the literature. We remark that for the second party P_2 (often referred to as "server" in the literature), its output is always $\perp$: it never knows whether the threshold policy is satisfied or not. For the first party P_1 (often referred to as "client" in the literature), if its output is not $\perp$, it knows that the threshold policy is satisfied; however, when its output is $\perp$, it cannot distinguish whether that is due to $C \cap S = \phi$, or $|C \cap S|$ not meeting the policy. Here, the input of the parties includes the size of the input set of the counterparty. This is the standard practice in the multi-party computation literature, when the protocol considered is not "input-size hiding" [40]. We leave "input-size hiding" variants of threshold PSI as a future work.

Definition 2 (Below-Threshold Private Set-Intersection ($t^{\leq}$-PSI) [9]). *Let S and C be subsets of a predetermined domain, the functionality $\mathcal{F}_{t^{\leq}\text{-}\mathsf{PSI}}$ is:*

$$((C, |S|), (S, |C|)) \mapsto \begin{cases} (C \cap S, \perp) & \textit{if } |C \cap S| \leq t \\ (\perp, \perp) & \textit{otherwise} \end{cases}$$

Definition 3 (Over-Threshold Private Set-Intersection ($t^{\geq}$-PSI) [26,45]). *Let S and C be subsets of a predetermined domain, the functionality $\mathcal{F}_{t^{\geq}\text{-}\mathsf{PSI}}$ is:*

$$((C, |S|), (S, |C|)) \mapsto \begin{cases} (C \cap S, \perp) & \textit{if } |C \cap S| \geq t \\ (\perp, \perp) & \textit{otherwise} \end{cases}$$

To realize both $\mathcal{F}_{t^{\leq}\text{-}\mathsf{PSI}}$ and $\mathcal{F}_{t^{\geq}\text{-}\mathsf{PSI}}$, our key insight is to leverage a primitive called *encrypted* PSI *cardinality* (ePSI-CA) functionality in a novel way. This new functionality is formally defined below.

Definition 4 (Encrypted PSI Cardinality (ePSI-CA)). *Let S and C be subsets of a predetermined domain. Let $(\mathsf{pk}_1, \mathsf{sk}_1)$ be a public / secret key pair of a homomorphic encryption scheme. The functionality $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ is:*

$$((C, |S|, \mathsf{pk}_1, \mathsf{sk}_1), (S, |C|, \mathsf{pk}_1)) \mapsto (\perp, \mathsf{Enc}(\mathsf{pk}_1, |C \cap S|))$$

We choose to single out $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ not only for a more compact presentation of $\Pi_{t^{\leq}\text{-}\mathsf{PSI}}$ and $\Pi_{t^{\geq}\text{-}\mathsf{PSI}}$, but also because we believe that $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ itself is an interesting primitive. In Appendix B, we use it to construct the most efficient *existential private set-intersection protocol* (PSI-X) and *private set projection* [11] to date. Motivations and applications of these two protocols can also be found in Appendix B. and [11].

3.2 Intuition

We describe our $\Pi_{t^{\leq}\text{-}\mathsf{PSI}}$ and $\Pi_{t^{\geq}\text{-}\mathsf{PSI}}$ constructions in the ($\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$, $\mathcal{F}_{\mathsf{PSI}}$)-hybrid model, while the concrete instantiation for $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ is deferred to Sec. 4. In what follows, we use $t^{\leq}$-PSI as an example to showcase how it is readily achievable with $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ as a building block. The crux of our novel combination of ePSI-CA and oblivious polynomial evaluation (OPE) is that we use OPE to transform $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ (namely, the output of $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$) to an encryption of a session key K if and only if $|C \cap S| \in [0, t]$. If $|C \cap S| \notin [0, t]$, the evaluation result is random and contains no information about K.

In more details, P_2 re-randomizes the encrypted cardinality obtained from $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ by a random number r as $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$ such that P_1 knows nothing about $|C \cap S|$ after decrypting $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$, but can obliviously evaluate a polynomial at $|C \cap S| + r$. To be more specific, P_2 first prepares a polynomial $p(\cdot)$ whose roots are $r, r + 1, \ldots, r + t$, and chooses a random symmetric key K. P_2 then sends encrypted coefficients of polynomial $p'(\cdot) = r' \cdot p(\cdot) + K$ under its own public key pk_2. In this way, when P_1 obliviously evaluates $p'(\cdot)$ at $|C \cap S| + r$ via additive homomorphism (See Section 2.1), the result will be an encryption of K if and only if $|C \cap S| \leq t$; otherwise it will be encrypting a random number that reveals no information about K (because $r' \cdot p(\cdot)$ serves as a one-time pad encrypting K, while P_2 only used r' once).

To retrieve the value of $p'(\cdot)$ at $|C \cap S| + r$ (which is either the correct session key K or a random value) in plaintext from the previous OPE (which produces $\mathsf{Enc}(\mathsf{pk}_2, p'(|C \cap S| + r))$, P_1 randomizes this ciphertext (denoted by $\mathsf{Enc}(\mathsf{pk}_2, K')$ in Step (4) in Fig. 1) simply by further blinding it with a random number r'' and asks P_2 for decryption. What P_1 eventually obtains is a value K' which equals to K if and only if $|C \cap S| \leq t$. This K' serves as a token showing if the intersection $|C \cap S|$ is below the threshold. The final step is to have P_1 and P_2 engage in a normal Π_{PSI}, in which P_1 and P_2 uses $C^{K'} = \{c_i || K'\}$ and $S^K = \{s_i || K\}$ as input respectively. Hence P_1 can recover the intersection if it possesses the same key $K' = K$.

3.3 $t^{\leq}$-PSI Protocol: $\Pi_{t^{\leq}\text{-}\mathsf{PSI}}$

We describe our $t^{\leq}$-PSI protocol $\Pi_{t^{\leq}\text{-}\mathsf{PSI}}$ in the ($\mathcal{F}_{\mathsf{ePSI\text{-}CA}}, \mathcal{F}_{\mathsf{PSI}}$)-hybrid model in Fig. 1, following to the intuition above. Some points to note are in order.

In Step (2), P_2 blinds the encrypted cardinality $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ obtained from $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ by a uniformly random number r as $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$, which can be viewed as the result of double encryption: firstly encrypt $|C \cap S|$ using a one-time pad r, and then further encrypt under pk_1.

In Step (3), the polynomial $p'(\cdot)$ and $p(\cdot)$ are both degree $(t+1)$, so the number of coefficients to be encrypted and transmitted is $(t+2) \in O(|C| + |S|)$. Also, note that these coefficients are independent of the set S. Looking ahead, such independence allows outsourcing computation to a cloud server.

In Step (4), P_1 needs to blind the evaluation of polynomial by a one-time pad r'' similar to what P_2 does in Step (2). This is because if P_2 sees K' in plaintext, it can check if $K = K'$ and learn a single bit of information $b = (|C \cap S| > t)$, violating the requirement of $\mathcal{F}_{t\leq\text{-}\mathsf{PSI}}$. Recall that in Definition 2, the output of P_2 is always $\perp$.

In Step (6), the session key K' serves as a token that allows P_1 to obtain the intersection when the threshold policy is satisfied via $\mathcal{F}_{\mathsf{PSI}}$ on transformed sets. Yet, when the output of $\mathcal{F}_{\mathsf{PSI}}$ is $\perp$, P_1 does not know whether it is due to $C \cap S = \phi$, or the threshold policy is not met. Meanwhile, P_2 learns nothing from $\mathcal{F}_{\mathsf{PSI}}$, thus satisfying the requirement of $\mathcal{F}_{t\leq\text{-}\mathsf{PSI}}$.

$\Pi_{t\leq\text{-}\mathsf{PSI}}$

Input: P_1's input is $(C,\ |S|)$. P_2's input is $(S,\ |C|)$.

1. **[invoke $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$]** P_1 sends his AHE public key pk_1 to P_2. Next, they invoke an ideal execution of $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ where the input of P_1 is $(C, |S|, (\mathsf{pk}_1, \mathsf{sk}_1))$ and the input of P_2 is $(S, |C|, \mathsf{pk}_1))$. P_2 then gets $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|))$.
2. [P_2 **masks encrypted** $|C \cap S|$] P_2 randomly chooses r and homomorphically computes $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$.
3. [P_2 **prepares encrypted polynomials**] P_2 prepares a polynomial $p(\cdot)$ whose roots are $r, r+1, \ldots, r+t$. P_2 also chooses a random number r' and a random symmetric key K. Finally, P_2 sends encrypted coefficients of the polynomial $p'(\cdot) = r' \cdot p(\cdot) + K$ under pk_2, as well as $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$.
4. [P_1 **evaluates polynomial**] P_1 decrypts the received ciphertext to get $|C \cap S| + r$, and obliviously evaluates $p'(\cdot)$ at point $|C \cap S| + r$. Denote the result by $\mathsf{Enc}(\mathsf{pk}_2, K')$. P_1 blinds it with randomness r'' into $\mathsf{Enc}(\mathsf{pk}_2, K' + r'')$ and asks P_2 for decryption.
5. [P_2 **decrypts**] P_2 decrypts $\mathsf{Enc}(\mathsf{pk}_2, K' + r'')$ and returns $K' + r''$ to P_1, who recovers K'.
6. **[invoke $\mathcal{F}_{\mathsf{PSI}}$]** P_1 and P_2 invoke an ideal execution of $\mathcal{F}_{\mathsf{PSI}}$ where the input of P_1 is $(C^{K'}, |S|)$ and the input of P_2 is $(S^K, |C|)$, where $C^{K'} = \{c_i || K'\}$ and $S^K = \{s_i || K\}$.
7. **[output]** P_1 outputs whatever it receives in the previous step (stripping away the trailing key K' if the output is non-empty).

Fig. 1: Below-Threshold Private Set-Intersection ($\Pi_{t\leq\text{-}\mathsf{PSI}}$)

3.4 Analysis

By the correctness of $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ functionality, P_2 obtains $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ in Step (1). If the size of intersection $|C \cap S| \leq t$, then in Step (3) the polynomial $p(\cdot)$ will be evaluated to 0, and hence the evaluation of $p'(\cdot)$ will be $K' = K$. On the other hand, if $|C \cap S| > t$, the evaluation of $p'(\cdot)$ will be $K' \neq K$. Then by the correctness of $\mathcal{F}_{\mathsf{PSI}}$, P_1 obtains $C \cap S$ if and only if $|C \cap S| \leq t$.

For efficiency, since public-key operations are much slower than symmetric-key ones, we only count the total number of public-key operations, including encryption, decryption, and homomorphic operations (addition and multiplication by a constant). We assume using $\Pi_{\mathsf{ePSI\text{-}CA}}$ to instantiate $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ in Step (1). The number of public-key operations of $\Pi_{\mathsf{ePSI\text{-}CA}}$ will be presented in Table 2. We also assume an efficient Π_{PSI} construction. Note that the state-of-the-art Π_{PSI} protocols [38, 42, 44] under the semi-honest model require linear computation and communication complexities. Table 1 summarizes the overall result, showing that the $\Pi_{t\leq\text{-}\mathsf{PSI}}$ protocol features linear computational complexity.

Table 1: Computational Efficiency of $\Pi_{t\leq\text{-}\mathsf{PSI}}$
($\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ is instantiated by $\Pi_{\mathsf{ePSI\text{-}CA}}$ with complexity in Table 2)

		Enc	Dec	addition	multiplication	Π_{PSI}
Step 1	(P_1)	$\omega(\log\lambda)O(\|C\|+\|S\|)$	0	$\omega(\log\lambda)O(\|C\|+\|S\|)$	0	0
	(P_2)	$\omega(\log\lambda)O(\|C\|+\|S\|)$	$O(\|C\|)$	$\omega(\log\lambda)O(\|C\|+\|S\|)$	$\omega(\log\lambda)O(\|C\|)$	0
Step 2	(P_2)	1	0	1	0	0
Step 3	(P_2)	$t+3$	0	0	0	0
Step 4	(P_1)	1	1	$t+2$	$t+1$	0
Step 5	(P_2)	0	1	0	0	0
Step 6	(P_1)	0	0	0	0	$O(\|C\|+\|S\|)$
	(P_2)	0	0	0	0	$O(\|C\|+\|S\|)$
Total		$\omega(\log\lambda)O(\|C\|+\|S\|)$	$O(\|C\|)$	$\omega(\log\lambda)O(\|C\|+\|S\|)$	$\omega(\log\lambda)O(\|C\|)$	$O(\|C\|+\|S\|)$

Theorem 1. *Assuming the existence of CPA-secure additive homomorphic encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, *whose plaintext space is super polynomial in the security parameter; then the protocol* $\Pi_{t\leq\text{-}\mathsf{PSI}}$ *in Fig. 1 securely implements the functionality* $\mathcal{F}_{t\leq\text{-}\mathsf{PSI}}$ *in Def. 2 in the presence of semi-honest adversaries under the* $(\mathcal{F}_{\mathsf{ePSI\text{-}CA}}, \mathcal{F}_{\mathsf{PSI}})$*-hybrid model.*

Proof. **Simulating the view of P_1 using $\mathsf{Sim}_1^{t\leq\text{-}\mathsf{PSI}}$.** The view of P_1 contains the messages sent by P_2: $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$, pk_2, $\mathsf{Enc}(\mathsf{pk}_2, p'(\cdot))$ (encryptions of coefficients of the polynomial $p'(\cdot)$), $(K' + r'')$, the fixed empty output $\perp$ from $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$, as well as the output from $\mathcal{F}_{\mathsf{PSI}}$.

Let $\mathcal{A}$ be a probabilistic polynomial time (PPT) adversary corrupting party P_1. We design a PPT simulator $\mathsf{Sim}_1^{t\leq\text{-}\mathsf{PSI}}$ that invokes $\mathcal{A}$ by playing the role of the honest party P_2 and $\mathsf{Sim}_1^{t\leq\text{-}\mathsf{PSI}}$ emulates the two ideal functionalities $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$,

$\mathcal{F}_{\mathsf{PSI}}$. The simulator will generate a view indistinguishable from a hybrid one. $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ has different simulation strategies for different outputs of P_1. We consider two disjoint cases.

(1) the output is $\perp$:

1. Given input $((C, |S|, t), \perp)$, $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ invokes $\mathcal{A}$ on input $(C, |S|, t)$, and receives $\mathcal{A}$'s first message pk_1.
2. $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ plays as the trusted party and emulates the ideal calls to $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$.
3. $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ generates a random public/private key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$, a random symmetric key K, just as what P_2 will do, and continues the protocol emulation by sending encryptions of 0 under pk_2 (representing an all-zero polynomial) instead of encryption of coefficients of the polynomial $p(\cdot)$. It also encrypts a random number R_1 under pk_1 to emulate the intended message $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$.
4. When $\mathcal{A}$ obliviously evaluates the zero-polynomial at point R in Step (4), the result will be an encryption of 0 under pk_2. $\mathcal{A}$ randomizes this encryption of 0 into an encryption of r'' and asks $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ for decryption. $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ returns yet another random value R_2.
5. Finally, $\mathcal{A}$ will compute $K' = R_2 - r''$ ($\neq K$ with overwhelming probability). Then $\mathcal{A}$ uses $(C' = \{c_i || K'\}, |S|)$ as input to the $\mathcal{F}_{\mathsf{PSI}}$ functionality emulated by $\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$, who will return $\perp$ as intended.

We argue that this simulated view is indistinguishable from the real one. First, notice that the simulated messages $\mathsf{Enc}(\mathsf{pk}_1, R_1)$ and R_2, are identically distributed as the real ones $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$ and $K' + r''$. It is because, in the real protocol, r is selected by P_2 uniformly at random and K' is distributed uniformly at random when $|C \cap S| > t$ thanks to the randomness r' in Step (3). Second, notice that the other simulated messages are encryptions under pk_2. Any distinguisher of these two views can be transformed to an adversary breaking CPA-security of the encryption scheme.

(2) the output is a subset $\hat{C} \subseteq C$ whose size $|\hat{C}|$ is less than t:
$\mathsf{Sim}_1^{t^{\leq}\text{-}\mathsf{PSI}}$ works as the previous case, except in Step (4) it decrypts the ciphertext to get r''. In Step (5) it computes $K' = R_2 - r''$, converts the set $\hat{C}$ into $\hat{C}^{K'}$ by appending K' to each element in $\hat{C}$. Finally, it uses $\hat{C}^{K'}$ to emulate the output of $\mathcal{F}_{\mathsf{PSI}}$ for $\mathcal{A}$.

This simulated view is indistinguishable from the real view because everything is the same as the other case with the only exception being the output of $\mathcal{F}_{\mathsf{PSI}}$.

Simulating the view of P_2 using $\mathsf{Sim}_2^{t^{\leq}\text{-}\mathsf{PSI}}$.
This part is easy because P_2's view contains only $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ from $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$, $\perp$ from $\mathcal{F}_{\mathsf{PSI}}$ and $\mathsf{Enc}(\mathsf{pk}_2, K' + r'')$. The third element is an encryption of a truly random value (because r'' is chosen by P_1 uniformly at random), which can be perfectly simulated using $\mathsf{Enc}(\mathsf{pk}_2, R)$ where R is also chosen uniformly at random. The first element can be simulated by $\mathsf{Enc}(\mathsf{pk}_1, 0)$. A straightforward

reduction shows that any distinguisher who can distinguish $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ from $\mathsf{Enc}(\mathsf{pk}_1, 0)$ can be used to break the CPA-security of the encryption scheme.

3.5 $t^{\geq}$-PSI Construction and Generalizations

To construct $\Pi_{t^{\geq}\text{-PSI}}$, we modify $\Pi_{t^{\leq}\text{-PSI}}$ in Fig. 1. The modification is simple and straightforward: in Step (3), P_2 prepares a polynomial whose roots are $r + t, r + (t + 1), \ldots, r + \min(|S|, |C|)$, where the function $\min(x, y)$ returns the smaller value of x and y. The rest of the protocol remains exactly the same. Note that the degree of this polynomial is $\min(|S|, |C|) - t + 1$, which remains to be $O(|S| + |C|)$, and hence the efficiency analysis in Table 1 also holds for $\Pi_{t^{\geq}\text{-PSI}}$.

In general, P_2 can specify the roots of the polynomial at will. For example, the roots could be integers within a certain interval $\{r + a, r + (a + 1), \ldots, r + b\}$. It means P_1 only learns the intersection if $|C \cap S|$ falls within the range $[a, b]$. Further generalizing, we can change the criteria to be $|C \cap S| \in \{m_1, m_2, \ldots, m_q\}$, *i.e.*, an arbitrary set of numbers (a subset of even numbers, a subset of prime numbers, *etc.*) instead of consecutive numbers.

Proving the security of $\Pi_{t^{\geq}\text{-PSI}}$ (and its generalizations) just takes a very straightforward adaption of Theorem 1 and thus we omit the repetitive details.

4 Encrypted PSI-Cardinality

Both our $\Pi_{t^{\leq}\text{-PSI}}$ and $\Pi_{t^{\geq}\text{-PSI}}$ protocol heavily rely on $\mathcal{F}_{\text{ePSI-CA}}$, which is a novel variant of PSI and PSI-CA protocol in the literature. It is tempting to instantiate $\mathcal{F}_{\text{ePSI-CA}}$ from $\mathcal{F}_{\text{PSI-CA}}$ by just adding encryption: P_1 and P_2 execute $\mathcal{F}_{\text{PSI-CA}}$, and P_1 encrypts the output $|C \cap S|$ under its public key pk_1, forwards the ciphertext to P_2. Yet, this approach leaks $|C \cap S|$ to P_1 and hence does not satisfy our Definition 4, namely, P_1 should output nothing but $\perp$.

One may instead wonder if some trivial extensions of $\Pi_{\text{PSI-CA}}$ can do. Yet, all $\Pi_{\text{PSI-CA}}$ protocols that we are aware of proceed by transforming elements in the two parties' respective set via some one-way transformation, and let P_1 count the number of common elements in the transformed domain. If one follows this paradigm, it seems impossible to hide the number $|C \cap S|$ from P_1. After all, in $\mathcal{F}_{\text{PSI-CA}}$ and $\mathcal{F}_{\text{PSI}}$, it is P_1 who has non-trivial output while in $\mathcal{F}_{\text{ePSI-CA}}$, it is P_2 instead. Such inherent inconsistency suggests that new techniques are required to design ePSI-CA.

4.1 ePSI-CA Protocol: $\Pi_{\text{ePSI-CA}}$

Our construction is inspired by the *existential private set-intersection* (denoted by $\mathcal{F}_X$) protocol by Carpent *et al.* [11]. We observe that the core of their protocol is implementing an encrypted private membership test protocol, a special case of $\mathcal{F}_{\text{ePSI-CA}}$ where P_1's input set consists of a single element. Unfortunately, their construction is too inefficient, making it unsuitable for our application. We significantly improve their construction by a novel combination of OPE and

Bloom filter. Here we only highlight our design principle. A detailed discussion of $\mathcal{F}_X$ is deferred to Appendix B.

Recall that a Bloom filter BF_S encoding a set S supports efficient membership test by hashing the test element x into k locations using k hash functions. If the value of BF_S at those locations are *all* "1", we conclude that $x \in S$. Namely, the predicate "$x \in S$ or not" is transformed to determining the number of "1"s.

$$P(x, BF_S) = \begin{cases} 1 & x \in S \\ 0 & x \notin S \end{cases}$$
$$= \begin{cases} 1 & \text{exactly } k \text{ "1"s in } \{BF_S[h_1(x)], \ldots, BF_S[h_k(x)]\} \\ 0 & \text{less than } k \text{ "1"s in } \{BF_S[h_1(x)], \ldots, BF_S[h_k(x)]\} \end{cases}$$

Suppose that P_2's Bloom filter BF_S is encrypted under its public-key pk_2, then P_1 can obliviously compute an encryption of the number of "1"s under pk_2 by adding the ciphertexts of those k locations for element x. Let n_X be such number of "1"s, and e_{n_X} be its encryption under pk_2. Our task becomes how to transform e_{n_X} to an encryption of 0 if $n_X \in [0, k-1]$; or to an encryption of 1 if $n_X = k$. This task is similar to what we have in Sec. 3, in which we transform an encryption of $|C \cap S|$ to an encryption of 0 if $|C \cap S| \in [0, t]$; or to an encryption of a non-zero number otherwise.

Fig. 2 gives the full details of $\Pi_{\mathsf{ePSI\text{-}CA}}$. Two notes are in order. In Step (1), $k = \omega(\log \lambda)$ and $N = \omega(\log \lambda)|S| \log_2 e$, which are optimal values for a false positive rate of ϵ that is negligible in the security parameter λ (see Sec. 2.2). In Step (4), P_1 can precompute or outsource this step since the polynomials $p_i(\cdot)$ is independent of the private set C.

4.2 Analysis

Regarding the correctness, in Step (3), the resulting ciphertext is encrypting an integer $\hat{n_i} \in \{r_i, \ldots, r_i + k\}$ because the sum consists of a random r_i and k encrypted numbers in $\{0, 1\}$. For any element $c_i \notin S$, P_1 can only collect less than k encryptions of "1" in Step (3) (otherwise a false positive of Bloom filter has occurred). Therefore, the polynomials $p_i(\hat{n_i})$ will be evaluated to 0 in the ciphertext domain in Step (5). On the other hand, for $c_j \in S$, then the polynomial evaluation $p_j(\hat{n_j}) = k \times (k-1) \times \cdots \times 1 = k!$. After normalizing by a factor of $(k!)^{-1}$, the protocol output is an encryption of $|C \cap S|$.

Next, we analyze the efficiency by counting the total number of public-key operations from Step (2) to Step (5). We assume that the false positive rate of Bloom filter is set to be $\epsilon = 2^{-\omega(\log \lambda)}$, so $k = \omega(\log \lambda)$. The other parameters of Bloom filter are set to the optimal values accordingly. Moreover, we assume that Horner's rule is applied in the evaluation of $p_i(\cdot)$ in Step (5), which requires k additions and k multiplications for evaluating a degree-k polynomial. Table 2 summarizes the result, which shows that the complexity of our construction is only linear in the set size.

In terms of security, we have the following theorem:

$\Pi_{\mathsf{ePSI\text{-}CA}}$

Input: P_1's input is C, $|S|$, and an AHE key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$.
P_2's input is S, $|C|$, and an AHE key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$.

1. [**setup**] The parties perform a secure coin-tossing sub-protocol to choose random seeds for Bloom filter hash functions $h_1, \ldots, h_k : \{0,1\}^* \to [N]$.
2. [P_2 **encrypts its Bloom filter**] P_2 builds an N-bit Bloom filter of his set S using $h_1, \ldots, h_k$, encrypts the bits of the Bloom filter under pk_2, and sends the resulting encrypted bits $e_1, e_2, \ldots, e_N$ to P_1.
3. [P_1 **masks the query results**] For each element $c_i \in C$, P_1 hashes c_i using those k hash functions to obtain k indices $h_1(c_i), h_2(c_i), \ldots, h_k(c_i)$. P_1 creates a ciphertext $e_{\hat{n_i}}$ to be sent to P_2 by homomorphically summing up all ciphertexts at those indices $(e_{h_1(c_i)}, \ldots, e_{h_k(c_i)})$ and another ciphertext of a randomly chosen number r_i.
4. [P_1 **prepares encrypted polynomials**] For all i, P_1 encrypts under pk_1 the coefficients of a degree-k polynomial $p_i(x) = (x-r_i)(x-r_i-1)\cdots(x-r_i-k+1)$. P_1 sends the encrypted coefficients of $p_i(\cdot)$ and $e_{\hat{n_i}}$ to P_2.
5. [**output**] P_2 decrypts $e_{\hat{n_i}}$ to get $\hat{n_i}$. P_2 obliviously evaluates $(k!)^{-1} \cdot (\sum_i p_i(\hat{n_i}))$ via additive homomorphism. Outputs this encrypted result.

Fig. 2: Encrypted PSI Cardinality ($\Pi_{\mathsf{ePSI\text{-}CA}}$)

Table 2: Computational Complexity of $\Pi_{\mathsf{ePSI\text{-}CA}}$
(false positive rate $\epsilon = 2^{-\omega(\log \lambda)}$, # of hash $k = \omega(\log \lambda)$)

	Enc	Dec	addition	multiplication
Step 2 (P_2)	$(\log_2 e)k\|S\|$	0	0	0
Step 3 (P_1)	$\|C\|$	0	$k\|C\|$	0
Step 4 (P_1)	$(k+1)\|C\|$	0	0	0
Step 5 (P_2)	0	$\|C\|$	$(k+1)\|C\|-1$	$k\|C\|+1$
P_1 **Total**	$\omega(\log \lambda)O(\|C\|+\|S\|)$	0	$\omega(\log \lambda)O(\|C\|+\|S\|)$	0
P_2 **Total**	$\omega(\log \lambda)O(\|C\|+\|S\|)$	$O(\|C\|)$	$\omega(\log \lambda)O(\|C\|+\|S\|)$	$\omega(\log \lambda)O(\|C\|)$

Theorem 2. *Assuming the existence of a CPA-secure additive homomorphic encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, whose plaintext space is super-polynomial in the security parameter; then the protocol* $\Pi_{\mathsf{ePSI\text{-}CA}}$ *in Fig. 2 securely implements the functionality* $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ *under the semi-honest model.*

Proof. We consider two corruption cases.
Simulating the view of P_1 using $\mathsf{Sim}_1^{\mathsf{ePSI\text{-}CA}}$. The view of P_1 only contains its view in the coin-tossing protocol View_1^{coin}, pk_2, and $(e_1, \ldots, e_N)$ (encryptions of binary numbers under pk_2). $\mathsf{Sim}_1^{\mathsf{ePSI\text{-}CA}}$ can generate the first two using the Sim_1^{coin}, KeyGen algorithm, while the third one can be simulated by encryptions of zero due to the CPA-security of the encryption scheme. Assume for contradiction that there exists a distinguisher $\mathcal{D}$ for the simulated view and the real view. One can build a distinguisher $\mathcal{D}'$ breaking the CPA-security of the encryption scheme. In the CPA-security game, $\mathcal{D}'$ is given a public-key pk. $\mathcal{D}'$ submits two vectors of plaintexts $\boldsymbol{m_0}, \boldsymbol{m_1}$ where $\boldsymbol{m_0}$ is an all-zero bit vector as constructed in the simulated view, and $\boldsymbol{m_1}$ is the Bloom filter as in the real execution. $\mathcal{D}'$ receives a vector of ciphertext $\boldsymbol{c}$ corresponding to an encryption of either $\boldsymbol{m_0}$ or $\boldsymbol{m_1}$, and directly forwards $(\mathsf{pk}, \boldsymbol{c})$ to $\mathcal{D}$. Finally, $\mathcal{D}'$ outputs what $\mathcal{D}$ outputs. It is easy to see that the advantages of $\mathcal{D}$ and $\mathcal{D}'$ are the same.

Simulating the view of P_2 using $\mathsf{Sim}_2^{\mathsf{ePSI\text{-}CA}}$. The view of P_2 can also be simulated in an analogous way. In particular, the view of P_2 contains View_2^{coin}, P_1's public-key pk_1, encryptions of random numbers $\hat{n_i} = r_i + n_i$ under pk_2, where n_i is a number in $[0, k]$, encryptions of coefficients of a polynomials $p_i(\cdot)$ whose roots are $r_i, r_i + 1, \ldots, r_i + k - 1$ under pk_1. The first two elements can be simulated using Sim_2^{coin} and KeyGen. Encryptions under pk_2 can be generated by encrypting random numbers, while encryptions under pk_1 can be emulated by encryptions of 0. By a similar argument as above, the simulation will be indistinguishable from the real view.

4.3 Reducing Communication Cost

We can reduce the communication cost of Step (2) (sending N encryptions) and Step (4) (sending $k|C|$ encryptions) via *private information retrieval* (PIR) and oblivious transfer (OT) respectively.

In Step (2), instead of transferring the whole encrypted Bloom filter $e_1, e_2, \ldots, e_N$ from P_2 to P_1, it suffices to obliviously transfer only a subset of those ciphertexts which will be used by P_1 in Step (3). The exact number of such ciphertexts depends on P_1's set size $|C|$ and the number hash functions k, and can be upper-bounded by $k|C|$. In more detail, Steps (2) and (3) are replaced by the following steps: Firstly, P_1 constructs a Bloom filter for its private set C according to the hash functions specified in Step (1). Then P_1 records the indices of non-zero bits of the Bloom filter, and uses these indices as input to $k|C|$ instances of a single-server PIR protocol, in which P_2 plays the role of the server holding a database $(e_1, e_2, \ldots, e_N)$ of size N.

Let $\mathsf{PIR}(N)$ be the communication cost of a single server PIR scheme whose database size is N. The overall communication cost of the above approach can be bounded by $k|C| \times \mathsf{PIR}(N)$. Since the state-of-the-art scheme [19] gives $\mathsf{PIR}(N) \in O(\log\log(N))$, the above approach incurs $O(k|C|\log\log(N))$ communication cost instead of $O(N)$. The improvement can be significant when $|C| \in o(\frac{|S|}{\log\log(|S|)})$, namely P_2's set size is much larger than P_1's. Such an unbalanced set size setting is considered in the literature [36] recently.

The security of this optimization can be easily understood as follows: (1) With PIR optimization, P_1 obtains fewer ciphertexts, so it will not harm the security of P_2. (2) PIR also guarantees that P_2 does not know which ciphertexts P_1 has queried, just like the case without optimization in which P_2 sends all ciphertexts and lets P_1 locally decides which of them are useful.

Conceptually, Steps (4) and (5) are executing $|C|$ instances of the following variant of 1-out-of-$(k+1)$ OT. Namely, P_2 is holding an index $\hat{n_i}$ and P_1 is holding a number r_i and an array of $(k+1)$ ciphertexts, the first k being $\mathsf{Enc}(\mathsf{pk}_1, 0)$ and the last one being $\mathsf{Enc}(\mathsf{pk}_1, 1)$. In the end, P_2 obtains the $(\hat{n_i} - r_i)$-th ciphertext. A concrete instantiation of the above protocol using $\log(k+1)$ instances of 1-out-of-2 OT was given by Jarrous and Pinkas [31] as a part of their *bin*HDOT protocol (Fig. 1 in [31]). After applying this technique, the communication cost of Step (4) is reduced from sending $(k+1)|C|$ ciphertexts to executing $|C| \times \log(k+1)$ 1-out-of-2 OT.

The benefits of reduced bandwidth using these two techniques come with a price. On one hand, the use of PIR significantly increases P_2's computational cost. On the other hand, we will discuss how to outsource heavy computations of $\Pi_{\mathsf{ePSI\text{-}CA}}$ to an untrusted cloud in the next section. Unfortunately, these two optimizations do not seem to be compatible with our outsourcing techniques. Therefore, we choose to present $\Pi_{\mathsf{ePSI\text{-}CA}}$ as in Fig. 2. The complexity analysis of $\Pi_{\mathsf{ePSI\text{-}CA}}$ is computed according to the protocol in Fig. 2, without taking the above techniques into consideration.

5 Outsourcing Threshold PSI

Heavy computations in our protocols can be outsourced to an oblivious cloud. As discussed in Sec. 1.4, there are quite a few outsourced PSI protocols [1, 2, 33–35]. However, they only implement the basic PSI protocol with different degrees of outsourceability. None of them supports flexible control as our $t^{\leq}$-PSI and $t^{\geq}$-PSI do. We extend the PSI model by introducing an additional cloud server, denoted by *CSP*. This party serves as an oblivious helper in our $\mathsf{ePSI\text{-}CA}$, $t^{\leq}$-PSI, and $t^{\geq}$-PSI protocols: it helps P_1 and P_2 to perform some heavy computations but remains oblivious to both P_1 and P_2's inputs, and the protocol outcome. *CSP* may share information with P_1 or P_2 (when P_2 or P_1 is an adversary respectively) but we assume that it follows the protocol specification faithfully. Such a semi-honest server is widely accepted in the literature [1, 33–35].

5.1 Outsourcing $\Pi_{\mathsf{ePSI\text{-}CA}}$

As the major building block of the bigger protocols $\Pi_{t\leq\text{-}\mathsf{PSI}}$ and $\Pi_{t\geq\text{-}\mathsf{PSI}}$, we first discuss the outsourceability of each step of $\Pi_{\mathsf{ePSI\text{-}CA}}$.

Step 1 P_1 and P_2 run a coin-tossing protocol per execution to obtain the random seed for hash functions. The seed should remain hidden from CSP.

Step 2 P_2 bitwise-encrypts its Bloom filter. Note that a Bloom filter is always a binary string. As a result, P_2 can prepare a set of ciphertext encrypting "0"s and "1"s under pk_2 *offline* before any protocol execution.
Recall that P_2 uses an N-bit Bloom filter to represent its set. Hence it suffices to let CSP prepare $N_0 = \frac{1+\delta}{2} \times N$ encryptions of "0"s and the same number $N_1 = N_0$ for "1"s. (δ is a small constant and N is the same as that in Sec. 4.1). P_2 permutes these ciphertexts according to a pseudorandom permutation generated from a secret random seed, and uploads the $(N_0 + N_1)$ ciphertexts to the CSP.
These ciphertexts can be reused for different protocol executions as follows: after obtaining the random seeds for the hash functions in Step (1), P_2 locally generates its *plaintext* Bloom filter. For each bit $BF[i]$, P_2 randomly selects one of the N_0 (or N_1) ciphertexts stored in CSP. P_2 informs CSP its choice by sending N indices in total, so that CSP can prepare an *encrypted* Bloom filter for P_2. Note that only P_2 knows whether these ciphertexts are "0"s or "1"s. Hence CSP remains totally oblivious to the content of the Bloom filter.

Step 3 P_1 hashes its elements according to the hash functions agreed in Step (1), and obtains k indices for each element. P_1 sends these locations to CSP so that CSP can homomorphically calculate the sum of these ciphertexts for P_1. Since CSP does not know the random seeds for the hash functions generated in Step (1), these indices are completely random numbers from CSP's point of view. Moreover, the random number r_i is independent of its corresponding element c_i, so CSP can choose r_i on behalf of P_1.

Step 4 The coefficients of the polynomial $p_i(x) = (x - r_i) \cdots (x - r_i - k + 1)$ are completely determined by r_i, which are now selected by CSP in Step (3). Hence CSP can perform the whole Step (4).

Step 5 The polynomial evaluation step homomorphically computes encryption of $a_k \cdot \hat{n_i}^k + \cdots + a_1 \cdot \hat{n_i}^1 + a_0$ in the ciphertext domain using $\hat{n_i}$ from decryption. Note that we cannot reveal $\hat{n_i} = n_i + r_i$ to CSP because CSP knows r_i. The knowledge of n_i leaks whether the i-th element is in the intersection or not. Still, P_2 can locally compute ciphertexts of $a_j \cdot \hat{n_i}^j$ for all $j \in [0, k]$, and then ask CSP to add them together (which saves some computation). In this way, P_2 can still outsource $(k + 1)|C| - 1$ homomorphic additions to CSP.

Putting these together, Fig. 3 presents the outsourced $\Pi_{\mathsf{ePSI\text{-}CA}}$ protocol. Table 3 shows its online computational complexity, with the saving highlighted in red. In short, P_1 can outsource *all* public key operations to the CSP while P_2 can outsource some. Sec. 6 will show that such improvement is significant.

Table 3: Online Computational Complexity of Outsourced $\Pi_{\mathsf{ePSI\text{-}CA}}$
(false positive rate $\epsilon = 2^{-\omega(\log\lambda)}$, # of hash $k = \omega(\log\lambda)$)

	Enc	Dec	addition	constant-multiplication
Step 2 (P_2)	0	0	0	0
Step 3 (P_1)	0	0	0	0
Step 4 (P_1)	0	0	0	0
Step 5 (P_2)	0	$\|C\|$	0	$k\|C\|+1$
P_1 **Total**	0	0	0	0
P_2 **Total**	0	$O(\|C\|)$	0	$\omega(\log\lambda)O(\|C\|)$
CSP **Total**	$\omega(\log\lambda)O(\|C\|+\|S\|)$	0	$\omega(\log\lambda)O(\|C\|+\|S\|)$	0

Outsourcing $\Pi_{\mathsf{ePSI\text{-}CA}}$

Input: P_1's input is C, $|S|$, and an AHE key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$.
P_2's input is S, $|C|$, and an AHE key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$. CSP has no input.

Offline Phase:

– P_2 encrypts N_0 zeros and N_1 ones under pk_2. P_2 randomly permutes these ciphertexts according to some pseudorandom permutation π before uploading these $(N_0 + N_1)$ ciphertexts $(\tilde{e}_1, \ldots, \tilde{e}_{N_0+N_1})$ to CSP.

Online Phase:

1. [**setup**] P_1 and P_2 perform a secure coin-tossing sub-protocol to choose seeds for random Bloom filter hash functions $h_1, \ldots, h_k : \{0,1\}^* \to [N]$.
2. [P_2 **builds encrypted Bloom filter at** CSP] P_2 builds an N-bit Bloom filter BF_S with k hash functions on its set S. P_2 sends an ordered list of N indices $(\mathsf{idx}_1, \ldots, \mathsf{idx}_N)$ to CSP such that $\mathsf{Dec}(\mathsf{sk}_2, \tilde{e}_{\mathsf{idx}_i}) = BF_S[i]$. These N ciphertexts are denoted by $e_1, \ldots, e_N$.
3. [P_1 **sends the query to** CSP] For each element $c_i \in C$, P_1 hashes c_i using those k hash functions to obtain k indices $h_1(c_i), h_2(c_i), \ldots, h_k(c_i)$. P_1 sends these indices to CSP.
4. [CSP **forms encrypted queries**] For each i, CSP creates a ciphertext $e_{\hat{n}_i}$ by homomorphically summing up all ciphertexts at those indices $(e_{h_1(c_i)}, \ldots, e_{h_k(c_i)})$ and another ciphertext of a random number r_i.
5. [CSP **prepares encrypted polynomials**] For all i, CSP prepares encrypted coefficients of a degree-k polynomial $p_i(x) = (x - r_i)(x - r_i - 1)\cdots(x - r_i - k + 1)$ under pk_1. For all i, CSP sends the set of encrypted coefficients of $p_i(\cdot)$ (*e.g.*, $a_{k,i}, \ldots, a_{0,i}$) and $e_{\hat{n}_i}$ to P_2.
6. [P_2 **partially evaluates** $p_i(\cdot)$] For each i, P_2 decrypts $e_{\hat{n}_i}$ to get $\hat{n}_i$, and computes ciphertexts of $a_{j,i} \cdot \hat{n}_i{}^j$ for all $j \in [0, k]$. P_2 sends these ciphertexts to CSP.
7. [**output**] CSP homomorphically adds these ciphertexts into e_Σ. CSP homomorphically multiplies e_Σ with the constant $(k!)^{-1}$. CSP sends this encrypted result to P_2, who outputs it directly.

Fig. 3: Outsourcing $\Pi_{\mathsf{ePSI\text{-}CA}}$

5.2 Outsourcing $\Pi_{t\leq\text{-PSI}}$

We use $\Pi_{t\leq\text{-PSI}}$ as a showcase. It is very straightforward to apply the same technique to $\Pi_{t\geq\text{-PSI}}$ and its generalizations. Basically, most of the public key operations (except decryption) can be outsourced.

Step 1 Invoking of $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$. This can be (partially) outsourced as in Sec. 5.1.

Step 2 The blinding factor r is independent of P_2's private input. Hence the computation of $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$ can be delegated to the cloud.

Step 3 The coefficients of the polynomial $p'(\cdot)$, like those in Step (4) of Fig. 2, are again independent of P_2's private input. Therefore the encryption of coefficients can be outsourced to *CSP*, who will choose r', K on behalf of P_2.

Step 4 Since only P_1 knows sk_1, *CSP* cannot decrypt $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$. Moreover, the decryption result $|C \cap S| + r$ cannot be revealed to *CSP* because *CSP* knows r in Step (2). As a result, the evaluation of $p'(\cdot)$ cannot be fully outsourced to *CSP*, but still P_1 can locally compute encryptions of $a_j \cdot (|C \cap S| + r)^j$ for $j \in [0, t]$, as well as $\mathsf{Enc}(\mathsf{pk}_2, r'')$, and then ask *CSP* to homomorphically add them together.

Step 5 Decryption cannot be outsourced.

Step 6 Intuitively outsourcing Π_{PSI} requires outsourceable PSI. There are quite a few potential solutions with different levels of outsourceability in the literature [1, 2, 33–35]. We refer readers to these papers for more details.

Putting these pieces together, Fig. 4 presents the outsourced below-threshold private set-intersection protocol. Table 4 gives the online computational complexity of outsourced $\Pi_{t\leq\text{-PSI}}$.

Table 4: Online Computational Complexity of $\Pi_{t\leq\text{-PSI}}$
(using outsourced $\Pi_{\mathsf{ePSI\text{-}CA}}$ (*cf.* Table 3) to instantiate of $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$)

		Enc	Dec	addition	multiplication	Π_{PSI}
Step 1	(P_1)	0	0	0	0	0
	(P_2)	0	$O(\lvert C\rvert)$	0	$\omega(\log\lambda)O(\lvert C\rvert)$	0
Step 2	(P_2)	0	0	0	0	0
Step 3	(P_2)	0	0	0	0	0
Step 4	(P_1)	1	1	0	$t+1$	0
Step 5	(P_2)	0	1	0	0	0
Step 6	(P_1)	0	0	0	0	$O(\lvert C\rvert + \lvert S\rvert)$
	(P_2)	0	0	0	0	$O(\lvert C\rvert + \lvert S\rvert)$
P_1 **Total**		$O(1)$	$O(1)$	0	$O(t)$	$O(\lvert C\rvert + \lvert S\rvert)$
P_2 **Total**		0	$O(\lvert C\rvert)$	0	$\omega(\log\lambda)O(\lvert C\rvert)$	$O(\lvert C\rvert + \lvert S\rvert)$
CSP **Total**		$\omega(\log\lambda)O(\lvert C\rvert + \lvert S\rvert)$	0	$\omega(\log\lambda)O(\lvert C\rvert + \lvert S\rvert)$	0	$O(\lvert C\rvert + \lvert S\rvert)$

Outsourcing $\Pi_{t\leq\text{-PSI}}$

Input: P_1's input is (C, $|S|$). P_2's input is (S, $|C|$).
(CSP has no input.)

Offline Phase:

- Execute the offline phase of outsourced $\Pi_{\mathsf{ePSI\text{-}CA}}$ and outsourced Π_{PSI}.

Online Phase:

1. [**execute outsourced** $\Pi_{\mathsf{ePSI\text{-}CA}}$] P_1 generates an AHE public/private key pair $(\mathsf{pk}_1, \mathsf{sk}_1)$, sends pk_1 to P_2. Next P_1, P_2, and CSP engage in an execution of outsourced $\Pi_{\mathsf{ePSI\text{-}CA}}$ where the input of P_1 is $(C, |S|, (\mathsf{pk}_1, \mathsf{sk}_1))$ and the input of P_2 is $(\mathsf{pk}_1, S, |C|)$.
2. [CSP **masks encrypted** $|C \cap S|$] P_2 sends the encrypted cardinality $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ to CSP, who blinds it by a random number r as $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$.
3. [CSP **prepares encrypted polynomials**] CSP prepares a polynomial $p(\cdot)$ whose roots are $r, r+1, \ldots, r+t$. CSP also chooses a random number r' and a random symmetric key K. Finally, CSP sends encrypted coefficients of the polynomial $p'(\cdot) = r' \cdot p(\cdot) + K$ under pk_2, as well as $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$ to P_1.
4. [P_1 **partially evaluates polynomial**] Let the decryption of $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S| + r)$ be x. P_1 homomorphically computes encryption of $a_j \cdot x^j$ under pk_2 for each polynomial coefficient a_j. P_1 sends these ciphertexts to CSP.
5. [CSP **completes polynomial evaluation**] CSP homomorphically adds all received ciphertexts from P_1. Denote the result by $\mathsf{Enc}(\mathsf{pk}_2, K')$. CSP blinds it with randomness r'' into $\mathsf{Enc}(\mathsf{pk}_2, K' + r'')$ and asks P_2 for decryption. CSP sends r'' to P_1 in plaintext.
6. [P_2 **decrypts**] P_2 decrypts $\mathsf{Enc}(\mathsf{pk}_2, K' + r'')$ and returns $K' + r''$ to P_1, who recovers K'.
7. [**execute outsourced** Π_{PSI}] P_1, P_2, and CSP engage in an execution of outsourced Π_{PSI} where the input of P_1 is $(C^{K'} = \{c_i || K'\}, |S|)$ and the input of P_2 is $(S^K = \{s_i || K\}, |C|)$.
8. [**output**] P_1 outputs whatever it receives in the previous step (stripping away the trailing key K' if the output is non-empty).

Fig. 4: Outsourcing $\Pi_{t\leq\text{-PSI}}$

6 Evaluation

For examining the performance of our proposed protocols $\mathit{\Pi}_{\mathsf{ePSI\text{-}CA}}$ and $\mathit{\Pi}_{t\le\text{-}\mathsf{PSI}}$, we conducted our experiment on a desktop machine running Windows 8.1, with 2 Intel(R) Core(TM) i5-4590 3.30GHz CPUs, and 8GB RAM. We fix the size of the sets to be 100 and the threshold t to be half of the set size, namely 50. as they should be sufficient for private-matching application in reality. Note that a dating site eHarmony recently only uses a couple questions that can be finished within 10 minutes to build up a model called "29 dimensions" to build up a user profile. The Bloom filter uses 30 hash functions instantiated by SHA-256, implemented by the OpenSSL library[1]. This number of hash functions reflects a false negative rate $\epsilon = 2^{-30}$ and Bloom filter size 4500 bits. We note that one can achieve better performance at the cost of a larger false negative rate by reducing the number of hash functions, which will lead to a smaller Bloom filter. We use existing Paillier encryption implementation[2], and set the key length to be 2048. Experiments were measured in seconds via wall clock runtime, and the reported runtimes are the average of 100 trials.

We report the computation time of each step of $\mathit{\Pi}_{\mathsf{ePSI\text{-}CA}}$ and $\mathit{\Pi}_{t\le\text{-}\mathsf{PSI}}$ in Table 5 and 6 respectively. The last column represents the online computation time for P_1 and P_2 in the outsourced setting. We do not include the running time of the last step of $\mathit{\Pi}_{t\le\text{-}\mathsf{PSI}}$, because it relies on existing efficient $\mathit{\Pi}_{\mathsf{PSI}}$ protocol (and its outsourced version), which is not part of the contribution of this paper.

From the tables, we see that the outsourced version achieves a significant reduction in computation time for both P_1 and P_2. There are several ways to further reduce it for P_2. First, note that our protocols are easily parallelizable, which means significant improvement can be achieved via multi-threading. Second, recently Jost *et al.* [32] reported optimizations on Paillier cryptosystem that improves naïve implementation by a factor of over 150. Taking these into consideration, our constructions can finish within 1s.

Table 5: Execution time of $\mathit{\Pi}_{\mathsf{ePSI\text{-}CA}}$ (without the optimization of [32])

		time (s)	online (s)
Step 2 (P_2)	create Bloom filter	0.001	0
	encrypt Bloom filter	60.404	0
Step 3 (P_1)	query Bloom filter	2.685	0.001
Step 4 (P_1)	encrypt polynomial	81.268	0
Step 5 (P_2)	decryptions	1.322	1.322
	evaluate polynomials	76.627	76.562
P_1 **Total**		83.953	0.001
P_2 **Total**		138.353	77.884

[1] `https://www.openssl.org`
[2] `https://github.com/herumi/mie`

Table 6: Execution time of $\Pi_{t^{\leq}\text{-PSI}}$, $t = 50$ (without the optimization of [32])

		time (s)	online (s)
Step 1 ($\Pi_{\mathsf{ePSI\text{-}CA}}$)	P_1	83.953	0.001
	P_2	138.353	77.884
Step 2 (P_2)		0.026	0
Step 3 (P_2)		1.363	0
Step 4 (P_1)		1.325	1.324
Step 5 (P_2)		0.013	0.013
P_1 **Total**		85.278	1.325
P_2 **Total**		139.755	77.897

7 Private Matchmaking

We discuss how to utilize our (outsourceable) protocols in our motivating scenario. The matchmaking application is set up as follows. The service provider acts as CSP. When each user joins the system, apart from generating a public key pair for AHE, they randomly pick a symmetric key K and use it to encrypt their profile such as photos and contact. The system suggests a set of attributes (*e.g.*, highly-educated, smoking). The user can mark a subset of attributes to be desired, and mark another disjoint subset to be undesired. The unmarked ones will be considered as "don't care", and they will not be part of the protocol input. The user also picks two thresholds: t_o which is for the least number of desired attributes, another one is t_b which is for the maximum number of undesired attributes.

A user Alice is considered to be matched with another user Bob if and only if she possesses of more than t_o desired attributes specified by Bob, and possess less than t_b undesired ones. If matched, Alice should obtain the symmetric key K that can decrypt the profile of Bob. We use $t^{\leq}$-PSI and $t^{\geq}$-PSI simultaneously to implement the above functionality as follows: Bob splits the symmetric key K into two parts by a simple $(2, 2)$ secret sharing based on XOR. Specifically, Bob picks a key K_o which is as long as K, and outputs both K_o and $K_b = K \oplus K_o$. Bob puts all the undesired (resp. desired) attributes as the private set input of $t^{\leq}$-PSI (resp. $t^{\geq}$-PSI).

Users who joined the service can either be passively matched by others or actively request for matching. Here we discuss a typical protocol run from the perspective of an active user Alice. The service provider will pick a potential user, called a passive user Bob, and execute the PSI protocols on behalf. In other words, that is where the outsourced feature of our protocols comes into the play. Suppose the number of undesired attributes of this passive user Bob is below the threshold t_b after running $t^{\leq}$-PSI, and the number of desired attributes is over the threshold t_o after running $t^{\geq}$-PSI.

If our protocols are used directly, the active user Alice will get the intersection of either kind of attributes. This may not be the most privacy-preserving way for doing matchmaking since Bob has no way to control about whether revealing any secret information (such as the profile) or not. Luckily, the intersection result can be easily removed from our protocols by removing the last step of performing the (keyed-)PSI. As a result, even Bob passed the matching criteria, Alice only obtains a secret key generated by Bob (from the second last step of the protocol).

Here, we utilize the idea of secret transfer with access structure from Zhao and Chow [45]. This key will serve as a proof of criteria satisfaction. Upon the presentation of the aforementioned secret key, the user can decide to reveal K_b (resp. K_o) or not. Such a decision can be done after the service provider executes the PSI protocols on behalf of this passive user. If the interest is mutual, *i.e.*, the requesting user also satisfies the search criteria of the "passive" user, the passive user can finally reveal the encrypted profile.

Two remarks are in order. First, note that even with the help of the service provider who mediates the requests between two users, the outsourced PSI protocols are not entirely non-interactive. For this, the service provider still needs to relay messages between the users. However, it matches with the workflow used by non-private matchmaking apps nowadays which the user cannot connect to another user until there are mutual interests. Second, some user may expect to assign different weightings to different attributes. A trivial approach is to replicate the attribute multiple times. Devising cleverer solutions which maintain a similar level of efficiency requires further twisting of our PSI protocols (perhaps by borrowing techniques from Zhao and Chow [45]). We left it as future work.

8 Conclusion

We propose efficient protocols for three important extensions of PSI, namely, *encrypted private set-intersection cardinality, over threshold private set-intersection, and below threshold private set-intersection*. The last two provide affirmative answers to two open problems posed recently in the literature. All of our constructions achieve linear computational complexity by utilizing and extending existing building blocks in a novel way. We prove that our constructions are secure against semi-honest adversaries. Our constructions provide useful building blocks to realize privacy-preserving online matchmaking.

Acknowledgement

We thank anonymous reviewers for suggesting the use of the technique in [31] to reduce communication cost.

Sherman S.M. Chow is supported by the General Research Fund (CUHK 14210217) of the Research Grants Council, University Grant Committee of Hong Kong.

References

1. Aydin Abadi, Sotirios Terzis, and Changyu Dong. O-PSI: delegated private set intersection on outsourced datasets. In *SEC 2015*, pages 3–17, 2015.
2. Aydin Abadi, Sotirios Terzis, and Changyu Dong. VD-PSI: verifiable delegated private set intersection on outsourced private datasets. In *FC 2016*, 2016.
3. Vikas G. Ashok and Ravi Mukkamala. A scalable and efficient privacy preserving global itemset support approximation using bloom filters. In *DBSec 2014*, 2014.
4. Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (If) size matters: Size-hiding private set intersection. In *PKC 2011*, 2011.
5. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
6. Carlo Blundo, Emiliano De Cristofaro, and Paolo Gasti. Espresso: Efficient privacy-preserving evaluation of sample set similarity. *Journal of Computer Security*, 22(3):355–381, 2014.
7. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC 2005*, 2005.
8. Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang. On the false-positive rate of Bloom filters. *Inf. Process. Lett.*, 108(4):210–213, 2008.
9. Tatiana Bradley, Sky Faber, and Gene Tsudik. Bounded size-hiding private set intersection. In *SCN 2016*, 2016.
10. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
11. Xavier Carpent, Sky Faber, Tomas Sander, and Gene Tsudik. Private set projections & variants. In *WPES 2017*, 2017.
12. Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. IACR Cryptology ePrint Archive 2018/105, 2018.
13. Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS 2009*, 2009.
14. Paolo D'Arco, Maria Isabel Gonzalez Vasco, Angel L. Pérez del Pozo, and Claudio Soriente. Size-hiding in private set intersection: Existential results and constructions. In *AFRICACRYPT 2012*, 2012.
15. Alex Davidson and Carlos Cid. An efficient toolkit for computing private set operations. In *ACISP Part II 2017*, 2017.
16. Sumit Kumar Debnath and Ratna Dutta. Secure and efficient private set intersection cardinality using bloom filter. In *ISC 2015*, 2015.
17. Sumit Kumar Debnath and Ratna Dutta. How to meet big data when private set intersection realizes constant communication complexity. In *ICICS 2016*, 2016.
18. Sumit Kumar Debnath and Ratna Dutta. Provably secure fair mutual private set intersection cardinality utilizing bloom filter. In *Inscrypt 2016*, 2016.
19. Changyu Dong and Liqun Chen. A fast single server private information retrieval protocol with low communication cost. In *ESORICS 2014*, 2014.
20. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *CCS 2013*, 2013.
21. Changyu Dong and Grigorios Loukides. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Trans. Information Forensics and Security*, 12(11):2792–2806, 2017.
22. Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *ACISP 2015*, 2015.

23. Ellis Fenske, Akshaya Mani, Aaron Johnson, and Micah Sherr. Distributed measurement with private set-union cardinality. In *CCS 2017*, 2017.
24. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT 2004*, 2004.
25. Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
26. Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *CSF 2017*, 2017.
27. Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs. In *TCC Part-II 2015*, 2015.
28. Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC 2010*, 2010.
29. Susan Hohenberger and Stephen A. Weis. Honest-verifier private disjointness testing without random oracles. In *PET 2006*, 2006.
30. Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*, 2012.
31. Ayman Jarrous and Benny Pinkas. Secure hamming distance based computation and its applications. In *ACNS 2009*, 2009.
32. Christine Jost, Ha Lam, Alexander Maximov, and Ben J. M. Smeets. Encryption performance improvements of the paillier cryptosystem. IACR Cryptology ePrint Archive, Report 2015/864, 2015.
33. Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In *FC 2014*, 2014.
34. Florian Kerschbaum. Collusion-resistant outsourcing of private set intersection. In *SAC 2012*, 2012.
35. Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *ASIACCS 2012*, 2012.
36. Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *PoPETs*, 2017(4):177–197, 2017.
37. Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO 2005*, 2005.
38. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS 2016*, 2016.
39. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS 2017*, 2017.
40. Yehuda Lindell, Kobbi Nissim, and Claudio Orlandi. Hiding the input-size in secure two-party computation. In *ASIACRYPT 2013*, 2013.
41. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT 1999*, 1999.
42. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security 2015*, 2015.
43. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT 2018*, pages 125–157, 2018.
44. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX Security 2014*, 2014.
45. Yongjun Zhao and Sherman S. M. Chow. Are you the one to share? Secret transfer with access structure. *PoPETs*, 2017(1):149–169, 2017.

A Private Set Projection

Recently, Carpent *et al.* [11] started the study of private set projection (PSI-P). The server in PSI-P has a database $DB = \{d_1, \ldots, d_n\}$ (which may contain duplications) and a column of attributes $A = \{a_1, \ldots, a_n\}$, while the client has a set of attributes $B = \{b_1, \ldots, b_m\}$. After a PSI-P invocation, the server learns nothing while the client only learns $\{d_i | \exists (i, j) \text{ s.t. } b_j = a_i\}$. In particular, the client should not know which matching b_j corresponding to which d_i, nor how many matching b_j that d_i corresponds to.

PSI-P finds application in matching indicators of compromise (IOC), where attribute column A represents IOC (to be checked against the client set B) while the database DB represents patches of known vulnerabilities / attacks. In such scenarios, the correspondence between IOC and patches can be sensitive. Attackers may slightly adapt their attack strategy to avoid being detected by the same IOC. Using PSI-P as a solution, the server (as a security expert) can protect its valuable information (IOC and DB of patches), yet provides a just-enough list of patches to the client, without letting the server know the private set B of the client (*e.g.*, network traffic).

Carpent *et al.* recognize that none of the existing PSI protocols (or their variants) satisfies the security requirements of PSI-P. While outside their radar, oblivious transfer for a sparse array [45] can approximate PSI-P yet it leaks the number of distinct data elements. They thus propose a series of protocols with different leakages, and construct a full-fledged PSI-P from any existential PSI (PSI-X) [11].

B Existential PSI (PSI-X)

Π_X

Input: P_1's input is a set C and $|S|$. P_2's input is S and $|C|$.

1. **[invoke $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$]** P_1 sends his AHE public key pk_1 to P_2. Next the parties invoke an ideal execution of $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ where the input of P_1 is $(C, |S|, (\mathsf{pk}_1, \mathsf{sk}_1))$ and the input of P_2 is $(\mathsf{pk}_1, S, |C|)$.
2. **[P_2 randomizes the encrypted cardinality]** P_2 homomorphically multiplies $\mathsf{Enc}(\mathsf{pk}_1, |C \cap S|)$ obtained from $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ by a random number r as $\mathsf{Enc}(\mathsf{pk}_1, r|C \cap S|)$ before sending it to P_1.
3. **[output]** P_1 decrypts the ciphertext and returns 1 iff. the result is non-zero.

Fig. 5: Efficient PSI-X Protocol (Π_X)

Unfortunately, Carpent *et al.* [11]'s PSI-X construction is inefficient. For client and server set sizes being m and n, the computational complexity is of order

$O(mn)$. Any improvement for PSI-X immediately leads to a better private set projection protocol.

We propose an improved protocol for Π_X. It can be realized by slightly modifying Step (5) of our $\Pi_{\mathsf{ePSI\text{-}CA}}$. Recall that in Step (5) of $\Pi_{\mathsf{ePSI\text{-}CA}}$ P_2 obtains the set-intersection cardinality encrypted under pk_1. P_2 can rerandomize this ciphertext before sending it to P_1.

Definition 5 (Existential Private Set-Intersection (PSI-X)). *Let S and C be subsets of a predetermined domain, the functionality $\mathcal{F}_X$ is:*

$$((C, |S|), (S, |C|)) \mapsto \begin{cases} (1, \perp) & \textit{if } C \cap S \neq \phi \\ (0, \perp) & \textit{otherwise} \end{cases}$$

We begin with a high-level description of the first (but inefficient) PSI-X construction by Carpent *et al.* [11]. Suppose party P_1 has a set C of m elements and party P_2 has a set S of size n. In the existing PSI-X [11], they first jointly choose a *single* 2-universal hash function $h(\cdot)$ mapping set elements to $[N]$ where $N \in O(mn)$.

1. P_1 transforms set C into a bit string v_C of length N such that the $v_C[i] = 1$ if and only if $\exists x \in C : h(x) = i$.
2. P_2 also performs similar operations on S to derive v_S.
3. P_1 generates a BGN [7] public/private key pair $(\mathsf{pk}, \mathsf{sk})$, publishes pk and $\mathsf{Enc}(\mathsf{pk}, v_C[i])$ for all $i \in [N]$.
4. P_2 also encrypts v_S under pk and evaluates the 2-DNF $\phi = \bigvee_{\forall i}(v_C[i] \wedge v_S[i])$ via the homomorphism of BGN.
5. P_2 sends encryption of $r \cdot \phi$ to P_1 for a random r.
6. If P_1 gets 0 after decryption, P_1 concludes that the intersection is *definitely* empty; otherwise it is *probably* non-empty.

The uncertainty stems from the possible collision due to the hash function. One can reduce the error rate by increasing N, or repeating R independent instances of this protocol. Both increase the overall computational complexity in terms of the number of ciphertext multiplications.

Carpent *et al.* [11] show that the optimal choice is $N = \frac{mn}{\log 2}$ for any error rate, resulting in $O(mn)$ complexity. PSI-X with linear computational complexity was an open problem before our paper.

Fig. 5 gives the details of our protocol Π_X, which is very simple in the $\mathcal{F}_{\mathsf{ePSI\text{-}CA}}$ model. Basically we only add one last step: P_2 blinds the encrypted cardinality using a random r before sending it to P_1.

Corollary 1. *Assuming the existence of a CPA-secure additive homomorphic encryption scheme* (KeyGen, Enc, Dec)*, whose plaintext space is super polynomial in the security parameter; then the protocol Π_X in Fig. 5 securely implements the functionality $\mathcal{F}_X$ in Def. 5 under the semi-honest model.*

C A Zoo of Private Set-Intersection and its Variants

A summary of the known relations between private set-intersection and its variants. An arrow from A to B means B can be constructed (solely) from A. If the arrow is solid it means the asymptotic complexity of B is the same as A; otherwise a dashed arrow means the asymptotic complexity of B is worse than A. Protocols in red are proposed in this work. We do not consider protocols that assume or imply general two party computation such as [12].

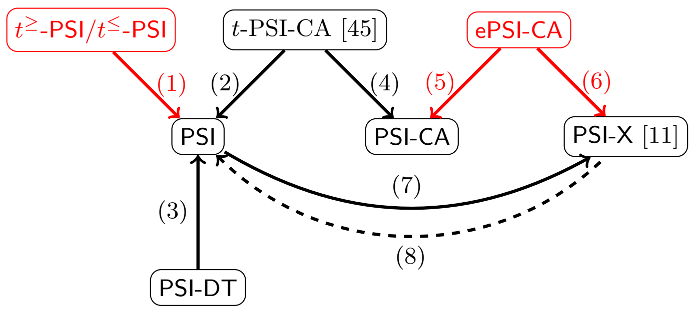

Fig. 6: PSI Zoo

(1) setting $t = 0$ in $t^{\geq}$-PSI or $t = \min(|C|, |S|)$ in $t^{\leq}$-PSI;
(2) setting $t = 0$;
(3) setting the data items to be the same as set elements;
(4) setting $t = \min(|C|, |S|)$;
(5) sending the output of ePSI-CA to the other party;
(6) homomorphically multiplying the output of ePSI-CA with a random number, and sending the result to the other party;
(7) setting the client set to be a single element;
(8) invoking PSI-X for each element in the client set.