Yongjun Zhao and Sherman S. M. Chow

Are you The One to Share? Secret Transfer with Access Structure

Abstract: Sharing information to others is common nowadays, but the question is with whom to share. To address this problem, we propose the notion of secret transfer with access structure (STAS). STAS is a two-party computation protocol that enables the server to transfer a secret to a client who satisfies the prescribed access structure. In this paper, we focus on threshold secret transfer (TST), which is STAS for threshold policy and can be made more expressive by using linear secret sharing. TST enables a number of applications including a simple construction of oblivious transfer (OT) with threshold access control, and (a variant of) threshold private set intersection (t-PSI), which are the first of their kinds in the literature to the best of our knowledge. The underlying primitive of STAS is a variant of OT, which we call OT for a sparse array. We provide two constructions which are inspired by state-of-the-art PSI techniques including oblivious polynomial evaluation (OPE) and garbled Bloom filter (GBF). The OPE-based construction is secure in the malicious model, while the GBF-based one is more efficient. We implemented the latter one and showed its performance in applications such as privacy-preserving matchmaking.

Keywords: oblivious transfer, access structure, private set-intersection, oblivious polynomial evaluation

DOI 10.1515/popets-2017-0010
Received 2016-05-31; revised 2016-09-01; accepted 2016-09-02.

1 Introduction

Many people are disseminating information every day, ranging from short tweet to video. While sharing about oneself is a trend, information can be personal or sensitive. Deciding whom to share becomes an important question, especially when the counter-party may make abusive use of the information. For example, location-based services should only share information between users who are in proximity, or dating apps in which users may only want to share information based on (mutual) interests of the profiled attributes. Since users may not want to reveal their whereabouts or their attributes, the criteria of whom to share may be sensitive as well. Moreover, expecting an *exact* match may be a bit idealistic, *e.g.*, it may be difficult to find someone via dating apps who matches every single desired attributes.

Deciding whom to share may involve comparison of two sets of elements. Also, computing the intersection of two sets is useful in various scenarios, *e.g.*, two Facebook users may check who are their common friends before accepting befriend request; two companies want to find the number of common customers before launching a joint promotion; or apps like tworlds which share photos to a random stranger across the globe simply based on the supplied hashtags. A straightforward approach requires two parties to reveal their sets and then compute the result locally. However, the sets might contain valuable information that should not be disclosed for economic reasons or the sets are too sensitive to reveal. Revealing only the intersection or only its size, while protecting the confidentiality of elements which are not in the intersection, is an important task.

Private set intersection (PSI) can solve the above problem. It involves two parties: a client and a server, each holding a private set C and S respectively. PSI allows the client to learn the intersection $C \cap S$ of their sets, while the server usually learns nothing. If both parties need to learn $C \cap S$, they can switch roles and engage in a second PSI instance. A variant called PSI cardinality (PSI-CA) only reveals the cardinality $|C \cap S|$ (*e.g.*, [19]). State-of-the-art PSI protocols are very efficient (*e.g.*, [17, 41, 42]). PSI has found numerous applications, *e.g.*, bot-net detection [37], proximity testing [38], biometric pattern matching [46], *etc.* Sec. 8 will describe more PSI-related applications.

1.1 PSI with Access Structure

This paper considers a general PSI which only reveals the sets when the intersection satisfies a certain structure. A notable example is threshold PSI (t-PSI), which

Yongjun Zhao: The Chinese University of Hong Kong, E-mail: zy113@ie.cuhk.edu.hk
Sherman S. M. Chow: The Chinese University of Hong Kong, E-mail: sherman@ie.cuhk.edu.hk

only reveals the intersection set $C \cap S$ when the *size* of the intersection $|C \cap S|$ is larger than a pre-agreed threshold t. When $t = 0$, t-PSI is equivalent to PSI; when $t = |C|$, t-PSI is equivalent to PSI-CA. We consider t-PSI as a natural and useful extension to PSI.

Designing an efficient t-PSI protocol (without resorting to *generic* secure multiparty computation) is not an easy task, as observed by Pinkas *et al.* [42]. By using the garbled circuit approach, one can add subsequent computation in a privacy-preserving manner [27]. Specifically, they consider an auditing policy that prevents revealing the intersection if its size exceeds some threshold. We could think of the resulting protocol as a complement of t-PSI defined in this paper. Their experimental results (the Sort-Compare-Shuffle protocol, but not the Bitwise-AND protocol which is only practical for a small universe) show that such additional computation does not incur measurable performance over their plain PSI. Anyhow, the complexity of garbled circuit based t-PSI like protocol is $O(n \log n)$ to the best of our knowledge. On the contrary, one of our constructions achieves linear complexity (at the cost of leaking the size of the intersection). Experiments [17] showed that garbled circuit based PSI protocols can be slower than customized protocols. Moreover, all their constructions are only secure in the semi-honest model, while one of ours is secure in the malicious model.

Constructing t-PSI based on other techniques with better efficiency is non-trivial. To the best of our knowledge, the literature has no such protocol, not to say access structure more general than threshold policy, *e.g.*, weighted threshold, in which different elements carry different weights counting towards the final threshold.

Apart from the technical challenge, we think that t-PSI is a useful primitive that deserves investigation. The threshold version of existing cryptosystems, *e.g.*, signature [45], encryption [15], password-based authenticated key exchange [35, 43], *etc.* have found various applications. We foresee the same will hold for t-PSI.

1.2 Technical Overview of Our Results

We propose the notion of *secret transfer with access structure* (STAS). STAS transfers a secret only to those who satisfy the access structure prescribed by the secret owner, with the aim of revealing as little information about the access structure as possible. STAS is a generalization of threshold PSI. To see, the secret is the intersection of two private sets. The access structure is a threshold policy on the two sets, *i.e.*, the intersection set is only revealed if its size is larger than a threshold.

We focus on threshold secret transfer (TST), as a special instance of STAS. TST allows a server to transfer a secret κ to a client if and only if their respective sets have more than t common elements. We formulate the security definitions for this new primitive, and provide two constructions based on either oblivious polynomial evaluation (OPE) [19], or a variant of garbled Bloom filter (GBF) [17]. OPE and GBF have been used to build state-of-art (vanilla) PSI protocols. For building TST, we introduce *oblivious transfer for a sparse array* (OTSA), in which the selection strings are from a large domain. This helps us to achieve fine-grained access control in private matching. Typical OT only works on an array indexed by polynomially many numbers.

We first show how to build t-PSI-CA, a variant of t-PSI, which allows the client to learn the *size* of the intersection $|C \cap S|$. This suffices for some scenarios and can also serve as a *feature* (see Sec. 8). We propose two solutions that remove such leakage in the full version.

With TST, t-PSI-CA is readily achievable. The server and the client engage in a TST with their respective sets. The client will obtain a secret κ only if there are more than t overlapping elements. Conceptually κ plays the role of a "proof token" to show that the client indeed holds a set containing at least t common elements. We stress that which elements belong to the set remains hidden at this point. To transfer the set, the server appends κ to every element in its set S, and executes another PSI with the client using this new set.

Our construction blueprint is readily extensible to other access structures by replacing the polynomial-based threshold secret sharing with other schemes. For example, we can obtain secret transfer for weighted threshold and weighted t-PSI-CA. We also discuss how to construct STAS by replacing threshold secret sharing with linear secret sharing (LSSS) (see Def. 1), which supports more expressive access control policy. Figure 1 summarizes the roadmap of our constructions.

Our primary focus is to propose efficient protocols which are in the semi-honest (or honest-but-curious) model, in which the adversary follows the protocol exactly as specified, but may try to learn as much as possible about the input of the other party. Semi-honest constructions are often more efficient than their fully secure counterparts. Nevertheless, we also propose one construction that is secure in the malicious model.

We summarize our main contributions as follows:

1. We formally define oblivious transfer for a sparse array (OTSA), and provide two constructions by

integrating existing techniques for private set-intersection in a novel manner. The first construction is conceptually simple and we show how to extend it to be secure in the malicious model. The second one is more efficient with linear computation and communication cost in the size of the array.

2. We formalize a new cryptographic primitive called secret transfer with access structure (STAS). A notable special case of STAS is threshold secret transfer (TST). We provide a construction of STAS from OTSA and secret sharing. We also elaborate the applications of this new primitive.
3. We (for the first time) formalized the notion of threshold private set-intersection (t-PSI) and its weaker variant t-PSI-CA. We provide an efficient generic transform from TST and PSI to t-PSI-CA in the semi-honest model. Both the definition and construction can be easily extended to a more expressive access structure by replacing TST with STAS.
4. We proposed two ways to achieve t-PSI.
5. We implement the second OTSA construction and evaluate its performance. The result shows that it is practical for array size expected by our applications.

The rest of the paper is organized as follows. The next section discusses primitives related to TST including PSI. Sec. 3 introduces notations and important building blocks of our protocol. In Sec. 4, we introduce OTSA, which may be of independent interests. We then define TST and construct TST from OTSA in Sec. 5. Sec. 6 presents our generic construction of t-PSI-CA. We provide evaluation results in Sec. 7. Finally, we conclude with further applications and future work. Table 1 lists the acronyms of the major primitives.

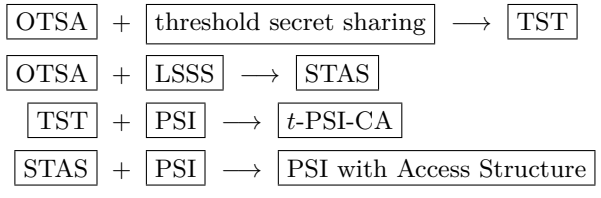

Fig. 1. Roadmap of Our Constructions

OPPRF	**oblivious permuted pseudorandom function**
OTSA	**oblivious transfer for a sparse array**
TST	**threshold secret transfer**
$(ST)^2$	**strong threshold secret transfer**
STAS	**secret transfer with access structure**
t-PSI	**threshold private set-intersection**
t-PSI-CA	**threshold private set-intersection cardinality**
OPE	**oblivious polynomial evaluation**
OPRF	**oblivious pseudorandom function**
OT	**oblivious transfer**
BF	**Bloom filter**
GBF	**garbled Bloom filter**
LSSS	**linear secret sharing scheme**
PSI	**private set-intersection**
PSI-CA	**private set-intersection cardinality**

Table 1. List of major acronyms: the upper table contains notions we introduced; the lower one list those existing in the literature.

2 Related Work

2.1 Private Set Intersections

The first rigorous treatment for PSI was done by Freedman *et al.* [19], who proposed a protocol based on *oblivious polynomial evaluation* (OPE). The key idea is that the client uses additive homomorphic encryption to encrypt the coefficients of a polynomial $p(x)$ whose roots are the elements in the set. The server obliviously evaluates the polynomial $rp(x) + x$ for each element $s_i \in S$. The client can then decrypt the evaluation results from the server and compare those with its own set. Using similar technique, Kissner and Song [31] proposed multi-party PSI protocol. PSI protocols in the malicious model using this technique are also proposed [13, 22, 24].

Another tool for constructing PSI is *oblivious pseudorandom function* (OPRF) [18, 23, 28]. The server sends to the client $f_k(s_i)$ for all $s_i \in S$, where $f(\cdot)$ is a pseudorandom function and k is a random secret key. The client and the server then engage in an OPRF protocol such that the client learns $f_k(c_i)$ for each $c_i \in C$ while the server learns nothing. PSI protocols based on OPRF achieve linear time and space complexities.

Recently, Dong *et al.* [17] introduced new techniques for PSI. The core component is a variant of Bloom Filter, which they called *garbled Bloom filter* (GBF), to be described in the next section. With *oblivious transfer* (OT), they constructed very efficient PSI protocols in both the semi-honest and malicious models. Inspired by

their work, we construct PSI with access structure with linear complexity in the semi-honest model.

Researchers have considered variants of PSI. *PSI-CA* reveals only the size (cardinality) of intersection but not the set itself [2, 19, 26, 31]. Camenisch and Zaverucha [10] and De Cristofaro *et al.* [11, 12] considered authorized PSI which requires a trusted party to authorize the input sets by signing on them. Ateniese *et al.* [3] and D'Arco *et al.* [14] proposed size-hiding PSI which hides the size of the input sets. Outsourcing the computation of PSI to an oblivious cloud is considered by Abadi *et al.* [1] and Kerschbaum [30].

Fuzzy variants of PSI are studied in the literature as well. Freedman *et al.* [19] considered fuzzy PSI where an element is considered to be in another set if this set contains a "similar" element. Kissner and Song [31] considered threshold set union and its variants in a multi-user setting: all n players learn which elements appear more than t times in the union (which is a multiset) of their private input sets. Both notions are different from t-PSI.

2.2 Fuzzy Vault

A fuzzy vault [29] allows a server to "lock" a small secret κ using a set S, such that a client holding another set C can recover κ efficiently if C is similar to S. It looks like TST, but we highlight the differences here.

We first briefly recall the existing construction [29]: the elements in set S are encoded as distinct x-coordinates. The server selects a random polynomial $p(\cdot)$ that encodes κ (say $p(0) = \kappa$), and evaluates $p(\cdot)$ on these coordinates. To hide κ, the server adds a number of random "noisy" points that do not lie on $p(\cdot)$, and publishes the set of both real and noisy points *in clear* as the vault.

If the client holds a set C that substantially overlaps with S, it can identify enough common elements via x-coordinates to reconstruct (the polynomial and) κ. If the intersection is not large enough, the client is unable to identify enough "correct" x-coordinates to perform polynomial interpolation. However, an adversary can still try to reconstruct κ via trial-and-error. The success probability of this attack is non-negligible unless an exponential number of noisy points are added.

In TST, we aim at imposing a *sharp distinction* between "over threshold" and "below threshold". The client only knows $|C \cap S|$ if it is less than the threshold, but not exactly which elements in C lie in the intersection. This is not easily achievable by fuzzy vault.

2.3 Attribute-Based Encryption

Attribute-based encryption (ABE) allows sharing of encrypted content to people according to some prescribed access control policy. Anyone in possession of attributes satisfying the access policy can use their secret keys for decryption. Early ABE schemes reveal the access policy of ciphertexts to everyone, which might be sensitive. Anonymous ABE [33, 39] is then proposed.

One might attempt to implement STAS by encrypting the secret value using anonymous ABE. Yet, ABE requires a trusted authority to set up the whole system and issue secret keys to participants according to their attributes. Such requirement somewhat trivializes the two-party computation since both parties need to trust the authority not to reveal the attributes to others.

Furthermore, the anonymity and/or functionality offered by practical anonymous ABE schemes nowadays are not ideal. For example, the size of public key and ciphertext in the first anonymous ABE scheme [39] are both linear in the size of the attribute domain $\mathcal{U}$, and it only supports limited form of policies. The scheme of Lai *et al.* [33] supports linear secret sharing scheme (LSSS) (see Def. 1) based policy, yet they sacrifice anonymity to some extent. Each attribute is a category-value (*e.g.* Title: Professor, Department: CS) pair in their construction. The LSSS matrix is defined over the categories which is public. Only the attribute values are anonymized. That is to say, a policy "(Title: Professor) AND (Department: CS)" is anonymized as "(Title: $*$) AND (Department: $*$)". In contrast, such an access policy is anonymized as "A AND B" in our STAS construction where A and B are predicates. To conclude, existing anonymous ABE is not a good fit.

3 Preliminary

3.1 Notations

For a bit-string x, $|x|$ denotes its length. The i^{th} bit of x is $x[i]$ and $x[i, j]$ denotes $x[i] \dots x[j]$ for $1 \leq i \leq j \leq |x|$. For a finite set S, $|S|$ denotes its size and $s \xleftarrow{\$} S$ denotes picking an element uniformly at random from S. For $i \in \mathbb{N}$, $[1, i] = \{1, \dots, i\}$. We write $\{s_i\}_n$ as a shorthand for the set $S = \{s_1, \dots, s_n\}$ of n elements. We drop the subscript n if it is clear from context. The security parameter is $\lambda \in \mathbb{N}$ and its unary representation is 1^λ.

Algorithms are probabilistic polynomial time (PPT) unless otherwise stated. By $y \leftarrow A(x_1, \dots; R)$, we run

algorithm A on input $x_1, \ldots$ using randomness R, and assign the output to y. For brevity we may omit R.

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 $X \overset{c}{\equiv} Y$, if for every PPT algorithm D, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $n \in \mathbb{N}$,

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

3.2 Secret Sharing

Threshold secret sharing is a fundamental cryptographic primitive and it could be considered as the most basic tool for threshold cryptography. It allows a dealer to split a secret κ into n shares, such that κ can be recovered efficiently with any subset of t or more shares. Any subset of size less than t reveals no information about the secret value. Shamir secret-sharing scheme [44], which is based on polynomial interpolation, is such a (t, n)-secret sharing scheme. We denote $\mathsf{SecretSharing}_{(t,n)}(\kappa)$ and $\mathsf{Reconstruct}_{(t,n)}(\{\kappa_i\}_{n'})$ as its sharing algorithm and reconstruction algorithm.

When $t = n$, an efficient (n, n)-secret sharing scheme can be obtained from $\oplus$ (XOR) operations.It works by picking random $|\kappa|$-bit strings $\kappa_1, \kappa_2, \ldots, \kappa_{n-1}$ as the first $n-1$ shares. The last share is given by $\kappa_n = \left(\bigoplus_{i=1}^{n-1} \kappa_i\right) \oplus \kappa$. We will use this scheme extensively.

3.3 Linear Secret-Sharing Schemes

We can extend threshold secret sharing scheme to support more general access structure. Let $\{\kappa_1, \ldots, \kappa_n\}$ be a set of secret shares. A collection $\mathbb{A} \subseteq 2^{\{\kappa_1, \ldots, \kappa_n\}}$ is monotone when $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure [4] (resp. monotonic access structure) is a collection (resp. monotone collection) $\mathbb{A}$ of non-empty subsets of $\{\kappa_1, \ldots, \kappa_n\}$, *i.e.*, $\mathbb{A} \subseteq 2^{\{\kappa_1, \ldots, \kappa_n\}} \setminus \{\varnothing\}$. We use $\mathsf{SecretSharing}_{\mathbb{A}}(\kappa)$ and $\mathsf{Reconstruct}_{\mathbb{A}}(\{\kappa_i\}_{n'})$ to denote the sharing algorithm and reconstruction algorithm in a secret sharing scheme implementing access structure $\mathbb{A}$. Any monotone access structure can be realized by a linear secret sharing scheme defined below [4].

Definition 1 (Linear Secret-sharing Schemes (LSSS)). *A secret sharing scheme* Π *is called linear over* $\mathbb{Z}_p$ *if*

1. *The shares of each party form a vector over* $\mathbb{Z}_p$.
2. *There exists a share-generating matrix* M *for* Π. *The matrix* M *has* ℓ *rows and* n *columns. For a column vector* $v = (\kappa, r_2, \ldots, r_n)$, *where* $\kappa \in \mathbb{Z}_p$ *is the secret to be shared and* $r_2, \ldots, r_n \in \mathbb{Z}_p$ *are randomly chosen, then* Mv *is a vector of* ℓ *shares of the secret* κ *according to* Π. *The share* $(Mv)_x$, *the* x*-th row of* Mv, *belongs to party* $\rho(x)$, *i.e.,* ρ *maps* $\{1, \ldots, \ell\}$ *to each party.*

Any LSSS defined as above enjoys the linear reconstruction property as follows. Suppose that Π is an LSSS for access structure $\mathcal{A}$. Let $S \in \mathcal{A}$ be an authorized set, and $I \subseteq \{1, \ldots, \ell\}$ be defined as $I = \{i : \rho(i) \in S\}$. There exist constants $\{w_i \in \mathbb{Z}_p\}_{i \in I}$ satisfying $\sum_{i \in I} w_i M_i = (1, 0, \ldots, 0)$, so that if $\{\lambda_i\}$ are valid shares of any secret κ according to Π, $\sum_{i \in I} w_i \lambda_i = \kappa$. Furthermore, these constants $\{w_i\}$ can be found in time polynomial in the size of the share-generating matrix M. For any unauthorized set, no such constants exist. The LSSS is denoted by (M, ρ), and its size is the number of rows of M.

Access structures can also be represented by monotonic boolean formulas. The techniques of transforming any monotonic boolean formula to LSSS are well known in the literature [4]. One can also convert the boolean formula into an access tree. An access tree of ℓ nodes results in an LSSS matrix of ℓ rows. Readers can refer to [34, appendix] for the conversion algorithm.

3.4 Oblivious Transfer

Oblivious transfer (OT) is another basic cryptographic building block. It allows the receiver to get only part of the sender input, while the sender remains oblivious about what the receiver obtains. Formally, in an OT_ℓ^m protocol, the sender inputs m pairs ℓ-bit strings $(x_{i,0}, x_{i,1})$ $(1 \leq i \leq m)$ and the receiver inputs an m-bit selection string $b = (b_1, \ldots, b_m)$. At the end of the protocol, the sender learns nothing about b, while the receiver only gets x_{i,b_i} for $1 \leq i \leq m$. OT protocols in the random oracle model can be very efficient.

The above formulation only allows the receiver to choose *1-out-of-2* according to each *bit* in the selection string. In k-out-of-n OT, the receiver only gets k of n strings from the sender by specifying a set of distinct indexes (in the range $[1, n]$ instead of $\{0, 1\}$) of size k.

We use the notion (which slightly abuses the one defined above) $\mathrm{OT}_{I_s}^{|I_s \cap I_r|}$ for *oblivious transfer for a sparse array* (OTSA). Note that the superscript is a number while the subscript is a set, thus differentiates OTSA

from OT_ℓ^m. Roughly speaking, $\mathrm{OT}_{I_s}^{|I_s \cap I_r|}$ can be considered as an $|I_s \cap I_r|$-out-of-$|I_s|$ OT. The meaning behind such superscript and subscript will be more clear after we explain the meaning of I_s and I_r in Sec. 4.1.

3.5 Additive Homomorphic Encryption

We will also use a CPA-secure additive homomorphic encryption scheme. One well known example is Paillier cryptosystem [40]. It supports addition, and multiplication by a constant, without private key sk: (1) given two ciphertexts $\mathsf{Enc}_{pk}(m_0)$ and $\mathsf{Enc}_{pk}(m_1)$, there is an efficient operation computing $\mathsf{Enc}_{pk}(m_0+m_1)$; (2) given one ciphertext $\mathsf{Enc}_{pk}(m)$ and a constant c, there is an efficient operation computing $\mathsf{Enc}_{pk}(c \cdot m)$. A corollary of these two properties is: given encryptions of the coefficients $a_0, \ldots, a_k$ of a polynomial $p(x)$, and a plaintext s, it is possible to compute an encryption of $p(s)$.

3.6 Oblivious (Permuted) PRF

An oblivious pseudorandom function [18] is a two-party protocol between a server S and a client C for securely computing a pseudorandom function $f_k(\cdot)$ under key k known by S only, while the input x is known by only C. The client learns $f_k(x)$ while the server learns nothing after their interaction. In this work, we consider the construction given by Jarecki and Liu [28]. It is secure under parallel composition [28] and our second construction relies on this special property. This protocol uses Camenisch-Shoup version [9] of Paillier encryption [40] $(\mathsf{Enc}_{pk}(\cdot), \mathsf{Dec}_{sk}(\cdot))$, which is additive homomorphic, to compute Dodis-Yampolskiy PRF [16] $f_k(x) = g^{\frac{1}{k+x}}$. We sketch the main procedures as follows:

1. The server sends $\mathsf{Enc}_{pk_s}(k)$ to the client.
2. The client chooses a random number r_c, computes $\mathsf{Enc}_{pk_s}(r_c(k+x))$ by the homomorphic property, and then replies with $(c_1, c_2) = (\mathsf{Enc}_{pk_s}(r_c(k+x)), \mathsf{Enc}_{pk_c}(r_c))$.
3. The server decrypts c_1 and computes its inverse $\frac{1}{r_c(k+x)}$. It also chooses a random number r_s and uses the homomorphic property to compute $s_1 = c_2^{\frac{1}{r_c(k+x)}} \cdot \mathsf{Enc}_{pk_c}(-r_s)$. Then it replies with $(s_1, s_2) = (\mathsf{Enc}_{pk_c}(\frac{1}{k+x} - r_s), g^{r_s})$.
4. The client decrypts s_1 to get $\frac{1}{k+x} - r_s$, and computes the final output as $g^{\frac{1}{k+x}} = g^{\frac{1}{k+x} - r_s} \cdot s_2$.

The parallel version of this OPRF can be easily obtained by replacing r_c, r_s, (c_1, c_2), and (s_1, s_2) with $\{r_c^{(i)}\}_{|C|}, \{r_s^{(i)}\}_{|C|}, \{(c_1^{(i)}, c_2^{(i)})\}_{|C|}$, and $\{(s_1^{(i)}, s_2^{(i)})\}_{|C|}$ respectively in Steps 2 to 4. Also note that if the server applies a random permutation Π on $\{(s_1^{(i)}, s_2^{(i)})\}_{|S|}$, the client will still get the same set $\left\{g^{1/(k+x_i)}\right\}_{|C|}$, but it does not know which $g^{\frac{1}{k+x_i}}$ corresponds to which x_i. We denote such parallel OPRF with extra permutation as *oblivious permuted pseudorandom function* (OPPRF).

3.7 (Garbled) Bloom Filters

An (m, n, k, H)-Bloom filter [5] is a compact array of m bits that represents a set S of at most n elements for efficient set membership testing. It consists of a set of k independent hash functions $H = (h_1, \ldots, h_k)$ where h_i uniformly maps elements to index numbers in $[1, m]$.

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. The bits at all 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. Otherwise, y is probably in S. So, it never yields a false negative, but there is a small fraction of false positives. The upper bound of the false positive probability [6] is: $\epsilon = p^k \left(1 + O(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}})\right)$ 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 the number of hash functions $k = (m/n) \cdot \ln 2 = \log_2 1/\epsilon_0$. The rest of the paper will stick with these optimal values when we use (garbled) Bloom filter. Specifically, we set the false positive probability to $\epsilon = 2^{-\lambda}$ where λ is the security parameter. As a result, $m = \lambda n \log_2 \epsilon$ and $k = \lambda$ in all cases. We represent a Bloom filter with optimal parameters as an (n, H, λ)-BF.

An (m, n, k, H, λ)-garbled Bloom filter [17] is a variant of Bloom filter that enables efficient PSI. Roughly speaking, a garbled Bloom filter uses an array of λ-bit strings instead of an array of bits in a normal Bloom filter. Initially all m strings in the garbled Bloom filter are set to NULL. To insert an element $x \in \{0,1\}^\lambda$, x is first split into k shares by XOR-based (k, k)-secret sharing. The i^{th} share is placed at location $h_i(x)$. If the location $h_i(x)$ is already occupied due to previous insertion, we *reuse* the string at that location, and adjust the value of subsequent shares accordingly, subject to the constraint that $\bigoplus_{i=1}^k GBF[h_i(x)] = x$. Such adjust-

ment is always possible unless all locations $\{h_i(x)\}_{1\le i\le k}$ are all occupied, which corresponds to a false positive. The probability of this happening can be negligible if the parameters are set properly. After inserting all elements in S to GBF, the undefined slots are filled with random strings. To check if an item y is in S, y is hashed by the k hash functions, and the strings in those locations are retrieved. If y can be reconstructed from these shares, we conclude that y is surely in S. For brevity, GBF with optimal parameters will be denoted by (n, H, λ)-GBF.

3.8 Zero-knowledge Proof of Knowledge

Zero-knowledge proof system, introduced by Goldwasser *et al.* [21], enables a prover to convince a verifier that some statement is true, without conveying any other information apart from such mere fact since the verifier can simulate the communication transcript. We use $\mathsf{ZKPoK}\{a|\phi(a)\}$ to denote a zero-knowledge proof of knowledge of value a that satisfies a publicly computable relation ϕ, where a can be extracted when given the trapdoor for the common reference string.

4 OT for a Sparse Array

We propose oblivious transfer for a sparse array (OTSA) as a building block for secret transfer with access structure (STAS). We first provide a formal definition for OTSA and then sketch the main design idea before presenting two concrete constructions.

Although in an abstract level, the two constructions both make use of the idea of *PSI with data transfer*, we believe the first OPE-based construction is conceptually simpler and easier to understand than the second GBF-based one for readers who do not have prior knowledge in recent advances of PSI, in particular, (garbled) Bloom filter. Moreover, at the end of this section, we will elaborate how to extend this construction to accommodate malicious adversaries. As a consequence, the OPE-based one is worth mentioning, even though its computation complexity is quite high (quadratic in the number of elements). The second construction achieves linear complexity using recent technique, *i.e.*, combining GBF and OT. In Sec. 7, we implement the second (more practical) construction and evaluate its performance.

4.1 Definitions

OTSA is a new variant of the original OT concept. The sender holds an array of n_s elements $\{e_i\}_{1\le i\le n_s}$ (from a certain domain), which are associated with n_s distinct indices $I_s = \{s_1, \ldots, s_{n_s}\}$, where each s_j for $1 \le j \le n_s$ is an element from the domain $\mathcal{D}$, not necessarily $[1, n_s]$. The receiver specifies a set of indices $I_r = \{r_1, \ldots, r_{n_r}\}$, not necessarily a subset of I_s, and asks for retrieving the associated elements. We assume the domain $\mathcal{D}$ is large, *i.e.*, $n_s \ll |\mathcal{D}|$ and $n_r \ll |\mathcal{D}|$, so not every index $r_j \in I_r$ specified by the receiver indeed has an element from the sender associated with it. Our OTSA satisfies the following properties:

- **Correctness:** The receiver retrieves $E' = \{e'_j\}_{1\le j\le n_r}$, where $e'_j = e_j$ if $s_j \in I_s \cap I_r$, or e'_j is an element randomly picked from a pre-defined domain if $s_j \notin I_s \cap I_r$.
- **Receiver privacy:** The sender learns nothing about I_r.
- **Sender array privacy:** The receiver learns nothing about e_j's whose index $s_j \notin I_s \cap I_r$.
- **Sender indices privacy:** The receiver learns nothing about I_s except $|I_s \cap I_r|$.

Distinction from Normal OT. Normal OT typically represents the sender indices by $[1, n_s]$. A possible way to use OT to realize the functionality we want to achieve is to require the sender to publish a 1-to-1 mapping between $[1, n_s]$ and $\{s_1, \ldots, s_{n_s}\}$, but this breaks the sender indices privacy. Another way is to use normal OT in which the sender holds $|\mathcal{D}|$ elements. This will incur $O(|\mathcal{D}|)$ communication complexity. Ideally, we shoot for $O(\max(n_r, n_s))$. We remark that our primitive implies the normal OT since one can simply set $I_s = [1, n_s]$.

More on Correctness Requirement. For OTSA to be useful, the receiver should have the ability to differentiate a correct data element from a random string. In case the application (*e.g.*, in TST) may not make it apparent, it can be achieved by asking the sender to append a special symbol to e_i for recognition, or publish $\{H(e_i)\}$ where $H(\cdot)$ is a one-way hash function.

On Sender Indices Privacy. When a correct data element is distinguishable from a random string, the receiver can learn the cardinality of the intersection, *i.e.*, $|I_s \cap I_r|$, according to the functionality requirement. We stress that OTSA itself does not leak such information. Even if the cardinality is revealed from auxiliary information like $\{H(e_i)\}$, the receiver will not know what

exactly the intersection set $I_s \cap I_r$ is. This level of privacy suffices for our applications in Sec. 8.

4.2 Security

Using the language of secure two-party computation, we define OTSA as the following functionality:

Definition 2 (Oblivious Transfer for a Sparse Array)**.** *OTSA is a two-party computation protocol that implements the following functionality*

$$f(x, y) = (\perp, E')$$

where the input of server $x = (E, I_s)$ consists of two sets of the same size n_s, one being the (multi-)set of data elements $E = \{e_1, \ldots, e_{n_s}\}$, another being the index set $I_s = \{s_1, \ldots, s_{n_s}\}$ of the server. The receiver input $y = I_r$ is a set of indices of size n_r. The output of the receiver E' is a subset of E, such that $e_j \in E'$ if and only if its index $s_j \in I_s \cap I_r$.

We assume that the size of index sets, namely n_s and n_r, are publicly known by both parties.

We say that a protocol π is an OTSA if it securely implements the above function f. As discussed in Sec. 3.4, we denote such a protocol by $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$.

Protocol: OPE-based $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

Input: The receiver input is an index set $I_r = \{r_1, \ldots, r_{n_r}\}$. The sender input is an index set $I_s = \{s_1, \ldots, s_{n_s}\}$ and a data set $E = \{e_1, \ldots, e_{n_s}\}$.

1. The receiver chooses a key pair (pk, sk) for a CPA-secure additive homomorphic encryption scheme $(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$, and publishes pk.
2. The receiver computes the coefficients of the polynomial $p(x) = \sum_{i=0}^{n_r} a_i x^i$ of degree n_r with elements in the *selection strings* set I_r as roots.
3. The receiver encrypts each of the (n_r+1) coefficients by the additive homomorphic encryption scheme and gives the sender the resulting set of ciphertexts, $\{\mathsf{Enc}_{pk}(a_i)\}$.
4. For each index $s_i \in I_s$, the sender:
   (a) Uses the homomorphic property to evaluate the encrypted polynomial at s_i, namely computes $\mathsf{Enc}_{pk}(p(s_i))$.
   (b) Chooses a random value r and computes $\mathsf{Enc}_{pk}(rp(s_i) + e_i)$.
5. The sender sends a random permutation of these n_s ciphertexts to the receiver.
6. The receiver decrypts all n_s ciphertexts received, picks up the set of valid elements.

Fig. 2. Protocol: OPE-based $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

4.3 Construction Idea

Our two constructions borrow ideas from the PSI literature. One is based on oblivious polynomial evaluation (OPE); the other one is based on a variant of garbled Bloom filter. It has been observed that the PSI protocol from OPE can actually allow the transfer of auxiliary information. Hence, we exploit this storage capacity to store the data elements for our OTSA. Corresponding, we make the same observation for the GBF-based PSI. We note that while both PSI protocols share the same property, their construction ideas are quite different.

4.4 OPE-based $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

Now we describe our first construction, which is based on oblivious polynomial evaluation, in Figure 2.

The correctness of this construction is straightforward: if $s_i \in I_r \cap I_s$, then $rp(s_i) + e_i = e_i$, meaning that the receiver successfully received one element; otherwise $rp(s_i) + e_i$ will be a random string containing no useful information about e_i. More formally, we assert the security of the above protocol in the following theorem:

Theorem 1. *If $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure homomorphic public key encryption scheme, whose plaintext space is super polynomial in the security parameter; the protocol in Fig. 2 securely implements the function f in Definition 2 (Sec. 4) in the semi-honest model.*

Proof. We consider two corruption cases.

Simulating the view of the sender using Sim_s. This case is easy, since the view of the sender only contains pk and $\{\mathsf{Enc}_{pk}(a_i)\}$. Sim_s can generate the first one using the KeyGen algorithm, while the second one can be simulated by encryption of zeros due to the CPA-security of the encryption scheme. Assume for contradiction, 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 $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. In the CPA-security game,

$\mathcal{D}'$ is given a public key pk. $\mathcal{D}'$ submits two vectors of plaintexts m_0, m_1 where m_0 is coefficients of zero-polynomials as constructed in the simulated view, and m_1 is the coefficients of polynomials as in the real execution. $\mathcal{D}'$ receives a vector of ciphertext c corresponding to encryptions of m_0 or m_1, and directly forwards (pk, 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 the receiver using Sim_r. It is easy to simulate this view because the receiver only sees either encryptions of random elements in the message space, or encryptions of a subset of server elements E'. In more details, Sim_r is given as input $I_r = \{r_1, \ldots, r_{n_r}\}$ and E'. It invokes a copy of the (semi-honest) receiver internally using I_r as input, while playing the role of the honest sender. Sim_r first gets the public key pk from the receiver, and a vector of ciphertexts encrypting the coefficients of a polynomial constructed from I_r. Sim_r outputs the simulated view as a permutation of the following set of ciphertexts: (1) for $|E'|$ of them, each encrypts $e_j \in E'$, (2) for the rest $|I_s| - |E'|$ of them, just encrypts random messages. We claim that the simulated view and the real view are statistically close. The only potential difference is that in the simulated view, the receiver sees $|I_s| - |E'|$ random messages while in the real view, it sees encryptions of $rp(s_i) + e_i$ for $s_i \in I_s \setminus I_r, e_i \in E \setminus E'$, where r is a fresh random element for each i. Due to the fresh randomness r and the requirement that the message space is super-polynomial in the security parameter, these two distributions are thus statistically close. □

4.5 Extending to the Malicious Model

One may expect that techniques in constructing malicious model PSI protocols [19, 22, 24] can be adapted for our setting to construct a malicious model $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$. Although this is true for simulating the case against a malicious client, we encounter some immediate difficulties when simulating in the face of a malicious server.

Recall that the major technique [19, 24] is to derandomizing computation for elements in the intersection. To make sure the server follows the protocol execution, the server needs to compute encryptions of $rp(s_i) + \hat{r}$, where $\hat{r}$ is chosen at random, and the randomness in the encryption algorithm is derived deterministically from $H(\hat{r})$. (Specifically, $H(\cdot)$ can be a PRF [24] or may be modelled as a random oracle in some security proof [19]). After decryption, the client receives a set $\{\hat{r}\}$, and then for each $\hat{r}$ repeats the encryption process for every element c_i in the client set. Because computation can be repeated at the client side deterministically, the server does not have the freedom to cheat. If we adapt it for our setting, it will violate sender indices privacy because $I_r \cap I_s$ is revealed in trial encryption. Similar obstacle appears when using techniques from a recent work of Hazay [22].

To solve this problem, we turn to the common reference string (CRS) model and use zero-knowledge proofs systems. We require a CRS containing a public key $\widetilde{pk}$ of a CPA-secure public key encryption scheme, whose distribution is the same as if it is generated through $\widetilde{\mathsf{KeyGen}}$ algorithm, but no party knows the corresponding secret key $\widetilde{sk}$. At the beginning, the sender encrypts its index-data pair (s_i, e_i) under this public key $\widetilde{pk}$ and sends a permutation of such pairs to the receiver. The receiver and the sender continue the original protocol execution except that when the sender returns $c_i = \mathsf{Enc}_{pk}(rp(s_i)+e_i)$, it also prepares a zero-knowledge proof proving the knowledge of r, s_i, e_i such that c_i can be computed from $\mathsf{Enc}_{pk}(r), \mathsf{Enc}_{pk}(s_i), \mathsf{Enc}_{pk}(e_i)$, and the encrypted polynomial $\{\mathsf{Enc}_{pk}(a_i)\}$. The same proof also shows that s_i, e_i are the plaintext of some ciphertext under public key $\widetilde{pk}$ sent previously. Figure 3 descries our protocol in detail.

Theorem 2. *If* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a CPA-secure homomorphic public key encryption scheme, whose plaintext space is super polynomial in the security parameter, and* $(\widetilde{\mathsf{KeyGen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ *is a CPA-secure encryption scheme, the protocol in Fig. 3 securely implements the function f in Def. 2 (Sec. 4) in the malicious model.*

Appendix D gives the proof of Theorem 2.

4.6 GBF*-based $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

Our second construction is based on a variant of garbled Bloom filter (GBF), which we call it *secret embedding GBF**. An (n, H, λ)-secret embedding $\mathrm{GBF}^*_{X,I}$ stores a secret set X based on another index set I of the same size n. In the original GBF, each element $x_j \in X$ is first split into $k = \lambda$ shares, these shares are then placed at locations $h_1(x_j), \ldots, h_k(x_j)$. While in our $\mathrm{GBF}^*_{X,I}$, specifically, the boxed line in $\mathsf{BuildGBF}^*$ (Algorithm 1) in Appendix B, each $x_j \in X$ is split and placed at locations defined by $h_1(i_j), \ldots, h_k(i_j)$, where $i_j \in I$. Namely, the set I "indexes" the locations to place X.

Protocol: OPE-based $\mathsf{OT}_{I_r}^{|I_s \cap I_r|}$ in the malicious model

Input: The receiver input is an index set $I_r = \{r_1, \ldots, r_{n_r}\}$. The sender input is an index set $I_s = \{s_1, \ldots, s_{n_s}\}$ and a data set $E = \{e_1, \ldots, e_{n_s}\}$.

Common reference string: a public key $\widetilde{pk}$ of a CPA-secure public-key encryption scheme $(\widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$.

1. The sender commits to its input I_s and E by computing $\{(\tilde{s_i}, \tilde{e_i})\} = \{(\mathsf{Enc}_{\widetilde{pk}}(s_i), \mathsf{Enc}_{\widetilde{pk}}(e_i))\}$, and sends a permutation of these pairs to the receiver.
2. The receiver chooses a key pair (pk, sk) for a CPA-secure additive homomorphic encryption scheme $(\mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$, and publishes pk.
3. The receiver computes the coefficients of the polynomial $p(x) = \sum_{i=0}^{n_r} a_i x^i$ of degree n_r with roots being elements in the *selection strings* set I_r.
4. The receiver encrypts each of the $(n_r + 1)$ coefficients by Enc_{pk} and gives the sender the resulting set of ciphertexts, $\{\hat{a_i} = \mathsf{Enc}_{pk}(a_i)\}$, and the following zero-knowledge proof of knowledge:

$$\mathsf{ZKPoK}_1\{(a_i, r_{a_i}) | \ \forall i, \ \hat{a_i} = \mathsf{Enc}_{pk}(a_i; r_{a_i}) \ \wedge \ p(\cdot) \not\equiv 0\}.$$

5. If ZKPoK_1 does not pass, the sender aborts. Otherwise, for each index $s_i \in I_s$, the sender:
   (a) Uses the homomorphic property to evaluate the encrypted polynomial at s_i, *i.e.*, $\mathsf{Enc}_{pk}(p(s_i))$.
   (b) Chooses a random value r and computes $c_i = \mathsf{Enc}_{pk}(rp(s_i) + e_i)$.
   (c) Prepares the following zero-knowledge proof of knowledge:

$$\mathsf{ZKPoK}_2\{(r, s_i, e_i) | (\hat{s_i}, \hat{e_i}) = (\mathsf{Enc}_{pk}(s_i), \mathsf{Enc}_{pk}(e_i)) \ \wedge \ (\tilde{s_i}, \tilde{e_i}) = (\mathsf{Enc}_{\widetilde{pk}}(s_i), \mathsf{Enc}_{\widetilde{pk}}(e_i))$$

$$\wedge \ \hat{r} = \mathsf{Enc}_{pk}(r) \ \wedge \ c_i = \hat{r} \cdot (\sum_{j=0}^{n_r} \hat{a_j} \hat{s_j}^j) + \hat{e_i}\}.$$

6. The sender sends a random permutation of these n_s ciphertexts together with the ZKPoK_2 to the receiver.
7. The receiver verifies the proofs. If all the proofs pass then it decrypts all n_s ciphertexts received, and picks up the set of valid elements. Otherwise it aborts.

Fig. 3. Protocol: OPE-based $\mathsf{OT}_{I_r}^{|I_s \cap I_r|}$ in the malicious model

When we query $\mathrm{GBF}^*_{X,I}$ using some element $i' \in I$ in $\mathsf{QueryGBF}^*$, $\mathrm{GBF}^*_{X,I}$ returns the corresponding element x'. If $i' \notin I$, it returns a uniformly random string. The $\mathsf{GBF}^*\mathsf{Intersection}$ algorithm takes as input a $GBF^*_{X,I}$ and a $BF_{I'}$, then outputs $GBF^*_{\tilde{X}, I \cap I'}$.

The $\mathsf{BuildGBF}^*$, $\mathsf{QueryGBF}^*$, and $\mathsf{GBF}^*\mathsf{Intersection}$ algorithms of GBF* as well as the related theorems are listed in Appendix B. Figure 4 details the procedure of our second $\mathsf{OT}_{I_r}^{|I_s \cap I_r|}$ construction. In the second step, the set I_r of the receiver is transformed into a pseudorandom set I'_r. We implement this transformation using an OPPRF protocol (Sec. 3.6) instead of a normal OPRF because we need to hide the one-to-one correspondence between r_j and r'_j from the receiver. Any secure OPPRF suffices, so our design is modular.

Theorem 3. *The protocol in Fig. 4 securely implements the function f in Sec. 4 in the semi-honest model.*

Proof. The proof mostly follows the proof of [17, Theorem 7], which we highlight the differences here. The simulator needs to randomly generate a set of indices $\tilde{I}$ of size $|I_s \cap I_r|$, and uses it as the additional input to construct $GBF^{*\pi}_{\tilde{E}, \tilde{I}}$. This is computationally indistinguishable from $GBF^*_{\tilde{E}, I'_r \cap I'_s}$ by the security of PRF. Thus, by Theorem 9 in Appendix B, it is also indistinguishable from $GBF^{*\pi}_{\tilde{E}, I'_s \cap I'_r}$. □

5 (Threshold) Secret Transfer

We now discuss how to make use of OTSA to construct *secret transfer with access structure* (STAS). For illustration, we first define STAS with a simple threshold access structure. We call it threshold secret transfer (TST). Next we demonstrate how to easily extend TST to support general access structure, namely STAS.

Protocol: GBF* based $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

Input: The receiver input is an index set $I_r = \{r_1, \ldots, r_{n_r}\}$, the sender input is an index set $I_s = \{s_1, \ldots, s_{n_s}\}$ and a data set $E = \{e_1, \ldots, e_{n_s}\}$.

1. The sender chooses a random key k_{PRF} for Dodis-Yampolskiy PRF $f_{k_{\mathsf{PRF}}}(\cdot)$.
2. The receiver and the sender engage in an OPPRF protocol using I_r and k_{PRF} as input respectively, such that the sender learns nothing while the receiver obtains $I'_r = \{r'_j\}$ where $r'_j = f_{k_{\mathsf{PRF}}}(r_j)$, $\forall j$. The sender also converts its own private set I_s into I'_s in a similar manner.
3. The receiver generates an (n, H, λ)-$BF_{I'_r}$ that encodes its private set I'_r, the sender invokes Algorithm 1 to generate an (n, H, λ)-GBF^*_{E,I'_s}.
4. The receiver uses $BF_{I'_r}$ as the *selection strings* and acts as the receiver in an OT_λ^m protocol.
5. The sender sends m pairs of λ-bit strings $(x_{i,0}, x_{i,1})(1 \le i \le m)$ in OT_λ^m, where $x_{i,0}$ is a uniformly random string and $x_{i,1}$ is $GBF^*_{E,I'_s}[i]$. For $1 \le i \le m$, if $BF_{I'_r}[i]$ is 0, then the receiver receives a random string; if $BF_{I'_r}[i]$ is 1 it receives $GBF^*_{E,I'_s}[i]$. The result obtained by the receiver is denoted by $GBF^{*\pi}_{\tilde{E},I'_s \cap I'_r}$.
6. The receiver queries $GBF^{*\pi}_{E,I'_s \cap I'_r}$ using its own set I'_r, and picks up the valid elements set $\{\tilde{E}\}$.

Fig. 4. Protocol: GBF*-based $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

5.1 Definition

Definition 3 (Threshold secret transfer (TST))**.** *TST is a two-party computation protocol that implements f:*

$$f(x, y) = \begin{cases} (\perp, \kappa \text{ and } |C \cap S|) & \text{if } |C \cap S| \ge t \\ (\perp, |C \cap S|) & \text{otherwise} \end{cases}$$

where (κ, S) and C are the inputs of server and client.

This definition always leaks the intersection size to the client. It is due to the technical difficulty of simulating to the client without this knowledge, which we will discuss in more details later. For the sake of completeness, we give the stronger definition called *strong threshold secret transfer* ($(\mathrm{ST})^2$) in Appendix C, and provide two candidate $(\mathrm{ST})^2$ constructions in the full version.

5.2 TST Construction from $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

The basic idea behind the construction is to split κ into $\{\kappa_i\}$ using a $(t, |S|)$-secret sharing scheme. The server and the client then engage in an $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ protocol with the server acting as the sender using secret input set K and indexing set S, and the client acting as the receiver using C as the input set. By the security of $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$, the client receives only a subset of secret shares of κ corresponding to elements in $|C \cap S|$. The security of TST naturally follows from that of $(t, |S|)$-secret sharing. Figure 5 describes our construction in detail.

Protocol: TST from $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

Input: (Client) $C = \{c_i\}_{n_c}$ / (Server) $S = \{s_i\}_{n_s}$, κ.

1. The server computes the set of shares $K = \{\kappa_i\} \xleftarrow{\$} \mathsf{SecretSharing}_{(t,n_s)}(\kappa)$.
2. The server acts as the sender with input $x = (K, S)$ and the client acts as the receiver with input C in an $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ protocol.
3. Let $\tilde{K}$ be the set of the secret shares collected by the client. If $|\tilde{K}| \ge t$, outputs $\kappa = \mathsf{Reconstruct}_{(t,n_s)}(\tilde{K})$ and $|\tilde{K}|$; $|\tilde{K}|$ otherwise.

Fig. 5. TST Construction from $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$

Theorem 4. *If the underlying secret sharing scheme and the $OT_{I_r}^{|I_s \cap I_r|}$ protocols are secure, then the protocol defined in Fig. 5 securely evaluates the TST functionality in the semi-honest model.*

Proof. If $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ is secure, the simulators for both sides exist. We can use them as subroutines to construct the simulator for the whole protocol.

Server: We construct a simulator Sim_S, when given the private input and output of the server, simulates its view View_S of a real protocol execution. View_S contains the input set S, the secret value κ, the random coins, and the view of $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$. The simulator Sim_S selects random coins r^s uniformly random, and also computes $K' = \{\kappa'_i\} \xleftarrow{\$} \mathsf{SecretSharing}_{(t,n_s)}(\kappa)$. Then Sim_S invokes the simulator for $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ to obtain $\mathsf{Sim}_{\mathrm{SX}}^{OT}((K', S), \perp)$. Finally, Sim_S outputs $(S, K', r^s, \mathsf{Sim}_{\mathrm{SX}}^{OT}((K', S), \perp))$ as the simulated view. Because K' is generated in the same way as in the real protocol, it follows that $\mathsf{Sim}_{\mathrm{SX}}^{OT}((K', S), \perp) \stackrel{c}{\equiv} \mathsf{Sim}_{\mathrm{SX}}^{OT}((K, S), \perp)$. Thus by the se-

curity of $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$, the simulated view should be indistinguishable from View_S.

Client: We construct Sim_C which simulates View_C of a real protocol execution when given the private input and output of the client. View_C contains the input set C, the random coins, and the view of $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$.

- If $|C \cap S| \geq t$, the simulator Sim_C is given $C, \kappa, |K| = |C \cap S|$ as input. Sim_C picks coins r^c uniformly random, and also computes $K' \xleftarrow{\$} \mathsf{SecretSharing}_{(t,n_s)}(\kappa)$. Sim_C then selects a random subset $\tilde{K'} \subseteq K'$ of size $|K|$ uniformly at random. Then Sim_C invokes the simulator for $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ to obtain $\mathsf{Sim}_{\mathrm{rx}}^{OT}(C, (\tilde{K'}, |K|))$. Sim_C outputs $(C, r^c, \mathsf{Sim}_{\mathrm{rx}}^{OT}(C, (\tilde{K'}, |K|)))$ as the simulated view. Since K' and $\tilde{K'}$ are generated in the same way as K and $\tilde{K}$ in the real protocol, $\mathsf{Sim}_{\mathrm{rx}}^{OT}(C, (\tilde{K'}, |K|))$ is identically distributed as $\mathsf{Sim}_{\mathrm{rx}}^{OT}(C, (\tilde{K}, |K|))$. By the security of $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$, the simulated view is indistinguishable from View_C.
- If $|C \cap S| < t$, Sim_C is given only C and $|K| = |C \cap S|$. This time Sim_C selects a random κ', and computes $K'' \xleftarrow{\$} \mathsf{SecretSharing}_{(t,n_s)}(\kappa')$. Sim_C then selects a subset $\tilde{K''} \subseteq K''$ of size $|K| < t$ uniformly at random. Sim_C outputs $(C, r^c, \mathsf{Sim}_{\mathrm{rx}}^{OT}(C, (\tilde{K''}, |K|)))$ as the simulated view. To see that the above simulation works well, notice that by the security of the (t, n_s)-secret sharing scheme, both $\tilde{K''}$ generated by Sim_C and $\tilde{K}$ received in the real protocol execution leak no information about the original value κ' and κ respectively. Thus $\tilde{K''}$ and $\tilde{K}$ are computationally indistinguishable. Therefore $\mathsf{Sim}_{\mathrm{rx}}^{OT}(C, (\tilde{K''}, |K|))$ simulates the view of $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ perfectly.

In both cases, the view of the client can be simulated. With the simulation for the server, we conclude that the protocol in Fig. 5 is secure in the semi-honest model. □

5.3 Extending to the Malicious Model

Note that the above proof only asserts that the construction in Figure 5 is secure in the semi-honest model. Here we sketch how to obtain TST in the malicious model using the construction presented in Figure 3 (in the common reference string model) as an example. Firstly, we would need to have the sender commits to the secret value κ at the beginning of the protocol together with $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ input (K, S). Moreover, the sender has to prepare a zero-knowledge proof showing that K is indeed the result of secret sharing κ. By doing so, we are assured that a malicious server cannot place invalid secret shares $\hat{K}$ without being caught.

5.4 Transferring Multiple Secrets

Our exposition only considers transferring the secret shares of a single secret. Yet, it is possible to store multiple shares of the same secret in the same slot for supporting weighted TST. One step further, it is also possible to store multiple shares of *different* secrets in the same slots. However, since the capacity of each slot is limited. One may need to resort to a hybrid approach, such that the TST stores symmetric keys which can in turn decrypt the ciphertexts of multiple shares.

5.5 Extending to General Access Structure

The threshold access structure of TST directly comes from the underlying threshold secret sharing scheme $(\mathsf{SecretSharing}_{(t,n_s)}, \mathsf{Reconstruct}_{(t,n)})$. By replacing it with another scheme with different access structure $(\mathsf{SecretSharing}_{\mathbb{A}}, \mathsf{Reconstruct}_{\mathbb{A}})$, the TST construction readily transforms into STAS with access structure. The proof strategy remains mostly unchanged. In particular, linear secret sharing (LSSS) [4] (see Def. 1) fits with our design well. We do not need the usual mapping ρ from attributes to row number of the matrix, since our OTSA supports a sparse array with indices from a large domain.

Recall that in LSSS, the share generating matrix M is public. The secret value κ is embedded in a column vector $v = (\kappa, r_2, \ldots, r_n)$, and party $\rho(x)$ gets the share $(Mv)_x$. In STAS, the server also needs to publish M. Moreover, it explicitly appends an index x to each share $(Mv)_x$, so that the client knows how to calculate constants $\{w_i \in \mathbb{Z}_p\}_{i \in |I_s \cap I_r|}$ according to M in the reconstruction phase. Exposing M in clear reveals some information about the access structure, *e.g.*, the shape of the corresponding access tree. However, we would like to stress that by the security property (sender indices privacy) of the underlying OTSA, the client does not know the correspondence between the elements in its secret set C and the leaf nodes of the access tree.

5.6 Discussion on Intersection Size

From a theoretical point of view, knowing both C and $|C \cap S|$ allows one to infer some information about S,

especially when $|C| \approx |C \cap S|$. For instance, if $|C| = 100$ and the client learns that $|C \cap S| = 70$, it can conclude that many elements in C are also in S. Moreover, if the client can interact with the server multiple times, it can change C by one element each time and monitor how $|C \cap S|$ changes accordingly, which eventually lead to S.

From a practical standpoint, such leakage is acceptable because the aforementioned probing attack can be mitigated by limiting the number of interactions. Moreover, in the next section we will see an immediate application of TST (resp. STAS), *i.e.*, a generic t-PSI-CA (resp. PSI with access structure) construction from TST (resp. STAS). The fact that TST always leaks the cardinality of intersection hinders us from obtaining t-PSI. Nevertheless, we believe t-PSI-CA is still a useful primitive and it is the first of its kind in the literature. Finally, we remark that, when the cardinality of the intersection is not exceeding the threshold, t-PSI-CA is the same as PSI-CA; otherwise t-PSI-CA works as a normal PSI.

6 Threshold-PSI-CA Protocol

6.1 Definitions

Definition 4 (Threshold Private Set-Intersection)**.** *t-PSI is a two-party computation protocol that implements the following functionality*

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

where the server input is S and the client input is C.

Definition 5 (t-PSI-Cardinality)**.** *t-PSI-CA is a two-party computation protocol that implements the following functionality*

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

where the server input is S and the client input is C.

We assume t is known by the client beforehand.

6.2 Generic t-PSI-CA from TST

With all the building blocks at hand, we are now ready to build t-PSI(-CA) in a simple manner. The client and the server first engage in a TST protocol, such that the client learns κ if $|C \cap S| \geq t$; then they engage in a normal PSI protocol, in which the server and the client uses $S^\kappa = \{s_i||\kappa\}_{n_s}$ and $C^\kappa = \{c_i||\kappa\}_{n_c}$ as input respectively. In case $|C \cap S| < t$, the client chooses a random κ' uniformly at random and use $C^\kappa = \{c_i||\kappa'\}_{n_c}$ instead. The correctness of the above idea is straightforward. What we need to prove is its security. To this end, we first formally describe the above construction in Fig. 6.

Protocol: t-PSI-CA

Input: (Client) $C = \{c_i\}_{n_c}$ / (Server) $S = \{s_i\}_{n_s}$.

1. The server randomly picks κ from $\{0,1\}^\lambda$.
2. The client and the server engage in a secure TST protocol π^{TST} with the server input being (κ, S) and the client input being C.
3. The server computes $S^\kappa = \{s_i||\kappa\}_{n_s}$.
4. If the client obtains κ from π^{TST}, it computes $C^\kappa = \{c_i||\kappa\}_{n_c}$. Otherwise it randomly samples κ', and computes C^κ as $\{c_i||\kappa'\}_{n_c}$.
5. The client and the server engage in a secure PSI protocol π^{PSI} with the server input being S^κ and the client input being C^κ.

Fig. 6. Protocol: t-PSI-CA

Theorem 5. *Let π^{TST} be a secure two-party computation protocol that securely implements the function f in Sec. 5 in the semi-honest model. Let π^{PSI} be a secure PSI protocol in the semi-honest model. Then the protocol in Fig. 6 securely evaluates the t-PSI-CA functionality.*

Proof. If π^{TST} and π^{PSI} are secure, there exist simulators for the participants in both protocols. We can use them as subroutines to construct our simulators.

Server: The view of a real protocol execution contains the input set S, the random coins, the view of π^{TST}, the view of π^{PSI} ($\mathsf{View}_S = (S, r^s, \mathsf{View}^{\pi^{\mathsf{TST}}}_{\mathtt{SVR}}, \mathsf{View}^{\pi^{\mathsf{PSI}}}_{\mathtt{SVR}})$). Given S, the simulator Sim_S picks coins r^s uniformly at random, chooses κ uniformly at random from $\{0,1\}^\lambda$, computes $S^\kappa = \{s_i||\kappa\}_{n_s}$. Sim_S then invokes the simulator for the underlying protocols to obtain $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{SVR}}((\kappa, S), \perp)$ and $\mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{SVR}}(S^\kappa, \perp)$. Sim_S outputs $(S, r^s, \mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{SVR}}((\kappa, S), \perp), \mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{SVR}}(S^\kappa, \perp))$ as the simulated view. Because κ is identically distributed as in the real execution, so will S^κ. Thus by the security of π^{TST} and π^{PSI}, $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{SVR}}((\kappa, S), \perp), \mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{SVR}}(S^\kappa, \perp)$ are computationally indistinguishable from $\mathsf{View}^{\pi^{\mathsf{TST}}}_{\mathtt{SVR}}, \mathsf{View}^{\pi^{\mathsf{PSI}}}_{\mathtt{SVR}}$.

Client: The view of a real protocol execution contains the input set C, the random coins,

the view of π^{TST}, the view of π^{PSI} ($\mathsf{View}_C = (C, r^c, \mathsf{View}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}, \mathsf{View}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}})$).

- If $|C \cap S| \geq t$, the simulator Sim_C is given C and $C \cap S$ as input. Sim_C selects κ' from $\{0,1\}^\lambda$ and coins r^c uniformly at random, and computes $C^{\kappa'}$ and $C^{\kappa'} \cap S^{\kappa'}$ accordingly. Sim_C invokes the simulator for π^{PSI} and π^{TST} to obtain respectively the simulated views $\mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}}(C^{\kappa'}, C^{\kappa'} \cap S^{\kappa'})$ and $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}(C, (\kappa', |C \cap S|))$. Sim_C then outputs $C, r^c, \mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}}(C^{\kappa'}, C^{\kappa'} \cap S^{\kappa'})$ and $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}(C, (\kappa', |C \cap S|))$ as the simulated view. Because κ' is identically distributed as κ (which is selected uniformly at random by the server in the real protocol), $\mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}}(C^{\kappa'}, C^{\kappa'} \cap S^{\kappa'})$, and $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}(C, (\kappa', |C \cap S|))$ are identically distributed as $\mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}}(C^{\kappa}, C^{\kappa} \cap S^{\kappa})$ and $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}(C, (\kappa, |C \cap S|))$. By the security of π^{PSI} and π^{TST}, the simulated view is computationally indistinguishable from the real one.
- If $|C \cap S| < t$, Sim_C is the same as above except replacing $\mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}}(C^{\kappa'}, C^{\kappa'} \cap S^{\kappa'})$ with $\mathsf{Sim}^{\pi^{\mathsf{PSI}}}_{\mathtt{clt}}(C^{\kappa'}, \perp)$ and $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}(C, (\kappa', |C \cap S|))$ with $\mathsf{Sim}^{\pi^{\mathsf{TST}}}_{\mathtt{clt}}(C, |C \cap S|)$.

In every case, the view can be simulated. Thus the protocol in Fig. 6 is secure in the semi-honest model. □

6.3 PSI with Access Structure

Sec. 5.5 shows how to construct STAS by replacing threshold secret sharing in TST with a linear secret sharing scheme. Following the same vein, we obtain PSI with *expressive* access structure easily by replacing threshold secret sharing in t-PSI-CA with linear secret sharing. The largely repetitive construction is omitted.

7 Evaluation

Both the computational cost and communication cost of GBF* are linear in $n = |I_s|$, while the OPE-based one is quadratic. We thus only implemented[1] the GBF*-based $\mathrm{OT}^{|I_s \cap I_r|}_{I_r}$ in C, and evaluated its performance. The major modification we did is adding OPPRF before GBF related operations. We use existing Paillier encryption implementation[2] for OPPRF. We consider $|I_r| = n$ here.

The experiment is conducted on a virtual machine running Ubuntu 12.04 LTS, allotted 2GB memory and 2 CPUs. Both the client and the server program are run on this virtual machine. The host machine is running Windows 8.1, with 2 Intel(R) Core(TM) i5-4590 3.30GHz CPUs, and 8GB RAM. We only implemented a single thread version but we remark that both OPPRF and GBF* are easily parallelizable. We expect the resulting $\mathrm{OT}^{|I_s \cap I_r|}_{I_r}$ retains such property.

The major bottleneck of our GBF*-based $\mathrm{OT}^{|I_s \cap I_r|}_{I_r}$ is Paillier encryption, decryption, and computing $\mathsf{Enc}_{pk}(c \cdot m)$ in OPPRF. For a naïve implementation of OPPRF with n elements, the client needs to encrypt n elements in its set X, together with n random number r_c. The server also needs to encrypt n random elements r_s. However, these values do not depend on the input of the other party, so both the client and the server can *precompute* these values before starting the whole protocol. Our implementation exploits this observation. When a more efficient OPPRF is available, the efficiency of our protocol will be improved correspondingly.

We first fix the key length for Paillier encryption to be 1024-bit, and the security of GBF* to 80-bit, per NIST suggestion[3] that factorization-based cryptography with 1024-bit key length has 80-bit security. We vary the set size n to be $64, 128, 256, 512, 1024, 2048$, and measure the execution time of GBF*-based $\mathrm{OT}^{|I_s \cap I_r|}_{I_r}$ construction. We follow the approach of [17] to use randomly generated sets to conduct evaluation. For a higher level of security, we fix Paillier key length to 3072-bit and the security of GBF* to 128-bit. The result is shown in Table 2 and Figure 7, where all the reported execution times are the average running time of 5 independent executions. We can see that the computation time increases linearly with the set size. When the access structure is simple, *e.g.*, the set size is < 20, $\mathrm{OT}^{|I_s \cap I_r|}_{I_r}$ terminates around half a second at 80-bit security. At 128-bit security, the protocol finishes in a few seconds.

8 Further Applications

8.1 Private Matchmaking

In dating apps, users are mostly strangers to each other. They may not want to reveal their (potentially sensitive)

1 We modify the GBF source code from Dong at https://personal.cis.strath.ac.uk/changyu.dong/PSI/PSI.html

2 http://acsc.cs.utexas.edu/libpaillier

3 http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf

Set size	80-bit security (ms)	128-bit security (ms)
64	**1769.30**	**24811.71**
128	3385.99	49184.89
256	6582.20	96890.47
512	13080.66	199575.18
1024	25737.88	400866.31
2048	51649.81	778964.15

Table 2. Execution time under various set sizes and security levels

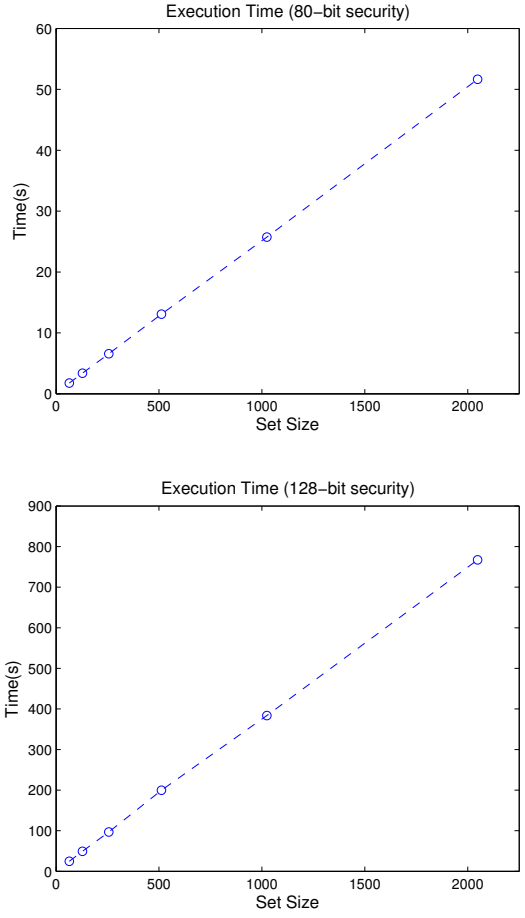

Fig. 7. Execution time (a) 80-bit security / (b) 128-bit security

attributes to any other users. On the other hand, two users are willing to share information when a "match" is found, *i.e.*, when they share a lot of attributes in common. TST, or STAS, supports privacy-preserving matchmaking in a straightforward manner.

In more details, the attribute set is treated as input to TST or STAS. User Alice chooses a random secret value κ such that another user Bob can learn κ only if his attribute set overlaps with that of Alice to a certain degree specified by Alice. (In case of STAS, the attribute set of Bob must satisfy the policy specified by Alice.) Such a secret value serves as an access control token for further interaction between Alice and Bob.

A nice observation here is that, the asymmetry in the roles of the client and the server in our protocol may actually be useful in the context of private matchmaking. For example, a business model of paying user (client) and free user (server) may be employed since the client will know if there is a match and hence has the choice to contact the other party (server) upon receipt of the secret κ (which can be the profile picture).

Moreover, even if the threshold is not reached, the users may know to what extent they are similar, which can be a *useful feature* allowing the users to adjust their expected similarity level for future matching.

It is also possible to store the shares of multiple secrets corresponding to different policies through a *single* invocation of our protocol. For example, a requester who satisfies only the gender criteria can get an $((1,1)$ share of) pseudonym with a $(1,2)$ share of a real name. When another criteria is satisfied, the corresponding slot will contain only the other $(1,2)$ share of the real name. This gives great flexibility for matchmaking apps.

Our current non-optimized construction of TST shows reasonable performance. A dating site eHarmony uses a questionnaire consisting of 258 questions requiring 18 hours from each user for building a detailed profile[4]. We expect around 200 attributes suffice since some of the questions are for consistency check. Figure 7a shows that matching (without reconstructing the secret) by our TST construction finishes in just a few seconds.

8.2 Oblivious Transfer with Access Control

OT with access control (OTAC) is introduced by Camenisch *et al.* [8]. Their construction supports conjunctive policy and is based on a specific construction which covers the credential and the encryption mechanisms.

Our TST enables for the first time OT with threshold access control, in a modular manner. The initial setup and execution of our approach are similar to those considered by Camenisch *et al.* [8]. The server encrypts each data item p_i by a homomorphic encryption into e_i under the server public key. The server publishes all these ciphertexts. The client can use private informa-

4 https://en.wikipedia.org/wiki/EHarmony

tion retrieval (PIR) technique to get e_i of interest, re-randomizes it by a factor r, and sends it to the server.

Now, the decryption result of this ciphertext is treated as the secret κ of the TST protocol, to be transferred to the client. If the attributes of the client satisfy the threshold policy, $r \cdot p_i$ will be transferred.

We need some zero-knowledge proofs, in particular, to prove that the re-randomized ciphertext is originated from the server. This can be done by proving that it is a ciphertext signed by the server which is then randomized by a factor of r, without revealing the signature or r. Also, we need to add the proof for showing that the credential of the client is certifying the attributes which are used as the selection strings in TST. Remarkably, one can plugin any credential scheme and encryption scheme supporting the corresponding zero-knowledge proofs efficiently, which are abundant. We think that it is a conceptually simpler and possibly more efficient approach, yet we enjoy more expressive policy.

At the application level, this class of primitive can find applications in pay-per-download music repository, pay-per-retrieval DNA database, *etc.* For example, a specific solution based on PIR instead of OT with an integrated payment system has been proposed with these kinds of e-commerce applications in mind [25].

8.3 Publish/Subscribe System

As our protocol is for matching in general, it can also find applications in other scenarios where the transfer of material is based on matching of interests, and the interest may be sensitive to disclose. One example is privacy-preserving content-based publish/subscribe system [36]. For matching interest while preserving its privacy, existing work resorts to computation over encrypted data [36], or uses attribute-based mechanism for specifying interest as the encryption policy [47].

Our solution can act as a handy tool in a peer-to-peer setting, where there is no authority to setup the attribute-based cryptosystem. Moreover, all our constructions avoid relatively heavyweight pairing operations that are common in attribute-based systems.

8.4 Existing PSI(-CA) Applications

TST serves as a better alternative in typical applications utilizing PSI. For example, PSI has been used to enforce fog of war in real-time online games [7]: a player can only see the details of the map as determined by the other player, only if the player has a unit of army nearby. Existing schemes rely on the game logic to determine this nearby condition. TST/STAS allows cryptographic revelation of the fog of war under more flexible criteria, *e.g.*, different units with different visibility.

Consider another example of joint promotion by two companies which share some common customers. They should protect the customer lists before confirming that cooperation is beneficial (say, knowing the number of common customers is over a threshold). PSI-CA can be a solution. However, not all customers are equal, a company may assign different weights according to purchase history, *etc.* In general, STAS provides a flexible yet privacy-preserving solution: a company can specify the policy for matching, with the policy kept private.

Applications like location-based services and biometric pattern matching, *etc.*, benefit from our new primitive as we allow more fine-grained control over "proximity" and "similarity", which is not easily supported by traditional PSI-based solutions.

9 Conclusion and Future Work

We formulate the notion of secret transfer with access structure (STAS), and propose two constructions based on oblivious polynomial evaluation and a new variant of garbled Bloom filter respectively. The first scheme is secure in the malicious model, while the second one is more efficient. We then show how to use STAS to construct private set-intersection with access structure, which is the first of its kind to the best of our knowledge. Further applications of STAS are also discussed.

This work leaves a number of open problems: improving efficiency of our malicious model construction, considering stronger models such as universal composability, and constructing t-PSI which does not leak the size of intersection, without resorting to obfuscation. We hope to see our work enables more new applications.

10 Acknowledgements

Sherman Chow is supported by the General Research Fund Grant No. 14201914, the Early Career Scheme, and the Early Career Award of the Research Grants Council, Hong Kong SAR (CUHK 439713).

References

[1] A. Abadi, S. Terzis, and C. Dong. O-PSI: delegated private set intersection on outsourced datasets. In *IFIP SEC*, 2015.

[2] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD*, 2003.

[3] G. Ateniese, E. D. Cristofaro, and G. Tsudik. (if) size matters: Size-hiding private set intersection. In *PKC*, 2011.

[4] A. Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Technion, Israel, 1996.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[6] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. H. M. Smid, and Y. Tang. On the false-positive rate of Bloom filters. *Inf. Process. Lett.*, 108(4):210–213, 2008.

[7] E. Bursztein, M. Hamburg, J. Lagarenne, and D. Boneh. Openconflict: Preventing real time map hacks in online games. In *IEEE Symp. on Security and Privacy, S&P*, 2011.

[8] J. Camenisch, M. Dubovitskaya, and G. Neven. Oblivious transfer with access control. In *ACM CCS*, 2009.

[9] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, 2003.

[10] J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *FC*, 2009.

[11] E. D. Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, 2010.

[12] E. D. Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *FC*, 2010.

[13] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. In *ACNS*, 2009.

[14] P. D'Arco, M. I. G. Vasco, A. L. P. del Pozo, and C. Soriente. Size-hiding in private set intersection: Existential results and constructions. In *AFRICACRYPT*, 2012.

[15] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.

[16] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, 2005.

[17] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS*, 2013.

[18] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, 2005.

[19] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[20] O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *STOC*, 1985.

[22] C. Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs. In *TCC Part-II*, 2015.

[23] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, 2008.

[24] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC*, 2010.

[25] R. Henry, F. G. Olumofin, and I. Goldberg. Practical PIR for electronic commerce. In *ACM CCS*, 2011.

[26] S. Hohenberger and S. A. Weis. Honest-verifier private disjointness testing without random oracles. In *PET*, 2006.

[27] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[28] S. Jarecki and X. Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, 2009.

[29] A. Juels and M. Sudan. A fuzzy vault scheme. *Des. Codes Cryptography*, 38(2):237–257, 2006.

[30] F. Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *ASIACCS*, 2012.

[31] L. Kissner and D. X. Song. Privacy-preserving set operations. In *CRYPTO*, 2005.

[32] I. Komargodski and M. Zhandry. Cutting-edge cryptography through the lens of secret sharing. In *TCC Part II*, 2016.

[33] J. Lai, R. H. Deng, and Y. Li. Expressive CP-ABE with partially hidden access structures. In *ASIACCS*, 2012.

[34] A. B. Lewko and B. Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, 2011.

[35] P. D. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange. *J. Cryptology*, 19(1):27–66, 2006.

[36] M. Nabeel, N. Shang, and E. Bertino. Efficient privacy preserving content based publish subscribe systems. In *SACMAT*, 2012.

[37] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov. BotGrep: Finding P2P bots with structured graph analysis. In *USENIX Security*, 2010.

[38] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[39] T. Nishide, K. Yoneyama, and K. Ohta. Attribute-based encryption with partially hidden encryptor-specified access structures. In *ACNS*, 2008.

[40] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[41] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security*, 2015.

[42] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX Security*, 2014.

[43] M. D. Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. *J. Comput. Syst. Sci.*, 72(6):978–1001, 2006.

[44] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[45] V. Shoup. Practical threshold signatures. In *EUROCRYPT*, 2000.

[46] Q. Ye, R. Steinfeld, J. Pieprzyk, and H. Wang. Efficient fuzzy matching and intersection on private datasets. In *ICISC*, 2009.

[47] T. H. Yuen, W. Susilo, and Y. Mu. Towards a cryptographic treatment of publish/subscribe systems. *Journal of Computer Security*, 22(1):33–67, 2014.

[48] Y. Zhao and S. S. M. Chow. Are you the one to share? Secret transfer with access structure. IACR ePrint, 2015/929.

Appendix A Secure 2PC

We use the simulation-based security definition for two-party computation (2PC). More details can be referred to [20]. 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 $f_2(x,y)$.

A.1 The Semi-Honest Setting

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 other party's input.

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 a party's *view* 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 6 (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*

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

A.2 The Malicious Setting

The adversary might deviate from the protocol execution arbitrarily even at the risk of being caught cheating. The standard way of defining security in such a malicious model is to formalize an ideal process that involves a trusted third party who computes the protocol result directly. A protocol is said to be secure if any adversary in the real protocol execution can be simulated by a simulator in the ideal model. More formally:

In an *ideal execution*, the two parties submit their inputs to the trusted party who will compute the desired output and send the outputs back. An honest party just directs its true input for the computation to the trusted party, while a malicious party may replace its input with any other value of the same length. As in most of the literature, we do not consider fairness, which means the malicious party can send an abort symbol $\perp$ to instruct the trusted party not to deliver the output to the honest party. Let $f = (f_1, f_2)$ be a deterministic function, and $\mathcal{A}_i$ be a PPT adversary that corrupts party $i \in \{1,2\}$, the *ideal execution of f* on inputs (x,y) and auxiliary input z to $\mathcal{A}$, denoted by $\mathsf{Ideal}_{f,\mathcal{A}_i(z)}(x,y)$, is defined as the output pair of the honest party and $\mathcal{A}_i$.

In the *real model*, the honest party follows the instructions of the protocol π to interact with $\mathcal{A}_i$, who on the other hand may follow any polynomial-time strategy. Let $f, \mathcal{A}_i$ to be the same as defined above and let π be a 2PC protocol for computing f. Then, the *real execution of π* on inputs (x,y), auxiliary input z to $\mathcal{A}_i$, denoted by $\mathsf{Real}_{\pi,\mathcal{A}_i(z)}(x,y)$, is defined as the output vector of the honest party and the adversary $\mathcal{A}_i$.

Security. Security is defined by requiring that adversaries (often called simulator in this context) are able to simulate the protocol execution in the real world.

Definition 7 (Malicious Model)**.** *With the definitions above, protocol π is said to securely compute f with abort in the presence of malicious adversaries if for every PPT adversary $\mathcal{A}_i$ in the real model, there exist a PPT adversary Sim_i in the ideal model, where for $i \in \{1,2\}$*

$$\{\mathsf{Ideal}_{f,\mathsf{Sim}_i}(x,y)\} \stackrel{c}{\equiv} \{\mathsf{Real}_{\pi,\mathcal{A}_i}(x,y)\}.$$

Appendix B GBF* Algorithms

This section gives the details and analyze our GBF* which is $(\mathsf{BuildGBF}^*, \mathsf{QueryGBF}^*, \mathsf{GBF}^*\mathsf{Intersection})$. The false positive of GBF* means when querying $\mathrm{GBF}^*_{X,I}$ with some index $i' \notin I$, $\mathsf{QueryGBF}^*()$ (Algorithm 2) returns some element $x \in X$. It is easy to see that our $\mathsf{BuildGBF}^*()$ (Algorithm 1) fails only when emptySlot remains unchanged before line 22. Following existing analysis [17], we have:

Theorem 6. *Algorithm 1 fails with probability* $\mathsf{negl}(\lambda)$.

Theorem 7. *The false positive probability is* $\mathsf{negl}(\lambda)$.

The underlying idea of our $\mathsf{GBF}^*\mathsf{Intersection}$ (Algorithm 3) is very similar to the $\mathsf{GBFIntersection}$ algorithm [17]. Thus Algorithm 3 inherits the corresponding

Algorithm 1 BuildGBF$^*(n, H, \lambda)$

Input: A secret set X, an indexing set I, n, λ, and λ uniform hash functions $H = \{h_1, \ldots, h_\lambda\}$
Output: An (n, H, λ)-$GBF^*_{X,I}$

procedure
  Set $m = \lambda n \log_2 e$
  $GBF^*_{X,I}$ = new m-element array of λ-bit strings
  for $j \leftarrow 1, m$ **do**
    $GBF^*_{X,I}[j]$ = NULL
  end for
  for each $x_j \in X$ **do**
    emptySlot $= -1$, finShare $= x_j$
    for $\ell \leftarrow 1, \lambda$ **do**
      $\boxed{index = h_\ell(i_j)}$
      if $GBF^*_{X,I}[index] ==$ NULL **then**
        if emptySlot$== -1$ **then**
          emptySlot $= index$
        else
          $GBF^*_{X,I}[index] \xleftarrow{\$} \{0,1\}^\lambda$
          finShare=finShare$\oplus GBF^*_{X,I}[index]$
        end if
      else
        finShare=finShare $\oplus$ $GBF^*_{X,I}[index]$
      end if
    end for
    $GBF^*_{X,I}$[emptySlot] = finShare
  end for
  for $j \leftarrow 1, m$ **do**
    if $GBF^*_{X,I}[j] ==$ NULL **then**
      $GBF^*_{X,I}[j] \xleftarrow{\$} \{0,1\}^\lambda$
    end if
  end for
  return $GBF^*_{X,I}$
end procedure

Algorithm 2 QueryGBF$^*(GBF^*_{X,I}, i', k, H)$

Input: An (n, H, λ)-secret embedding garbled Bloom filter $GBF^*_{X,I}$, $\lambda, H = \{h_1, \ldots, h_\lambda\}$
Output: An element $x \in X$ if $i' \in I$, a random string otherwise

procedure
  $\tilde{x} = \{0\}^\lambda$
  for $\ell \leftarrow 1, \lambda$ **do**
    $index = h_\ell(i')$
    $\tilde{x} = \tilde{x} \oplus GBF^*_{X,I}[index]$
  end for
  return $\tilde{x}$
end procedure

Algorithm 3 GBF*Intersection$(GBF^*_{X,I}, BF_{I'})$

Input: An (n, H, λ)-secret embedding garbled Bloom filter $GBF^*_{X,I}$, an (n, H, λ)-Bloom filter $BF_{I'}$
Output: (n, H, λ)-$GBF^*_{\tilde{X},I\cap I'}$

procedure
  Set $m = \lambda n \log_2 e$
  $GBF^*_{\tilde{X},I\cap I'}$ = new m-element array of λ-bit strings
  for $j \leftarrow 1, m$ **do**
    if $BF_{I'}[j] == 1$ **then**
      $GBF^*_{\tilde{X},I\cap I'}[j] = GBF^*_{X,I}[j]$
    else
      $GBF^*_{\tilde{X},I\cap I'}[j] \xleftarrow{\$} \{0,1\}^\lambda$
    end if
  end for
  return $GBF^*_{\tilde{X},I\cap I'}$
end procedure

theorems for GBFIntersection algorithm. We omit the proofs for the sake of simplicity.

Theorem 8. *For $GBF^*_{\tilde{S},I\cap I'}$ produced by Algorithm 3. Let a_ℓ be the event that $GBF^*_{\tilde{S},I\cap I'}[h_\ell(i_j)]$ equals the ℓ^{th} share of s_j, $1 \le \ell \le k$; then (i) $\forall i_j \in I \cap I' : \Pr[a_1 \wedge \ldots \wedge a_\ell] = 1$ (ii) $\forall i_j \notin I \cap I' : \Pr[a_1 \wedge \ldots \wedge a_k]$ is* negl(k).

Theorem 9. *Given sets I, I' and their intersection $I \cap I'$, let $\tilde{S} \subseteq S$ be a set such that $s_j \in \tilde{S}$ if and only if $i_j \in I \cap I'$. Let $GBF^*_{\tilde{S},I\cap I'}$ be the output of the Algorithm 3 from $GBF^*_{S,I}$ and $BF_{I'}$, let $GBF^{*\prime}_{\tilde{S},I\cap I'}$ be another GBF* produced by Algorithm 1 using $\tilde{S}$ and $I \cap I'$, then $GBF^*_{\tilde{S},I\cap I'} \overset{c}{\equiv} GBF^{*\prime}_{\tilde{S},I\cap I'}$.*

Appendix C (ST)^2

The TST (and STAS) construction in Sec. 5 always leaks the intersection size to the client. The main reason is that the reconstruction algorithm of plain secret sharing scheme typically assumes that the inputs are all "correct shares". We are thus forced to release information about the correct shares to let the client identify them. Such ability in turn implicitly leaks the intersection size.

To fix this gap, we formalize a stronger notion called *strong threshold secret transfer* ((ST)^2). By replacing TST with (ST)^2, we obtain t-PSI instead of t-PSI-CA.

Definition 8 (Strong threshold secret transfer ((ST)^2))**.** *(ST)^2 is a two-party computation protocol that implements the following functionality:*

$$f'(x,y) = \begin{cases} (\perp, \kappa) & \text{if } |C \cap S| \geq t \\ (\perp, \perp) & \text{otherwise} \end{cases}$$

where $x = (\kappa, S)$ is from server, $y = C$ is from client.

In $(\mathrm{ST})^2$, the server remains oblivious about the client's input set, but the client only learns if $|C \cap S| \geq t$ or not.

Here we use function secret sharing (equivalent to indistinguishability obfuscation, iO) [32] to construct $(\mathrm{ST})^2$. Another $(\mathrm{ST})^2$ construction based on anonymous ciphertext-policy attribute-based encryption can be found in the full version [48].

Definition 9 (Function secret sharing [32])**.** *Let $\mathcal{F} = \{F : 2^{\mathcal{P}_n} \to \{0,1\}^*\}$ be a class of functions. A function secret sharing scheme for $\mathcal{F}$ consists of a setup procedure* Setup *and a reconstruction procedure* $\mathsf{Reconstruct}$ *that satisfy the following requirements:*

1. $\mathsf{Setup}(1^\lambda, F)$ *gets as input a function $F \in \mathcal{F}$, and distributes a share for each party. For $i \in [n]$, we use $\Pi(F, i)$ to denote the random variable that corresponds to the share of party P_i. Furthermore, for $X \subseteq \mathcal{P}_n$, we denote by $\Pi(F, X)$ the random variable that corresponds to the set of shares of parties in X.*
2. ***Completeness:*** $\mathsf{Reconstruct}(1^\lambda, \Pi(F, X))$ *gets as input the shares of some subset X of parties, and outputs $F(X)$. More precisely,*

$$\Pr[\mathsf{Reconstruct}(1^\lambda, \Pi(F, X)) = F(X)] = 1,$$

   where the probability is over the internal randomness of the scheme and of $\mathsf{Reconstruct}$*.*
3. ***Indistinguishability of the function:*** *For every PPT algorithm D, every equal size $F_0, F_1 \in \mathcal{F}$ and $X \subseteq 2^{\mathcal{P}_n}$ such that $F_0(X') = F_1(X')$ for all $X' \subseteq X$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that*

$$|\Pr[D(1^\lambda, \Pi(F_b, X))] - 1/2| \leq \mathsf{negl}(\lambda),$$

   where the probability is over the internal randomness of the scheme, the internal randomness of D and $b \leftarrow \{0,1\}$ chosen uniformly at random.

We define a function F' that additionally verifies the correctness of input shares. Intuitively, only when the "correct shares" constitute a qualified set according to the prescribed access structure, the secret κ is revealed (*i.e.*, $F'(X') = \kappa$ iff. X' contains some qualified set). Due to the indistinguishability of function secret sharing, the actual number of "correct shares" is not leaked to the client. The details of the construction are listed in Figure 8. The straightforward security proof is omitted.

Protocol: $(\mathrm{ST})^2$ from $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ and iO

Input: (Client) $C = \{c_i\}_{n_c}$, (Server) $S = \{s_i\}_{n_s}$, κ.

1. The server runs $\mathsf{Setup}(1^\lambda, F')$ algorithm to obtain a set of share $\Pi(F, i)$ for $i \in [n_s]$.
2. The server and the client engage in an $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ protocol. The server acts as the sender with input $x = (\{\Pi(F, i)\}_{n_s}, S)$. The client acts as the receiver with input C.
3. The client tries to recover κ by running $\mathsf{Reconstruct}$ on the shares from the last step.

Fig. 8. $(\mathrm{ST})^2$ Construction from $\mathrm{OT}_{I_r}^{|I_s \cap I_r|}$ and iO

Theorem 10. *If iO exists and the $OT_{I_r}^{|I_s \cap I_r|}$ protocol is secure, the protocol in Fig. 8 securely evaluates the $(ST)^2$ functionality in the semi-honest model.*

Appendix D Proof of Theorem 2

Proof. We separately consider corrupt sender and receiver. Our proof consists of sequences of games transiting from a real protocol execution to an idealized execution in which the simulator can only obtain information about the honest party's input from an ideal functionality. We show that each transition is indistinguishable by the security of the underlying primitives.

Corrupt receiver. We need to prevent a malicious receiver from learning extra information about the sender's input. A non-zero degree k polynomial has at most $k-1$ roots. Intuitively, by extracting its coefficients from ZKPoK_1, the simulator is able to hand the actual input of the adversary to the ideal functionality.

Game $\mathbf{H}_0$: This game is the real execution. The simulator $\mathcal{S}_0$ plays the role of an honest sender with input (I_s, E).

Game $\mathbf{H}_1$: In this game, if $\mathcal{S}_1$ does not abort in Step 5, it uses the extractor to extract $\mathcal{A}$'s input polynomial $p(\cdot)$ from ZKPoK_1. $\mathcal{S}_1$ sets $\tilde{I}_r = \{r_i : p(r_i) = 0\}$ and completes $\tilde{I}_r$ to size n_r by adding distinct random elements. $\mathcal{S}_1$ only keeps this set internally so that $\mathcal{A}$ does not notice the difference between $\mathcal{S}_1$ and $\mathcal{S}_0$.

Game $\mathbf{H}_2$: This game is the same as **Game $\mathbf{H}_1$**, except that $\mathcal{S}_2$ replaces ZKPoK_2 with simulations. Any

party distinguishing this game from the previous one breaks the zero-knowledge property of the proof system.

Game $\mathbf{H}_3$: In this game, $\mathcal{S}_3$ encrypts dummy elements "1" at the beginning. If there exists a distinguisher that can tell apart this game from the previous one, we can build a distinguisher breaking the CPA-security of $(\widetilde{\mathsf{KeyGen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ similar to what we did in the proof of Theorem 1. Now ZKPoK_2 is "proving" false statements.

Game $\mathbf{H}_4$: Now we make the final important transition. Instead of using the true inputs (I_s, E) to compute c_i, $\mathcal{S}_4$ hands $\tilde{I}_r$ to the ideal functionality, whose inputs are (I_s, E), and receives answer $E' = \{e_i\}$. Then $\mathcal{S}_4$ completes E' by adding random element to size n_s, and encrypts each element $e_i \in E'$ as c_i. It is obvious that c_i are identically distributed in this game and the previous game, so $\mathcal{A}$ cannot tell the difference.

Game $\mathbf{H}_5$: This is identical to **Game $\mathbf{H}_4$** except that $\mathcal{S}$ does not get the true input (I_s, E) at the beginning. It is a conceptual change since $\mathcal{S}_4$ does not use (I_s, E) at all in **Game $\mathbf{H}_4$**. $\mathcal{S}$ works exactly in the idealized execution of the protocol, which completes our argument.

Corrupt sender. We need to prevent $\mathcal{A}$ from learning anything about the honest receiver's input, and from making the receiver output incorrect answer. The former one is from the security of encryption. The latter is achieved by having $\mathcal{A}$ first commit its input, and later prove in zero-knowledge that it follows the protocol.

Game $\mathbf{H}_0$: This game is the real execution. The simulator $\mathcal{S}_0$ plays the role of an honest receiver with input I_r.

Game $\mathbf{H}_1$: We now replace the public key in the common reference string with $(\tilde{pk}, \tilde{sk}) \leftarrow \widetilde{\mathsf{KeyGen}}$. When $\mathcal{S}_1$ receives encrypted ciphertext $(\tilde{s_i}, \tilde{e_i})$, it decrypts them using $\tilde{sk}$ and records the result (s_i, e_i). This change is purely conceptual to $\mathcal{A}$'s point of view.

Game $\mathbf{H}_2$: This game replaces ZKPoK_1 with simulations. Any party distinguishing this game from **Game $\mathbf{H}_1$** breaks the zero-knowledgeness of the proof system.

Game $\mathbf{H}_3$: In this game, instead of computing the encrypted polynomial according to input I_r, $\mathcal{S}_3$ just prepares $n_r + 1$ encryptions of zeros using fresh randomness. Any distinguisher telling apart this game from the previous one can be converted into a distinguisher breaking the CPA-security of homomorphic encryption. Notice that $\mathcal{S}_3$ gets to see all $\mathcal{A}$'s input (I_s, E) after decryption at the end of the protocol. If these two sets are not consistent with the records obtained by decrypting $(\tilde{s_i}, \tilde{e_i})$, $\mathcal{S}_3$ aborts. The probability of $\mathcal{S}_3$ aborts should be negligible by the soundness of ZKPoK_2. If $\mathcal{S}_3$ does not abort, it computes the output from I_r and (I_s, E).

Game $\mathbf{H}_4$: We replace the final step of $\mathcal{S}_3$ with the following: $\mathcal{S}_4$ hands (I_s, E) to an ideal functionality, whose input is I_r. This step is purely conceptual to $\mathcal{A}$ by definition of an ideal functionality, so it should not affect $\mathcal{A}$'s protocol view or the output distribution.

Game $\mathbf{H}_5$: We make the final transition that $\mathcal{S}$ does not know the input I_r at the beginning of the protocol. It is a conceptual change because in **Game $\mathbf{H}_4$**, $\mathcal{S}_4$ does not use I_r to compute anything at all. This game is identical to an idealized execution of the protocol. □