Bounded Size-Hiding Private Set Intersection

Tatiana Bradley Sky Faber Gene Tsudik

University of California, Irvine

Abstract. Private Set Intersection (PSI) and other private set operations have many current and emerging applications. Numerous PSI techniques have been proposed that vary widely in terms of underlying cryptographic primitives, security assumptions as well as complexity. One recent strand of PSI-related research focused on an additional privacy property of hiding participants' input sizes. Despite some interesting results, only one practical size-hiding PSI (SH-PSI) has been demonstrated thus far [1].

One legitimate general criticism of size-hiding private set intersection is that the party that hides its input size can attempt to enumerate the entire (and possibly limited) domain of set elements, thus learning the other party's entire input set. Although this "attack" goes beyond the honest-but-curious model, it motivates investigation of techniques that simultaneously hide and limit a participant's input size. To this end, this paper explores the design of *bounded size-hiding* PSI techniques that allow one party to hide the size of its input while allowing the other party to limit that size. Its main contribution is a reasonably efficient (quasi-quadratic in input size) *b*SH-PSI protocol based on bounded keyed accumulators. This paper also studies the relationships between several flavors of the "Strong Diffie-Hellman" (SDH) problem.

Keywords: Private Set Intersection, Size Hiding, Bounded Input, Cryptographic Accumulators, SDH Problem

1 Introduction

Private set operations have many potential applications in secure cloud computing and storage, as well as other settings involving mutually suspicious parties that wish to divulge to each other nothing beyond the outcome of a particular set operation. This serves as one motivating factor for research in more efficient and more secure techniques. The other, no less important, factor is intellectual curiosity. There is something inherently appealing about private set operations, perhaps because they represent an interesting and realistic-sounding application domain for secure two-party computation.

The most natural and popular private set operation is Private Set Intersection (PSI), a cryptographic technique that allows two parties, *server* and *client*, to interact such that one or both of them (often, *client*) computes the intersection $S \cap C$ over their respective input sets S and C. Typically, *server* and *client* learn nothing beyond the size of each other's set and the resulting intersection. There are multiple PSI flavors with varying privacy properties, security models, complexities and underlying cryptographic primitives [21, 22, 12, 13, 1, 17, 24, 16, 15, 14, 7, 26, 32, 27, 23].

One recent PSI research direction focused on techniques that additionally hide the input size of one participant. This property is sometimes called *one-sided input size-hiding*. This line of research is attractive because, in general, there are few cryptographic techniques that achieve non-padding-based input size-hiding. (See Section 2 for an overview of related work).

Meanwhile, one important criticism of size-hiding PSI (SH-PSI) is the *unlimited* nature of the size-hiding feature. In scenarios where the overall input domain is small[1], a dishonest *client* can enumerate all (or most) of the possible elements, use them as its input set and thus learn all (or most) of *server*'s input set.

On the one hand, this criticism seems unfair because a *client* that enumerates, and provides as input, elements that it does not actually have, goes beyond the "honest-but-curious" (HbC) adversary model considered in, for example, [1]. On the other hand, it could be that the entire notion of input size-hiding inherently motivates a slightly different adversary model than HbC.

Consequently, the main motivation for this paper is the need to combine hiding of one party's input size with the other party's ability to upper-bound it, i.e., to limit the amount of information potentially learned by the first party. Specifically, the goal is to explore PSI techniques that allow *client* to hide its set size while assuring *server* that it does not exceed some fixed threshold t. At the first glance, it seems that this can be trivially met by modifying current SH-PSI, PSI or similar techniques.

One intuitive approach to bounded size-hiding is to amend any regular PSI protocol by having *client* always pad its (linear-size) input with dummy elements, up to the server-selected upper bound t. While this approach would meet our goals, we consider it to be undesirable, for several reasons:

- Padding by *client* always incurs $O(t)$ computation and bandwidth costs, even if $|C|$ and/or $|S|$ are small relative to t.[2]
- Representation of dummy elements must be indistinguishable from that of their genuine counterparts. This very likely entails generating a random value for every dummy element, which, depending on the underlying PRNG, can involve as little computation as a hash, or as much as a large-integer arithmetic operation.
- If $|C| < t$, a misbehaving HbC *client* can easily cheat – and learn more about S than it is entitled to – by inserting extra actual elements into its input that it could later claim are just dummies.[3]

Even if aforementioned reasons are deemed to be superficial, we still consider padding-based size-hiding techniques to be inelegant.

Another simple way to force boundedness, is to modify any PSI protocol such that *server*, acting unilaterally, uses a subset $S^* \in S$ of no more than t set elements as its PSI input. This implies that *client* would learn an intersection of at most t elements. However, *client* would also very likely learn less than it is entitled to if

[1] For example: age, blood type, birthday, country, zip code, etc.

[2] In contrast, *b*SH-PSI incurs only $O(|C|)$ costs, since *client* can download *server*'s public key only once, ahead of time, i.e., off-line.

[3] As discussed later, although the proposed *b*SH-PSI has the same issue, it discourages *client*'s cheating by imposing a relatively high client computational cost for each additional element in the accumulator, up to the bound.

$|C \cap S^*| < |C \cap S| \leq t$. An equally trivial approach is for *server* to pick a random subset $C^* \in C$ of no more than t set elements (assuming $|C| > t$) of *client*'s input. This is doable since most (not size-hiding) PSI protocols involve a message from *client* to *server* that contains some linear representation of *client*'s input set. The end-result would be the same: *client* would likely learn less than $C \cap S$ even if $|C \cap S| \leq t$.

In this paper, we introduce the notion of Bounded Size-Hiding Private Set Intersection (*b*SH-PSI) and demonstrate the first provably secure and reasonably efficient[4] *b*SH-PSI protocol. In the process, we introduce two new cryptographic SDH-related assumptions and show their equivalence to more established assumptions. Finally, we discuss several *b*SH-PSI extensions and optimizations.

In a general sense, *b*SH-PSI operates as follows: before any interaction, *server* chooses a bound t. During the interaction, *client* inputs a set of size $m < t$ and *server* inputs a set of size n, which is independent of t. At the end of the interaction, *client* learns the intersection of the two sets and n, the *server*'s set size. The *server* learns nothing.

Notable features of proposed *b*SH-PSI include:

- It is particularly well-suited for scenarios where *server* needs to interact with *client* whose input set is larger than *server*'s. However, *b*SH-PSI is effective regardless of *client*'s and *server*'s relative set sizes.
- *Server* can set (and modify at will) the upper bound t on *client*'s input set size. In particular, if set elements are drawn from a small domain, this prevents *client* from enumerating all elements and determining the entirety of *server*'s set.
- It is based on a bounded cryptographic accumulator construct from [31].
- *Client* privacy is unconditional with respect to both set elements and their number, i.e., set size.
- *Server* security holds under the One-Generator [3] and Exponent [33] q-Strong Diffie-Hellman (SDH) Assumptions in the Random Oracle Model (ROM) [2].
- *Server* incurs computational complexity linear in *server*'s input size – $O(n)$ where $n = |S|$.
- *Client* incurs computational complexity of $O(m^2 log^2 m)$ in *client*'s input set size: $m = |C|$. With pre-computation, this can be lowered to $O(m^2)$.
- Overall bandwidth complexity is linear in *server*'s input size – $O(n)$.

Organization: Related work is discussed in Section 2. Section 3 formally defines SH-PSI, its security properties and underlying cryptographic assumptions. A concrete SH-PSI construct is presented in Section 4, along with its security arguments. Section 5 discusses scenario-specific extensions and open problems. Next, Section 6 presents reductions of new cryptographic assumptions to their better-known counterparts. Then, Section 7 details specific methodologies for efficient computation by *client*. The paper concludes with a summary in Section 8.

[4] The term "efficient" is used in the standard sense, i.e., efficient in the context of most cryptographic literature.

2 Related Work

The concept of size-hiding private set intersection (SH-PSI) was introduced by Ateniese *et al.* in [1]. It demonstrated the first SH-PSI technique using RSA accumulators, with unconditional privacy of *client*'s set size and its contents, *server* privacy based on the strong RSA assumption, and correctness in the HbC setting in the random oracle model (ROM).

D'Arco *et al.* [8] (revised in [9]) is the only other effort, to our knowledge, focused on SH-PSI. It demonstrates several results about the possibility of SH-PSI, including one that one-sided SH-PSI is possible in both the standard model and ROM. However, the proposed techniques – which are based on oblivious pseudorandom function (OPRF) evaluation and RSA, require a setup phase using a trusted third party (TTP). The revised version [9] presents a technique that avoids random oracles at the price of a commitment scheme which is unspecified; thus, the exact complexity is unclear.

There have been other efforts to define, and show feasibility of, various size-hiding two-party computation techniques. However, these results are largely theoretical.

Lindell *et al.* [29] prove some results about the feasibility of input-size hiding in two-party computation under various conditions. In particular, one-sided size-hiding is shown to be possible for every function in the HbC model without random oracles, given that the output size is upper-bounded by some function of a party's input size, which is the case in *b*SH-PSI. The concrete protocol presented in [29] is based on fully homomorphic encryption, which is not yet practical. The full version [28] shows how to modify the protocol for the case where one party hides its size and learns the outcome. Size-hiding is achieved by padding *client*'s input with random elements.

Chase *et al.* [6] present an extended definition of the real/ideal model that allows for input-size hiding in the presence of malicious players. The extended model allows one party in the ideal world to send what is called an "implicit representation" of its input (which does not necessarily reveal the input size) in lieu of the input itself. The generic protocol for two-party computation involves five rounds of communication, making use of fully homomorphic encryption. Also, the output size must be fixed, which is not the case in PSI.

Other results discuss the need for input size-hiding in secure computation, starting with Micali *et al.* [30], which introduces the notion of zero-knowledge sets – a size-hiding cryptographic primitive. The protocol allows a party to commit to a private set (with size hidden) and later prove whether a given element is a member of that set. This notion is different from PSI since the element (for which set membership is being tested) is public.

De Cristofaro *et al.* [11] focus on size- and position-hiding private substring matching in the context of genomic privacy. The proposed protocol is highly specialized, in particular, not suitable for generic PSI. Based on additively homomorphic encryption, it allows *client* to test whether a number of substrings are present in *server*'s string (genome) at pre-determined positions, while revealing neither positions nor sizes of the substrings to *server*, and precluding *client* from learning anything about *server*'s input beyond the binary result of the computation.

Ishai and Paskin [25] show that it is possible to securely evaluate branching programs while hiding the size of the program, given that the length of the program is

upper-bounded by some polynomial. In this context, *size* refers the number of instructions in the program, while *length* refers to the length of the longest branch in the program. The protocol in [25] is based on strong oblivious transfer.

Goyal *et al.* [20] show that constant-round public-coin zero knowledge is possible using only black box techniques, while hiding the size of the input string. The protocol is based on a commit-and-prove scheme using extendible Merkle trees.

3 Problem Statement and Preliminaries

We now formally define *b*SH-PSI as well as four relevant cryptographic problems. The latter include two new assumptions: *polynomial-generalized exponent q-SDH* (*PG-E-SDH*) and *polynomial-generalized one-generator q-SDH* (*PG-OG-SDH*) as well as their better-known counterparts: *exponent q-SDH* (*E-SDH*) and *one-generator q-SDH* (*OG-SDH*).

3.1 Bounded SH-PSI

Informally, *b*SH-PSI extends SH-PSI with the requirement that *client* can only input a limited number of set elements. This bound t is fixed by *server* prior to protocol execution. For ease of presentation, we define *b*SH-PSI directly, and refer to [4] and [1] for formal definitions of PSI and SH-PSI, respectively. In the following, $\sim$ denotes computational indistinguishability, as defined in [19].

Definition 1 (Bounded SH-PSI). *A scheme satisfying correctness, boundedness, client privacy and server privacy, (per Definitions 2, 3, 4 and 5, respectively) involving two parties: client $\mathcal{C}$ and server $\mathcal{S}$, and two components: $Setup$ and $Interaction$, where:*

- *$Setup$: an algorithm that selects global parameters, including t and server's public key, if any.*
- *$Interaction$: a protocol between $\mathcal{S}$ and $\mathcal{C}$ on respective inputs: $S = \{s_1, \ldots, s_n\}$ and $C = \{c_1, \ldots, c_m\}$.*

Definition 2 (Correctness). *If both parties are honest and $m \leq t$, then, at the end of $Interaction$ on inputs (S, C) server outputs $\perp$, and client outputs $(n, S \cap C)$.*

Definition 3 (Boundedness). *If client's set size exceeds the bound ($|C| = m > t$), and server is honest, client only learns $n = |S|$.*

Definition 4 (Client Privacy). *For any PPT adversary S^* acting as server on input S' in execution of b*SH-PSI*, we say that Client Privacy holds if the views of S^* are computationally indistinguishable when interacting with any pair of client input sets: $[C^{(0)}, C^{(1)}]$. Specifically, let $View_S(C)$ represent the view of S^* during protocol execution on input C. Then, Client Privacy is:*

$$\forall\ (C^{(0)}, C^{(1)})\ :\ View_S[C^{(0)}) \sim View_S(C^{(1)}]$$

We note that Client Privacy implies that S^* learns no information about C, including m, i.e., Client Privacy includes privacy of *client*'s set elements and of their number.

Definition 5 (Server Privacy). *Let View$_C(C, S)$ be a random variable representing $\mathcal{C}$'s view during execution of b*SH-PSI *on inputs: C, S. We say that Server Privacy holds if there exists a PPT algorithm C^* such that:*

$$\forall\ (C, S) : C^*(C, C \cap S, n) \sim \textit{View}_C(C, S).$$

In other words, for any pair of inputs $[C, S]$, $\mathcal{C}$'s view of the protocol can be efficiently simulated[5] *by C^* on input C and $C \cap\ S$ alone. In particular, this means C^* does not have access to S.*

Security Model. We aim to construct bSH-PSI techniques secure in the HbC model [18]. HbC assumes that, while all parties faithfully follow the protocol, they may try to infer or compute additional information from the protocol transcript(s). However, due to the unusual input-boundedness feature of bSH-PSI, we extend the HbC model for *client* by allowing it to attempt using an input set larger than the *server*-imposed bound t, while still adhering to the rest of the protocol. We refer to this as the HbC* model.[6] In particular, *client*'s messages are assumed to be well-formed. However, the HbC model for *server* is unchanged from its usual form.

3.2 q-Strong Diffie-Hellman Assumptions

As discussed later in the paper, security of the proposed bSH-PSI relies on the hardness of two non-standard cryptographic problems: *polynomial-generalized exponent q-SDH* (*PG-E-SDH*) and *polynomial-generalized one-generator q-SDH* (*PG-OG-SDH*). These are the generalizations of well-known *exponent q-SDH* (*E-SDH*) and *one-generator q-SDH* (*OG-SDH*) problems, to allow polynomials in the group exponent. Each of these problems assumes the same public information derived from the secret z. This public information is a $(q+1)$-tuple: $[g, g^z, \ldots, g^{(z^q)}]$, where all components are $\bmod p$, g is a generator of the p' order subgroup ($\mathbb{G}$) of Z_p^* and p and p' are large primes. (We omit the $\bmod p$ notation from here on).

PG-E-SDH generalizes *E-SDH* to reflect the difficulty of computing g exponentiated with any polynomial in z of degree larger than q, instead of simply z^{q+1}. Similarly, *PG-OG-SDH* generalizes *OG-SDH* to the difficulty of exponentiating a base (not just g) to the power of $\frac{1}{z+c}$. Specifically, it considers base elements of the form g exponentiated to any polynomial in z of degree less than or equal to q.

We now state the problems and then discuss the assumptions on their hardness. Our definitions of the standard problems (1 and 2 below) follow the presentation in [3].

Problem 1 (One-generator q-Strong Diffie-Hellman Problem). Given a $(q+1)$-tuple $[g, g^z, \ldots, g^{(z^q)}]$ as input, the *one-generator q-SDH* problem in $\mathbb{G}$ is to output a pair: $[c, g^{\frac{1}{(z+c)}}]$ where $c \in \mathbb{Z}_{p'}^*$. An algorithm $\mathcal{A}$ has advantage ϵ in solving *one-generator q-SDH* in $\mathbb{G}$ if:

$$\Pr\left[\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}]\big) = [c, g^{\frac{1}{z+c}}]\right] \geq \epsilon$$

[5] Being simulatable means that C^* can output a computationally indistinguishable transcript.

[6] Note that because the adversarial client has more power in the HbC* model than in plain HbC, security also holds in HbC.

where the probability is over the random choice of generator $g \in \mathbb{G}$, the random choice of $z \in \mathbb{Z}^*_{p'}$, and random bits consumed by $\mathcal{A}$.

Problem 2 (Exponent q-Strong Diffie-Hellman Problem). Given a $(q+1)$-tuple $[g, g^z, \ldots, g^{(z^q)}]$ as input, the *exponent q-SDH* problem in $\mathbb{G}$ is to output $g^{(z^{q+1})}$. An algorithm $\mathcal{A}$ has advantage ϵ in solving *exponent q-SDH* in $\mathbb{G}$ if

$$\Pr\left[\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}]\big) = g^{(z^{q+1})}\right] \geq \epsilon$$

where the probability is over the random choice of generator $g \in \mathbb{G}$, the random choice of $z \in \mathbb{Z}^*_{p'}$, and random bits consumed by $\mathcal{A}$.

The following are the two new problems that generalize the two above. We refer to Section 6 for formal reductions.

Problem 3 (Polynomial-generalized one-generator q-Strong Diffie-Hellman Problem). Given a $(q+1)$-tuple $[g, g^z, \ldots, g^{(z^q)}]$ and a polynomial $P_n(z)$ in z of degree $n \leq q$ with known coefficients in $\mathbb{Z}^*_p$ as input, the *polynomial-generalized one-generator q-SDH* problem in $\mathbb{G}$ is to output a pair: $[c, g^{\frac{P_n(z)}{(z+c)}}]$, where $-c$ is not a root of $P_n(z)$. An algorithm $\mathcal{A}$ has an advantage ϵ in solving *polynomial-generalized one-generator q-SDH* in $\mathbb{G}$ if:

$$\Pr\left[\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}], P_n(z)\big) = [c, g^{\frac{P_n(z)}{z+c}}]\right] \geq \epsilon$$

where the probability is over the random choice of generator $g \in \mathbb{G}$, the random choice of $z \in \mathbb{Z}^*_{p'}$, and random bits consumed by $\mathcal{A}$.

Note 1. Note that the *polynomial-generalized one-generator q-SDH* problem described above is not hard if $(z+c)$ divides $P_n(z)$ (i.e., $-c$, the additive inverse of c, is a root) because of the restriction that $n \leq q$. If $-c$ is a root of $P_n(z)$, the problem is equivalent to computing $(c, g^{P'_{n-1}(z)})$, where $P'_{n-1}(z) = \frac{P_n(z)}{z+c}$. This is achievable by exponentiation and multiplication of elements in $[g, g^z, \ldots, g^{(z^q)}]$.

Problem 4 (Polynomial-generalized exponent q-Strong Diffie-Hellman Problem). Given as input a $(q+1)$-tuple $[g, g^z, \ldots, g^{(z^q)}]$ and $P_n(z)$, a polynomial in z of degree $n > q$ (and n being polynomial in the security parameter) with known coefficients in $\mathbb{Z}^*_p$, the *polynomial-generalized exponent q-SDH* problem in $\mathbb{G}$ is to output $g^{(P_n(z))}$. An algorithm $\mathcal{A}$ has an advantage ϵ in solving *polynomial-generalized exponent q-SDH* in $\mathbb{G}$ if:

$$\Pr\left[\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}]\big) = g^{P_n(z)} \text{ s.t. } n > q\right] \geq \epsilon$$

where the probability is over the random choice of generator $g \in \mathbb{G}$, the random choice of $z \in \mathbb{Z}^*_{p'}$, and random bits consumed by $\mathcal{A}$.

Definition 6. *For each of the four q-SDH problems described above, we say that the corresponding (q, t', ϵ)-SDH assumption holds in $\mathbb{G}$ if no t'-time algorithm has advantage at least ϵ in solving that q-SDH problem in $\mathbb{G}$.*

As discussed later, security of our bSH-PSI protocol is based on these assumptions, against polynomial time adversaries with $q = t$, and negligible advantage ϵ.

Group Selection. While there are many candidate groups, we focus on the Diffie-Hellman prime-order integer subgroups modulo a large prime. Specifically, let τ be a security parameter and let $DH.setup(\tau)$ be an algorithm that outputs a triple: (p, p', g) such that: (1) p is a prime of the form $p = 2(p')^l + 1$ for some integer l, (2) p' is a prime, and (3) g is a generator of a subgroup of $\mathbb{Z}_p^*$ of order p'. For more on our choice of group see Section 5.5.

4 Protocol

We now present a concrete bSH-PSI technique, followed by security arguments.

4.1 Protocol Description

We first introduce the building blocks and intuition behind this realization of bSH-PSI. The primary building blocks are: (1) a t-bounded keyed accumulator [31], (2) a keyed unpredictable function $f_{z,X}(c) = X^{\frac{1}{z+c}}$, and (3) two cryptographic hash functions $F(\cdot)$ and $H(\cdot)$ modeled as Random Oracles: $F : \{0,1\}^* \rightarrow \{0,1\}^\omega$ where ω is a security parameter[7], and $H : \{0,1\}^* \rightarrow \{0,1\}^{\log p'}$. For the time being, we assume that $\omega = \log p'$, though, in practice, ω can be smaller.

Intuitively, *client* aggregates its input elements $C = \{c_1, \ldots, c_m\}$ into an accumulator of the form $X' = g^{\prod_{i=0}^{m} hc_i + z}$, where $hc_i = H(c_i)$. *Client* can compute this product using *server*'s public key $[g, g^z, \ldots, g^{(z^t)}]$ by expanding the product in the exponent into the polynomial of the form: $A(z) = a_0 + a_1 z + \ldots + a_m z^m$, where each coefficient a_k is a product-sum of a combination of *client*'s hashed inputs: $hc_1, \ldots, hc_m$.

Each a_k has a closed-form solution dependent only on *client*'s input and t. Optionally, it can be computed before protocol execution. Techniques for efficient computation of this polynomial are presented in Section 7. The resulting accumulator X' is then blinded as $X = X'^r$, (using a fresh random value r) and sent to *server*. Due to this consistent random blinding *client* benefits from unconditional privacy of its input. It also obtains unconditional privacy of its input size since X is $\log p$ bits long. Furthermore, total protocol bandwidth is independent of m.

Upon receipt of X, for each hashed element hs_j, *server* computes a distinct *tag*, denoted tg_j, as the composition of F and $f_{z,X}$. That is: $tg_j = F(X^{\frac{1}{z+hs_j}})$ where $hs_j = H(s_j)$. The resulting set of tags is then sent to *client* who, in turn, uses them to determine the actual set intersection.

Note that $f_{z,X}(hs_j)$ is of the form $g^{\frac{P_m(z)}{z+hs_j}}$ for some polynomial $P_m(z)$. Also, $f_{z,X}(hs_j)$ is unpredictable given public information provided to *client*, if and only if *PG-OG-SDH* assumption holds. Applying $F(\cdot)$ converts these unpredictable values into pseudorandom values, which is essential for server privacy.

Meanwhile (either before receiving *server*'s tags or upon receiving them), *client* computes a tag tg'_i for each hashed element hc_i in its input set. As part of computing

[7] A practical example is SHA-256 for $\omega = 256$.

each tg'_i *client* essentially constructs "witness" X_i for the original accumulator X, based on each hc_i, i.e., X_i is a partial accumulator, with one term missing from the product in the exponent. Specifically, each tg'_i is computed as: $F(\cdot)$ applied to a witness: g exponentiated with a product of $m-1$ binomials of the form $(hc_i + z)$ and the random value r. The product of binomials can be represented by a unique polynomial $A_i(z)$, such that: $A_i = a_{(i,0)} + a_{(i,1)}z + \ldots + a_{(i,m)}z^m$ and $a_{(i,k)}$ is a product-sum involving all of *client*'s hashed input, except hc_i. As mentioned above, *client* tags can be computed ahead of time. The intersection of: $\{tg'_i \mid 0 < i \leq\ m\}$ and $\{tg_j \mid 0 < j \leq\ n\}$, determines *client*'s output: $S \cap\ C$.

Figure 1 shows the $Interaction$ component of this bSH-PSI protocol. $Setup(z,t)$ returns the information extracted from the output of $DH.setup(\tau)$ and the public key $[g, g^z, \ldots, g^{(z^t)}]$ generated from a bound t and secret z. Before the protocol begins, *server* selects t and z and publishes the output of $Setup$.

4.2 Security Analysis

We now present proofs of security for Definitions 2, 3, 4, 5.

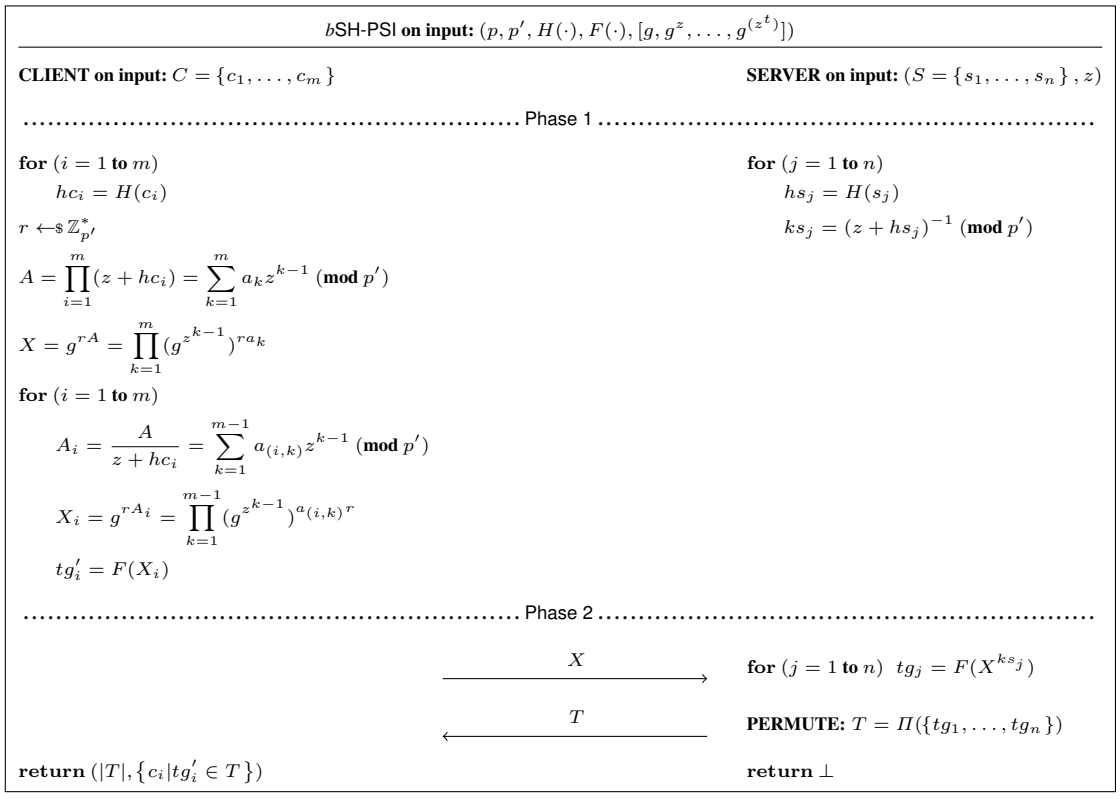

Fig. 1. bSH-PSI Protocol. All computation is (mod p) unless stated otherwise.

Correctness. Following Definition 2, we show that when both parties are honest, *client* outputs $(n, S \cap C)$ and *server* outputs $\perp$, i.e., nothing.

It is easy to see *client* correctly computes n. For every $s_j \in S$, HbC *server* sends exactly one tg_j to *client*. Thus, *client* needs only to count the number of tg_j's received.

To see that *client* correctly computes the intersection, let c_i be an an arbitrary element in *client*'s set, such that $c_i \in S \cap C$. Then, there is some $0 < j \leq n$ such that $c_i = s_j$ and $hc_i = hs_j$. Therefore, tg_j, computed by *server* and sent to *client* matches *client*'s tag tg'_i:

$$tg'_i = F(X_i) = F(g^{r(z+hc_1)\ldots(z+hc_{i-1})(z+hc_{i+1})\ldots(z+hc_m)}) =$$
$$= F(g^{\frac{r(z+hc_1)\ldots(z+hc_m)}{z+hc_i}}) = F(g^{r\frac{A}{z+hc_i}}) = F(X^{\frac{1}{z+hc_i}}) = tg_j.$$

Thus *client* concludes that c_i is in the intersection.

Now consider a *client*'s element $c_k \notin S \cap C$, i.e., there is no j such that $c_k = s_j$. Thus, there is also no j such that $hc_k = hs_j$ and $tg_j = tg'_k$ except for negligible probability, due to collisions in either $F(\cdot)$ or $H(\cdot)$, or degenerate input x such that $(H(x) + z) = 0 \bmod p'$. (If *server* ever detects such an input element, it must change its public key.) Therefore, *client* computes no match and concludes that c_k is not in the intersection.

Boundedness. As described in Definition 3, we need to show that *client* learns only *server*'s input set size if it attempts to input more than t set elements. We note that in order to extract $S \cap C$ with a set of size $m = u$, where $u > t$, *client* must aggregate u elements into X. This follows directly from our security model which requires *client* messages to be well formed. Then, we show that this is impossible under the *PG-E-SDH* assumption. Thus, if *client* is able to extract $S \cap C$ then $m \leq t$.

More formally, we show by contradiction, that constructing a well-formed X, as described, is infeasible. We now assume that *client* can aggregate u elements into X. Then, *client* must have a PPT algorithm $\mathcal{A}$ which – given C, $H(\cdot)$, and $[g, g^z, \ldots, g^{(z^t)}]$ – computes:

$$X = g^{r(z+hc_1)\ldots(z+hc_u)} = g^{(A_0+\ldots+A_{u-1}z^{u-1}+z^u)},$$

where each A_i is a product-sum of values known to *client*: $r, hc_1, ..., hc_u$.

However, computing X is the same as solving the *polynomial-generalized exponent q-SDH* problem on inputs: $[g, g^z, \ldots, g^{(z^t)}]$ and $P_n(z) = A_0 + ... + A_{u-1}z^{u-1} + z^u$, which, based on our assumption, is infeasible since $u > t$. Hence, by contradiction, the embedding is impossible and *client* learns only n.

Client Privacy. The only message sent from *client* to *server* is:

$$X = g^{r(z+hc_1)\ldots(z+hc_m)} \mod p.$$

X is always of this form as an HbC *server* always correctly generates its public key. Since g is a generator of the cyclic subgroup $\mathbb{G} \subset \mathbb{Z}_p^*$ of order p', and no $(z + hc_i)$ is a multiple of p', except for negligible probability, we can assume that $A = g^{(z+hc_1)\ldots(z+hc_u)}$ is also a generator of $\mathbb{G}$. Since r is chosen uniformly, at random, from $\mathbb{Z}_{p'}^*$, X also has a uniform distribution in $\mathbb{G}$.

Thus with overwhelming probability[8], $\text{View}_S(C^0)$ and $\text{View}_S(C^1)$ are two uniformly distributed group elements and are thus indistinguishable. Therefore, Client Privacy holds in the presence of an HbC *server*.

By making one slight modification to the protocol, Client Privacy can be guaranteed unconditionally, regardless of the adversarial model of *server*. To mitigate the possibility of a malicious *server* presenting an invalid public key, *client* can simply verify that: $(g^{(z+hc_1)...(z+hc_m)})^{p'} \mod p = 1$. If so, then g^A is a generator of $\mathbb{G}$ and $X = (g^A)^r$ is uniformly distributed in $\mathbb{G}$. Otherwise *client* aborts the protocol by sending just g^r and ignoring *server*'s response. In either case, Client Privacy is guaranteed.

Server Privacy. Following Definition 5, in order to show Server Privacy we construct an efficient simulator C^* of *client*'s view that is computationally indistinguishable from a real protocol execution. First, C^* computes the first message X from C, using $H(\cdot)$. It computes the remainder of the transcript as follows: It uses knowledge of $S \cap C$ to construct: $\{F(K_i) \mid c_i \in S \cap C\}$. Then, it adds to the set $\{F(r_j)\}$ for $0 < j \leq n - |S \cap C|$, where each r_j is chosen at random. C^* then randomly permutes this set and returns the result as the second message to *server*.

To arrive at a contradiction, suppose that a distinguisher D exists which can differentiate between the real protocol execution: $\text{View}_C(C, S)$ and that of the view simulated by C^*: $C^*(C, C \cap S, n)$. Then, by the hybrid argument, a PPT distinguisher D' must exist that can distinguish between random oracle outputs: $F(K_i)$ and $F(r_j)$ for some j and i, such that $s_i \notin S \cap C$. Thus, by the random oracle model, a simulator for D' can be used to construct an algorithm $\mathcal{A}$ which computes:

$$K_i = X^{\frac{1}{z+hs_i}} = g^{\frac{r(z+hc_1)...(z+hc_m)}{(z+hs_i)}}$$

where $s_i \neq c_k$, for all k. (K_i must be of this form due to boundedness and HbC behavior of *client*.)

Therefore, $(z + hs_i)$ is not a factor of $P_m(z) = r(z + hc_1)...(z + hc_m)$ and does not evenly divide it, with overwhelming probability. There are two possible events that occur with only negligible probability: (1) collisions in $H(\cdot)$, or (2) $P_m(z)/(z + hs_i)$ having a remainder that is a multiple of p'. Thus, we can use $\mathcal{A}$ to solve the *polynomial-generalized one-generator q-SDH* problem on inputs: $[g, g^z, \ldots, g^{(z^t)}]$ and $P_m(z) = r(z + hc_1)...(z + hc_m)$, which is infeasible, based on our assumption. Consequently, by contradiction, Server Privacy holds.

4.3 Computational and Communication Complexity

We now assess communication, computation and storage costs of *b*SH-PSI, as presented in Figure 1.

Communication complexity involves: (1) a single $\log(p)$-bit group element in the first message, and (2) n outputs of $F(\cdot)$ in the second message.

We partition computation costs into *Phase 1* and *Phase 2*. Computation costs are further broken down by specific cryptographic operations: (1) invocations of random

[8] This probability is taken over the input space. Given non-degenerate inputs, these views are perfectly indistinguishable.

oracles: $F(\cdot)$ and $H(\cdot)$ (2) short $\log(p')$-bit multiplications, exponentiations, and inversions, and (3) and long $\log(p)$-bit multiplications and exponentiations. We analyze costs for both *server* and *client*.

Server's *Phase 1* work starts with $O(t)$ mod(p) exponentiations to compute the public key. However, this can be done once for many interactions. It also includes $O(n)$ invocations of $H(\cdot)$, and $O(n)$ mod p' inversions. This requires *server* to know its input set S. If S is stable, this work can also be amortized for many interactions. *Server*'s *Phase 2* work consists of $O(n)$ short $\log(p')$-bit exponentiations and $O(n)$ invocations of $F(\cdot)$.

Client's *Phase 1* work is dominated by the computation of X and m witnesses: $\{X_i | 0 < i \leq \ m\}$. Most work is done in the expansion of the product of binomials of the form $\prod(z + hc_i)$. This can be performed as soon as *client*'s input set is known. Also, as long as p' is fixed globally, *client* does not even need to know which *server* will be involved in the interaction. Coefficients of the resulting reduced polynomial in z can be computed in $O(m^2)$ time using the naïve method of repeated polynomial multiplication. Thus, we can precompute the numerator of X and each X_i in $O(m^3)$ short multiplications. This can be further reduced to $O(m^2\ log^2\ m)$ by taking advantage of a more sophisticated technique (discussed in Section 7) leveraging an $O(d\ log\ d)$ algorithm for d-degree polynomial multiplication.

Also, *client* must perform $O(m)$ invocations of $H(\cdot)$ and $F(\cdot)$, $O(m^2)$ long multiplications and short exponentiations, and $O(m)$ multiplications and exponentiations for each X_i in order to embed the corresponding polynomial evaluated at particular z corresponding to *server*'s public key. In more detail, given $P_t(s) = \sum_{i=0}^{m} a_i s^i$, *client* computes $\prod_{i=0}^{m} {g^{s^i}}^{a_i}$, which is feasible because all g^{s^i} are known.

Client's only mandatory *Phase 2* work amounts to computing a cleartext set intersection, which is achievable with a single sort via $O((m+n)\ log\ (m+n))$ swaps.

Storage overhead is dependent on precomputation. If all possible precomputation is performed, then *server*'s storage is dominated by $O(n)$ $\log(p')$-bit group elements. *Client*'s storage is dominated by $O(m)$ $\log(p)$-bit group elements and $O(m)$ outputs of $F(\cdot)$. If *client* computes Phase 1 without knowledge of *server*'s public key then storage is dominated by $O(m^2)$ short ($\log(p')$-bit) integers.

Optimizations. Choices of public parameters are essential for fast operation. In particular, bSH-PSI can operate in different groups (e.g., on some elliptic curves). We chose integers mod p due to their more efficient operation [11, 10]. Practical current examples of sufficiently secure parameters are: $log(p) \approx\ 1024$ and $log(p') \approx\ 160$.

Furthermore, $H(\cdot)$ substantially influences computational complexity. If the range of H is considerably smaller than p' then $O(m^2\ log^2\ m)$ short ($\log(p')$) multiplications may reduce to $O(m^2\ log^2\ m)$ multiplications of $|H(\cdot)|$-bit integers, and $O(m^2)$ short multiplications (accounting for r).

5 Discussion and Open Problems

5.1 Unlinkability and Change Obliviousness

In settings where *client* and *server* interact more than once, additional privacy properties of *unlinkability* and *change obliviousness* might be desirable for either party.

Informally, *unlinkability* means that, if *client* and *server* interact twice, they should be unable to determine whether they have interacted before. *Change obliviousness* means: if one party's input changes between protocol executions, the other party should not learn this, unless: (1) input size changes, and/or (2) protocol output changes. Unlinkability subsumes change obliviousness; thus, is usually requires more effort.

The proposed *b*SH-PSI protocol provides both unlinkability and change obliviousness for *client*. This is due to *client*'s unconditional privacy. To attain *server* change obliviousness the protocol can be modified to use a keyed random oracle $F'_\gamma(\cdot)$ – instead of $F(\cdot)$ – with a fresh random *server*-selected γ for every interaction. Whereas, to obtain unlinkability, *server* must also generate new secret[9] (z) and public ($[g, g^z, \ldots, g^{(z^t)}]$) keys for every interaction, and communicate the latter to *client*.

These modifications require additional Phase 2 computation and storage for *client* and an extra round of communication. Specifically, γ and one-time public key ($[g, g^z, \ldots, g^{(z^t)}]$) must be communicated to *client* before it can send X. *Client* must now store X_i instead of $F(X_i)$, even if the target *server* is known. If *server* unlinkability is provided, *client* must also store A and A_i and compute X and X_i during Phase 2.

5.2 Flexibility of *t*

At times, it may be desirable for *server* to increase the upper bound t to t'. There are at least two intuitive ways to do so. One way is for *client* and *server* to simply run the protocol $\lceil t'/t \rceil$ times. Alternatively, *server* can publish the extra elements of the public key: $[g^{z^{t+1}}, \ldots, g^{z^{t'}}]$. Either approach provides forward security for both parties. That is, no additional information can be learned from prior protocol executions, with lower bounds. Note, however, that t cannot be decreased unless an entirely new public key is generated.

5.3 Interacting with Multiple Servers

Optimizations can be made to save *client*'s resources in settings where *client* intends to interact with multiple servers using the same input set. First, if *server*'s set is not known ahead of time, or if space is a concern, *client* can compute and store A and A_i instead of X and t'_i. Of course, this is only possible if all servers use the same public key parameters: (g, p, p').

5.4 Malicious Security

While our protocol is secure in the HbC* model, it provides unconditional client privacy regardless of the behavior of *server*[10]. Security against a fully malicious server [18] would require a proof of valid computation of the random oracle $F(\cdot)$ without revealing the oracle's input. Security against a malicious *client* would require a proof that

[9] Strictly speaking, a new z is not needed. Instead, *server* can generate a new base $\hat{g}$, compute the new $[\hat{g}, \hat{g}^z, \ldots, \hat{g}^{(z^t)}]$ and keep the same z.

[10] *Client* need only verify g^A is a generator by computing $(g^A)^{p'}$ before exponentiating with r.

the accumulator $X = g^u$ is well-formed, for some u. We believe that such a proof is challenging since the exponent u is not known to *client*. Moreover, it is unclear how to construct a proof without revealing *client*'s input size in the process. An alternative approach is to rely on a variant of the Exponent Strong q-SDH assumption which states that: computing $(c, x^{\frac{1}{z+c}})$ is hard for all $x \in \mathbb{Z}_p$ given $[g, g^z, \ldots, g^{(z^q)}]$.

5.5 Group Selection

Due to its computational efficiency of operations, we chose prime-order integer DH-groups. This efficiency is largely based on the fact that exponentiation can take advantage of the relatively small size of p'. Our protocol would work equally well in other DH-groups, such as the elliptic curve DH-group variant [5]. However, in our experience, these groups tend to be slower using existing implementations. Since computational cost (and not storage) is of primary importance, integer groups are the logical choice. We also conjecture that variants of the protocol composite groups (e.g., in the RSA setting) are easily realizable.

5.6 t-Intersection bSH-PSI

Thus far, we focused on limiting the amount of information revealed to *client* in each interaction by providing a guaranteed upper bound on *client*'s input size. An alternative approach would be limit the size of the intersection $|C \cap S|$. Although not secure against enumeration by *client*, this approach is useful in some situations. It is particularly applicable if *server*'s input set is much larger than t and the domain of set elements is large. For example, suppose that *server* owns a database and is willing to answer any query with a result set less than t. A hypothetical t-intersection bSH-PSI protocol could be realized in at least two variations (each of independent interest): (1) if $|C \cap S| > t$, *client* learns nothing, or (2) if $|C \cap S| > t$, *client* learns a random t sized subset of the intersection $C \cap S$. We defer the investigation of this topic for future work.

6 Equivalence of SDH Problems

We now show equivalence of the two new assumptions and their more established counterparts. First, we argue that *polynomial-generalized one-generator q-SDH* and *one-generator q-SDH* are equivalent. Next, we show equivalence of *polynomial-generalized exponent q-SDH* and *exponent q-SDH* . Both equivalence proofs describe two reductions (one in each direction) between respective problems.

Theorem 1. *The one-generator (q, t, ϵ)-SDH assumption holds iff the polynomial- generalized one-generator (q, t, ϵ)-SDH assumption holds.*

Proof. We show the contrapositive in each case. First, suppose that there exists an algorithm:

$$\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}]\big) \rightarrow (c, g^{\frac{1}{(z+c)}})$$

that has an non-negligible advantage ϵ in solving *one-generator q-SDH* . We can then construct an algorithm:

$$\mathcal{A}'\big([g, g^z, \ldots, g^{(z^q)}], P_n(z))\big) \rightarrow (c, g^{\frac{P_n(z)}{(z+c)}})$$

that has the same advantage in solving the *polynomial-generalized one-generator q-SDH* problem. First, $\mathcal{A}'$ runs $\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}]\big)$. With probability at least ϵ, $\mathcal{A}$ outputs:

$$[c, g^{\frac{1}{z+c}}] \tag{1}$$

for some $c \in \mathbb{Z}^*_{p'}$. We observe that $\mathcal{A}'$ may use the polynomial division algorithm to rewrite the non-trivial part of its desired output as:

$$g^{\frac{P_n(z)}{(z+c)}} = g^{\frac{P'_n(z)+r}{z+c}} = g^{\frac{P'_n(z)}{z+c}} g^{\frac{r}{z+c}},$$

where $P'_n(z)$ is a polynomial divisible by $(z+c)$, and r is a constant in $\mathbb{Z}^*_{p'}$. Because $(z+c)$ divides $P'_n(z)$,

$$g^{\frac{P'_n(z)}{z+c}} = g^{P''_{n-1}(z)}, \tag{2}$$

where $P''_{n-1}(z)$ is a polynomial in z of degree $n-1$. Because $(n-1) < q$, $\mathcal{A}'$ may compute (2) by exponentiating and multiplying together elements from $[g, g^z, \ldots, g^{(z^q)}]$. Using (1), $\mathcal{A}'$ computes

$$(g^{\frac{1}{z+c}})^r = g^{\frac{r}{z+c}}. \tag{3}$$

Finally, $\mathcal{A}'$ multiplies (2) by (3) to obtain the value $g^{\frac{P_n(z)}{(z+c)}} = g^{\frac{P_n(z)}{z+c}}$, which is then output with the known value c. If and only if $\mathcal{A}$'s output is correct, $\mathcal{A}'$ also outputs a correct solution. Therefore, $\mathcal{A}'$ has advantage equal to ϵ in solving the *polynomial-generalized exponent q-SDH* problem.

Now, conversely, suppose that there exists an algorithm:

$$\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}], P_n(z)\big) \rightarrow (c, g^{\frac{P_n(z)}{(z+c)}})$$

that has an advantage ϵ in solving the *polynomial-generalized one-generator q-SDH* problem. Then, we can construct an algorithm:

$$\mathcal{A}'\big([g, g^z, \ldots, g^{(z^q)}]\big) \rightarrow (c, g^{\frac{1}{(z+c)}})$$

that has an advantage ϵ in solving the *one-generator q-SDH* problem with probability at least ϵ by merely running $\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}], 1\big)$ and outputting the result. If $\mathcal{A}$ yields a correct output $[c, g^{\frac{1}{(z+c)}}]$, then $\mathcal{A}'$ is also correct. Thus $\mathcal{A}'$ has advantage equal to ϵ of solving the *one-generator q-SDH* problem.

Theorem 2. *The exponent (q, t, ϵ)-SDH assumption holds iff the polynomial-generalized exponent (q, t, ϵ)-SDH assumption holds.*

Proof. We show the contrapositive for both cases. Suppose there exists an algorithm:

$$\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}]\big) \rightarrow g^{(z^{q+1})}$$

that has an advantage ϵ in solving the *exponent q-SDH* problem. We then construct another algorithm:

$$\mathcal{A}'\big([g, g^z, \ldots, g^{(z^q)}], P_n(z)\big) \rightarrow g^{(P_n(z))}$$

that has an advantage of $(\epsilon)^{poly(n)}$ in solving the *polynomial-generalized exponent q-SDH* problem. (Note that $(\epsilon)^{poly(n)}$ is non-negligible if ϵ is non-negligible). $\mathcal{A}'$ creates an $(n+1)$-tuple of the form $[g, g^z, \ldots, g^{(z^n)}]$ as follows: for each $q < j \leq n$, it runs $\mathcal{A}\big([g, g^z, \ldots, g^{(z^{j-1})}]\big)$ to obtain g^{z^j} and saves it for subsequent calls to $\mathcal{A}$. If any call fails to produce the correct output, $\mathcal{A}'$'s output will also be incorrect. We observe that $P_n(z) = a_0 + a_1 z + \ldots + a_n z^n$. Thus,

$$g^{P_n(z)} = g^{a_0} g^{a_1 z} \ldots g^{a_n z^n}.$$

Since all coefficients a_i and values $([g, g^z, \ldots, g^{(z^n)}])$ are now known to $\mathcal{A}'$, it outputs $g^{P_n(z)}$. Thus, $\mathcal{A}'$ has non-negligible advantage ϵ^{n-q} in solving the *polynomial-generalized exponent q-SDH* problem.

Now suppose there exists an algorithm:

$$\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}], P_n(z)\big) \rightarrow g^{(P_n(z))}$$

that has a non-negligible advantage ϵ in solving *polynomial-generalized exponent q-SDH* . We construct another algorithm:

$$\mathcal{A}'\big([g, g^z, \ldots, g^{(z^q)}]\big) \rightarrow g^{(z^{q+1})}$$

that has the same advantage ϵ in solving *exponent q-SDH* by simply running and outputting $\mathcal{A}\big([g, g^z, \ldots, g^{(z^q)}], z^{q+1}\big)$. This call to $\mathcal{A}$ has probability at least ϵ of outputting $g^{z^{q+1}}$, and solving the *exponent q-SDH* problem.

7 Client Complexity to Compute Accumulator Embeddings

As discussed in Section 4.3, the dominating factor in *client* computation is the construction of $m+1$ accumulators: X and each X_i. Recall that $X = g^{\Pi_{i=0}^{m}(z+hc_i)}$ and $X_i = g^{\frac{1}{hc_i}\Pi_{j=0}^{m}(z+hc_j)}$. Intuitively, X is an embedding of all of *client*'s hashed set elements $\{hc_1, \ldots, hc_m\}$ into an accumulator, and X_i is the same embedding, with one hc_i missing. *Client* computes these accumulators in two steps: (1) reduces the polynomial in the exponent, and (2) embeds the coefficients of the reduced polynomial into the group using *server*'s public key: $[g, g^z, \ldots, g^{(z^t)}]$.

Step (1) can be computed in several ways described in the remainder of this section. However, each X_i only differs from X by one binomial in the exponent. Thus it might be possibly to take advantage of this shared structure to compute all X_i-s simultaneously. Step (2) can be done with m modular exponentiations and $m-1$ modular

multiplications. Despite the shared structure among X_i-s, each reduced polynomial has unique coefficients. Therefore, the total cost for the second step is $O(m^2)$.

We now describe two strategies for reducing the polynomial in the exponent: the *direct* strategy and the *tree* strategy. Complexity of each strategy depends on the underlying algorithm used for polynomial multiplication. We analyze performance of each strategy using both naïve $O(d^2)$ and FFT- based $O(d \log(d))$ algorithms for d-degree polynomial multiplication. *Direct* strategy entails multiplying each binomial in the product, iteratively with an accumulator (initialized to 1). This requires $m-1$ polynomial multiplications.

Tree strategy is best described by visualizing a binary tree where each binomial term (in the product) is a leaf. Each internal tree node is the product of its children. This way, the reduced polynomial is represented by the root. We construct this tree by starting at the leaves $(s + hc_i)$ and computing internal nodes one level at a time. This can be done with $O(m)$ polynomial multiplications.

Both strategies could potentially benefit from memoization, when computing X and each X_i simultaneously due to the shared structure between them. However, it is easier to exploit this structure with *tree* strategy. We memoize nodes in the tree while computing X. Since the trees corresponding to each pair $[X_i, X_j]$ $(i \neq j)$ only differ by one leaf node, we can obtain them by simply re-computing the co-path from the missing node to the root. This can be done with only $O(\log(m))$ polynomial multiplications. However, the degree of the polynomial increases as we get closer to the root. Therefore, the cost of each multiplication increases as we traverse the tree.

Direct strategy requires $O(m)$ polynomial multiplications. At each iteration, we multiply a polynomial (of degree equal to the number of iterations) by a degree-one binomial. Each such polynomial multiplication can be performed with exactly m regular modular multiplications. The total cost of *direct* strategy using a naïve algorithm for matrix multiplication is:

$$\sum_{i=1}^{m} i = \frac{m(m+1)}{2}.$$

Thus, the asymptotic cost of computing a single embedding is $O(m^2)$, and the cost of all embeddings is $O(m^3)$.

If we use a generic $O(d \log(d))$ algorithm for polynomial multiplication, the cost is

$$\sum_{i=1}^{m} i \log(i) \leq \sum_{i=1}^{m} m \log(m) = m^2 \log(m),$$

which results in a slightly worse total running time of $O(m^3 \log(m))$.

Switching to *tree* strategy, we count the number of multiplications needed for computing all nodes in one tree. The height of the tree is $\log(m)$. We assume that levels in the tree are indexed, starting at the leaves. Thus, the degree of the polynomial stored in a node at level i is 2^i. Also, there are $2^{\log(m)-i}$ nodes at each level. Therefore, the total cost to compute a tree using a $O(d^2)$ algorithm for polynomial multiplication is:

$$\sum_{i=1}^{\log(m)} (2^i)^2 (2^{\log(m)-i}) = \sum_{i=1}^{\log(m)} (2^i)^2 (\frac{m}{2^i}) = \sum_{i=1}^{\log(m)} (2^i)(m) = m \sum_{i=1}^{\log(m)} (2^i) =$$

$$= m(2^{\log(m)+1} - 1) = m(m+1).$$

The overall cost of computing all embeddings is $O(m^3)$, which is equivalent to *direct* approach. However, some efficiency gain can be obtained with *tree* strategy using a $O(d\ log\ d)$ algorithm for polynomial multiplication. Then, the cost of computing a single tree can be written as:

$$\sum_{i=1}^{\log(m)} (2^i\ log\ 2^i)(2^{\log(m)-i}) = \sum_{i=1}^{\log(m)} (2^i\ log\ 2^i)(m/2^i) = \sum_{i=1}^{\log(m)} (m\ log\ 2^i),$$

which is bounded above by:

$$\sum_{i=1}^{\log(m)} (m\ log\ 2^{\log(m)}) = \sum_{i=1}^{\log(m)} (m \log(m)) = m \log^2 m.$$

Thus, the cost to compute a single accumulator is $O(m\ log^2\ m)$ and the cost to compute m accumulators is $O(m^2 \log^2\ m)$.

8 Conclusions

Motivated by recent advances in size-hiding secure computation and, more specifically, SH-PSI: size-hiding private set intersection techniques, this paper investigated bounded variants thereof. The main contribution of this work is the construction of the first *b*SH-PSI technique that allows *client* to unconditionally hide its input size while allowing *server* to limit that size. We believe that *b*SH-PSI can be a useful tool in the arsenal of secure computation techniques. There are at least three directions for future work: (1) alternative and/or more efficient, *b*SH-PSI techniques, (2) other private set operations with bounded (one-sided) size-hiding input, e.g., private set union and private set intersection cardinality, and (3) modifications of our current construct and its proofs to provide security against malicious client in the standard model, i.e., without relying on random oracles.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments. We also thank Jaroslav Šeděnka for his contributions to the initial stages of this work.

References

1. G. Ateniese, E. D. Cristofaro, and G. Tsudik. (If) size matters: Size-hiding private set intersection. In *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 156–173. Springer, 2011.
2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

3. D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2004.
4. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.
5. W. J. Caelli, E. P. Dawson, and S. A. Rea. Pki, elliptic curve cryptography, and digital signatures. *Computers & Security*, 18(1):47–66, 1999.
6. M. Chase, R. Ostrovsky, and I. Visconti. Executable proofs, input-size hiding secure computation and a new ideal world. In *EUROCRYPT*, volume 9057 of *Lecture Notes in Computer Science*, pages 532–560. Springer, 2015.
7. D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient robust private set intersection. *International Journal of Applied Cryptography*, 2(4):289–303, 2012.
8. 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*, volume 7374 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2012.
9. P. D'Arco, M. I. G. Vasco, A. L. P. del Pozo, and C. Soriente. Size-hiding in private set intersection: what can be done and how to do it without random oracles. *IACR Cryptology ePrint Archive*, 2015:321, 2015.
10. E. De Cristofaro, S. Faber, P. Gasti, and G. Tsudik. Genodroid: are privacy-preserving genomic tests ready for prime time? In *WPES*, pages 97–108. ACM, 2012.
11. E. De Cristofaro, S. Faber, and G. Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In *WPES*, pages 107–118. ACM, 2013.
12. E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, pages 218–231, 2012.
13. E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology-ASIACRYPT 2010*, pages 213–231. Springer, 2010.
14. E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security*, pages 143–159. Springer, 2010.
15. C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800. ACM, 2013.
16. S. Faber, R. Petrlic, and G. Tsudik. Unlinked: Private proximity-based off-line osn interaction. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 121–131. ACM, 2015.
17. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004.
18. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
19. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.
20. V. Goyal, R. Ostrovsky, A. Scafuro, and I. Visconti. Black-box non-black-box zero knowledge. In *STOC*, pages 515–524. ACM, 2014.
21. C. Hahn and J. Hur. Scalable and secure private set intersection for big data. In *2016 International Conference on Big Data and Smart Computing, BigComp 2016, Hong Kong, China, January 18-20, 2016*, pages 285–288, 2016.
22. C. Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In *TCC (2)*, volume 9015 of *Lecture Notes in Computer Science*, pages 90–120. Springer, 2015.

23. C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography*, pages 155–175. Springer, 2008.
24. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
25. Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.
26. F. Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 85–86. ACM, 2012.
27. L. Kissner and D. Song. Private and threshold set-intersection. Technical report, DTIC Document, 2004.
28. Y. Lindell, K. Nissim, and C. Orlandi. Hiding the input-size in secure two-party computation. *IACR Cryptology ePrint Archive*, 2012:679, 2012.
29. Y. Lindell, K. Nissim, and C. Orlandi. Hiding the input-size in secure two-party computation. In *ASIACRYPT*, volume 8270 of *Lecture Notes in Computer Science*, pages 421–440. Springer, 2013.
30. S. Micali, M. O. Rabin, and J. Kilian. Zero-knowledge sets. In *FOCS*, pages 80–91. IEEE Computer Society, 2003.
31. L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.
32. B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 797–812, 2014.
33. N. Tanaka and T. Saito. On the q-strong diffie-hellman problem. *IACR Cryptology ePrint Archive*, 2010:215, 2010.