Unbalanced Private Set Intersection from Homomorphic Encryption and Nested Cuckoo Hashing

Kußmaul, Jörn
`joern.kussmaul@gmail.com`

Akram, Matthew
`matthew.akram.zaher.farid@sap.com`

Tueno, Anselme
`anselme.tueno@sap.com`

Abstract

Private Set Intersection (PSI) is a well-studied secure two-party computation problem in which a client and a server want to compute the intersection of their input sets without revealing additional information to the other party. With this work, we present nested Cuckoo hashing, a novel hashing approach that can be combined with additively homomorphic encryption (AHE) to construct an efficient PSI protocol for unbalanced input sets. We formally prove the security of our protocol against semi-honest adversaries in the standard model. Our protocol yields client computation and communication complexity that is sublinear in the server's set size and is thus of interest to clients with limited resources. The implementation and empirical evaluation of our protocol using the exponential ElGamal and BGV/BFV encryption schemes attests to state-of-the-art practical performance.

1 Introduction

Due to the rise of the world-wide-web, client-server protocols are omnipresent nowadays, and privacy requirements for legal and personal reasons are becoming more demanding. Currently, in practice, data is only encrypted during transmissions or storage but not when processed. As such, a server that performs computations with encrypted data first decrypts the data and thus must possess the secret key. Even if the server is assumed to behave honestly, security vulnerabilities might allow attackers to observe the processed data or fully control the server's computations. Over the last decades, techniques have been developed to process data securely without leaking sensitive information to the processing server.

The invention of garbled circuits by Yao [90] and the *Goldreich-Micali-Wigderson* (GMW) protocol [44] have laid the cornerstones for the computation on encrypted data (COED) field and are called generic *secure multi-party computation* (MPC) techniques. Both protocols can securely evaluate any fixed (boolean, respectively arithmetic) circuit and, as such, any computable functionality on the parties' inputs. *Fully homomorphic encryption* (FHE) is another cryptographic solution that allows COED and is usually considered the *holy grail* of cryptography. While many asymmetric cryptosystems like *ElGamal* [33] or *Rivest–Shamir–Adleman* (RSA) [84] offer some homomorphism (e.g., multiplications on ciphertexts), Gentry [40] invented an FHE scheme that allows (arbitrarily many) multiplications and additions and as such, any computable functionality on the parties' inputs. For many functionalities, e.g., with large inputs, a secure computation using a generic MPC or FHE circuit is often less efficient than custom protocols [77]. Hence, for particular functions with great practical interest, special-purpose protocols have been developed, such as for *private set intersection* (PSI), but also *private information retrieval* (PIR) [3; 5; 7; 24; 62; 67], Sealed-Bid Auctions [6] and more.

In this work, we will consider special-purpose protocols for the PSI problem. PSI is a secure two-party computation (2PC) problem where two parties want to learn which items they have in common without leaking anything else to the other party. PSI is a highly researched 2PC problem with a large and growing number of applications. Some of the most popular applications include *Private Contact Discovery* [46; 65; 89], *Advertisement Conversion Rate* [51; 52] and *Password Breaching Alerts* [87].

1.1 Related Work

We differentiate between PSI protocols that use interactive masking, unbalanced PSI protocols, and constructions in other security models or with different functionalities.

Masking Elements Most PSI protocols involve interactive masking of the input elements and a (local) intersection calculation on the masked elements. In practice, especially for contact discovery, mostly *naïve hashing* approaches are used, where (salted) hashes of the items are sent to the other party [46]. These approaches, however, are insecure and, e.g., offer no forward-secrecy and are vulnerable to dictionary attacks [46; 78; 89]. The first semi-honest secure PSI protocol by

Meadows [66] (later revisited by Huberman et al. [49]), called DH-PSI, uses adapted *Diffie-Hellman* (DH) key exchanges and random oracles to mask the input elements such that two identical elements get the same mask. The masked elements are then sent to the other party, which computes the intersection. De Cristofaro and Tsudlik [29] later presented a similar protocol based on blind RSA signatures, where Rosulek and Trieu [85] ported DH-PSI to the malicious adversary setting. The (computationally) fastest PSI protocols replace many computationally expensive calculations like exponentiations with symmetric key operations by, e.g., using *oblivious transfer* (OT) extensions [53; 77]. The efficiency is improved by using customized hashing schemes to securely compare each receiver element with a subset of the sender's set [77]. State-of-the-Art PSI protocols utilize Cuckoo hashing [77], permutation-based Cuckoo hashing [74; 78], and the so-called *sparse oblivious transfer* (SpOT) structure [72] to further enhance the practical communication and computation costs. In theory, instead of directly using OT to check set membership securely, Kolesnikov et al. [60] have constructed PSI protocols based on *oblivious pseudo-random function* (OPRF), and later Garimella et al. [38] abstracted the hashing and masking steps by introducing Oblivious Key-Value Stores. Novel approaches use state-of-the-art PSI hashing structures with *Vector Oblivious Linear Evaluation* to build the fastest PSI protocols [19; 57; 83] that also provide the lowest communication costs for many input sizes. However, all mentioned PSI protocols in this category yield a sender and receiver's communication and computation complexity which is at least linear in the larger set size. In practice, e.g., in a mobile contact discovery application, the mobile client must download and process gigabytes of data for large server databases (2^{30} server items). Even though the processing is very fast due to symmetric primitives and native CPU instructions (e.g., AES-NI [45]), a large amount of data can lead to poor running times and also high monetary costs (e.g., for mobile internet).

Protocols for Unbalanced Sets Several protocols work under the assumption that the number of elements in the input sets are unbalanced. This assumptions can be used to decrease the computation or communication costs of the client (i.e., the party with fewer items). Kiss et al. [59] present a framework for unbalanced PSI that is especially suitable for the application of private mobile discovery. Resende & Aranha [81; 82] have improved the performance in Kiss et al.'s framework using customized filter techniques and fast elliptic curve multiplications [1]. However, protocols in Kiss' framework generally suffer from false positives, high communication costs, or a large client state. Falk et al. [34] port the fast OPRF-based of Kolesnikov et al. to the unbalanced setting, which, however, still requires linear communication complexity in the larger set size.

Homomorphic encryption provides a secure way to evaluate polynomials to check if an input element is part of another party's input set. However, a naïve approach would require either precomputing many encrypted polynomials on the client side (including high communication costs) or computing many computationally inefficient homomorphic multiplications on the server. To reduce the complexity, Freedman et al. [36; 37] make use of hashing (later extended by Pinkas et al. [77]). However, for N server items, the protocol of Freedman et al. still requires computing and sending $O(N)$ encrypted messages. Chen et al. [22; 23] use (leveled) FHE to efficiently compute the polynomials and significantly reduce the client's communication and computation complexity. However, their protocol is only practical for set items with small bit-lengths ($\approx$32-bit) and requires complex adjustment of security-critical parameters. Cong et al. [27] later improved this work to support larger item bit-lengths. Novel PSI constructions [4; 8] in the laconic cryptography framework [80] offer asymptotically optimal communication. However, these protocols rely on heavy computations and are impractical compared to non-laconic protocols.

Different Security Models and Functionalities Besides the standard 2PC model with exactly two parties, many PSI protocols in other models have been presented. We only focus on PSI protocols with two parties' inputs but refer to the literature for so-called multi-party PSI protocols [12; 61]. Kerschbaum's protocol [56] uses a trusted third party (TTP) to achieve malicious security and to outsource the computation to another party without inputs. Demmler et al. [30] modify current PIR constructions (based on function secret sharing (FSS) [16]) and add additional hashing schemes to build a fast PSI protocol with low communication costs. However, the modified PIR scheme requires that the server's input is shared in a two non-colluding server model. Our protocol can be framed as an improved single-server variant of PIR-PSI.

Besides the PSI protocols that rely on other security models, some protocols do not output the intersection itself but, depending on the application, e.g., the number of elements in the intersection. The already mentioned protocol for computation of *advertisement conversion rates* [51] calculates and outputs the sum of associated payloads. Other protocols [75; 76] combine a PSI hashing scheme with generic MPC circuits to allow arbitrary computations on the set intersection. Ciampi and Orlandi [25; 64] present schemes for secure computation on intersections compatible with different MPC and FHE techniques. Janneck et al. [54] combine this idea with the low communication protocol of Cong et al. [27] to any functionality on the intersections with one round of communication. However, protocols for arbitrary computations on the intersection yield worse computational performance compared to other PSI protocols.

1.2 Contributions

In this work, we construct a client-server hashing scheme that might be of independent interest, e.g., for multiserver PIR-PSI or multi-message keyword PIR [88]. Based on our hashing scheme, we propose a novel generic unbalanced PSI protocol that can be instantiated with any additively homomorphic encryption scheme, has a low, one-round communication, and offers various communication-computation complexity tradeoffs. We present improved protocol instantiations based on the additively homomorphic *exponential ElGamal* [28] scheme and based on the *Brakerski/Fan-Vercauteren* (BFV) [17; 35] and *Brakerski-Gentry-Vaikuntanathan* (BGV) [18] (leveled) fully homomorphic encryption schemes (in Subsection 3.5). Our concrete protocol instantiations are constructed for an unbalanced PSI scenario and have some similarities with Chen et al. [23] combined with ideas from PIR-PSI [30]. We describe several protocol extensions, including efficient server updates and secure set-size computations. Since our protocol offers a sublinear client computation and communication (in the server size), it is especially suitable for clients with fewer computational resources (like mobile clients). We have implemented our schemes using state-of-the-art secure computation libraries. Our evaluations attest to practical performance, including low communication overhead. The description of our implementation and the evaluation is presented in Section 5.

1.3 Outline

In Section 2, we introduce the preliminaries of our work. Section 3 shows our unbalanced PSI protocol constructions. The security and complexity analyses are presented in Section 4. Section 5 shows empirical evaluations of our implementations.

2 Preliminaries

We assume knowledge of basic mathematical structures (e.g., groups), basic probability theory, complexity theory (e.g., *probabilistic polynomial time* (PPT) algorithms), and foundations of cryptography which include security definitions for set and ciphertext indistinguishability like *Indistinguishability under chosen-plaintext attack* (IND-CPA), and concrete security assumptions like *Decisional Diffie-Hellman* (DDH) and lattice-based *learning with errors* (LWE). Most of what we assume can be found in the book of Katz and Lindell [55] or Yang et al.'s tutorial [63] (for lattice-based matters).

2.1 Notations and Terminology

With *items* or *elements*, we refer to ρ-bit strings, also interpreted as ρ-bit unsigned integers (or boolean values if $\rho = 1$. The bit-wise exclusive or (XOR) is indicated by $\oplus$, and the bit-wise negation by $\neg$. With $v \leftarrow l$, we denote the assignment of the value of l to variable v. If the value of a variable v is uniformly at random from a set S, we write $v \leftarrow_\$ S$. For an array A of size n and $i \in \{1, \dots, n\}$, $A[i]$ denotes the ith entry in A. Depending on the context, we interpret an array of length k as a k-dimensional vector or $(k \times l)$-dimensional matrix if the array entries are arrays of length l. With $A^\top$, we denote the transposition of a vector or matrix A, and with $\langle , \rangle$, we denote the dot product. The ring of integers modulo $N \in \mathbb{N}$ is denoted as $\mathbb{Z}_N$. With $\mathbb{Z}_N^n$, we denote the n-dimensional cartesian product of $\mathbb{Z}_N$ with component-wise addition and multiplication modulo N. The multiplicative subgroup of $\mathbb{Z}_N$ is denoted as $\mathbb{Z}_N^\times$. We write $a + b := (a + b \mod N)$ for calculations over $\mathbb{Z}_N^n$ (or $\mathbb{Z}_N$) and omit $(\mod N)$ in our notation. For any $i \in \mathbb{N}$, H_i denotes a *universal hash function* (as defined in subsubsection 2.4.1).

For any two $j, k \in \mathbb{N}$ with $j \neq k$, we assume that $H_j \neq H_k$. $CT_{H_1,\dots,H_k}$ denotes a Cuckoo hash table (as defined in subsubsection 2.4.2) corresponding to the hash functions H_1, ..., H_k. We omit H_1, ..., H_k and writxe CT if the concrete hash functions are apparent by the context. With M, we refer to a finite set of elements and write $CT_{H_1,\dots,H_k}(M)$ for a Cuckoo table filled with the elements in M but also omit M if the set is clear from the context.

In our secure two-party computation scenario, the party that receives output is called the *receiver*, and the other party is called the *sender*. In the unbalanced PSI case, the receiver is called the *client* (marked as C), and the sender is called the *server* (marked as S). Each party has a finite set of elements where Y indicates the client's set and X indicates the server's set.

2.2 Secure Two-Party Computation

Nowadays, in cryptography, methods are offered and researched that enable privacy-preserving (and correctness-preserving) computations between two parties, so-called *secure two-party computation* (2PC). For 2PC, we will only consider semi-honest adversaries with static corruptions in the standalone execution model [48]. Since PSI protocols have a deterministic functionality, we later use separate semi-honest security requirements for correctness, client privacy, and server privacy, as defined by Hazay and Lindell [48].

2.2.1 Private Set Intersection

PSI protocols are modeled as a 2PC for the intersection of two sets that belong to different parties. In the PSI literature, many different functionalities related to PSI have been proposed (e.g., [27; 51; 69]). Some differ only in formal details, while others consider certain output variations. The common *asymmetric* PSI problem we focus on in this work requires that only the client learns the intersection which is shown in Figure 1.

In general, PSI protocols always leak information about

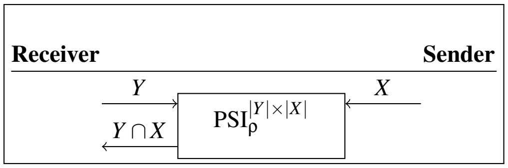


Figure 1: PSI functionality for set sizes $|X|$, $|Y|$ and items of bit-length ρ.

the size of at least one party's set. To formally overcome this problem, PSI protocols assume the client's and server's set sizes are publicly known system parameters (or output to both parties). However, in practice, one could obfuscate the exact set sizes, e.g., by adding a random amount of dummy elements to the input sets. Our proposed constructions are not restricted to certain set size parameter combinations, although they are optimized for $|Y| \ll |X|$.

2.2.2 Private Information Retrieval

Private information retrieval (PIR) is another famous problem in the COED field, where a client wants to retrieve a database entry at an index i from a server. In this scenario, the server shall not learn which database entry has been queried by the client. The database is assumed to be public (in contrast to PSI). The restriction that the communication shall be sublinear in the size of the database prohibits the trivial solution of just sending the whole database as plaintext to the client. Protocols that also hide the database entries are called *secure* PIR protocols. Novel PIR protocols show running time improvements compared to the trivial (no-)solution [5]. PIR protocols can be divided into two classes, protocols that replicate the database among multiple non-colluding servers [16; 20; 24] and non-replicating protocols that make use of homomorphic encryption (HE) [3; 7; 62; 67]. The PIR-PSI [30] protocol is based on the former class of non-colluding server PIR, while our protocol uses some constructions similar to HE-based PIR protocols. For an overview of current PIR schemes and formal definitions, we refer to Ali et al. [5].

2.3 Homomorphic Encryption

We define a public key encryption (PKE) scheme as 3-tuple of PPT algorithms as by Katz and Lindell [55]. If the specific keys are irrelevant or clear from the context, the keys will be omitted in the notation (e.g., $\mathrm{Enc}(\mathsf{m})$ instead of $\mathrm{Enc}(\mathsf{k}_{\mathsf{pk}},\mathsf{m})$). Secure PKE schemes require randomly drawn numbers in the encryption step. For a PPT PKE encryption $\mathrm{Enc}(\mathsf{m})$, $\mathrm{Enc}(\mathsf{m};r)$ is a deterministic algorithm such that $\mathrm{Enc}(\mathsf{m}) = \mathrm{Enc}(\mathsf{m};r)$, if r is used as randomness in the encryption step of $\mathrm{Enc}(\mathsf{m})$.

We consider homomorphic encryption (HE) as PKE schemes that additionally allow operating on ciphertexts using a PPT algorithm EvalHom that outputs a ciphertext $\mathsf{c}' \in \mathcal{C}$, given the public key k_{pk} and two ciphertexts $\mathsf{c}_1, \mathsf{c}_2 \in \mathcal{C}$ as inputs. For any $\mathrm{Dec}(\mathsf{c}_1) = \mathsf{m}_1$ and $\mathrm{Dec}(\mathsf{c}_2) = \mathsf{m}_2$, we assume that we can define a group $(\mathcal{M}, \otimes)$ with $\mathrm{Dec}(\mathrm{EvalHom}(\mathsf{k}_{\mathsf{pk}},\mathsf{c}_1,\mathsf{c}_2)) = \mathsf{m}_1 \otimes \mathsf{m}_2$. Remark, some HE definitions use, instead of the public key, a third *evaluation key* as input to EvalHom. For formal definitions of homomorphic encryption, we refer to Katz and Lindell [55] and Li et al. [63].

2.3.1 Additively Homomorphic Encryption

The homomorphic property of HE schemes assumes the existence of any group over the plaintext space with a corresponding ciphertext evaluation algorithm. For COED, *additively homomorphic encryption* (AHE) schemes where plaintext groups are of type $(\mathbb{Z}_N, +)$ are of particular interest.

We will write $\mathsf{c}_1 \boxplus \mathsf{c}_2$ instead of $\mathrm{EvalHom}(\mathsf{k}_{\mathsf{pk}},\mathsf{c}_1,\mathsf{c}_2)$ for AHE schemes and use $\mathsf{c} \boxplus \mathsf{m}$ as shorthand notation for $\mathsf{c} \boxplus \mathrm{Enc}(\mathsf{m})$. Remark that with ciphertexts additions, ciphertexts $\mathsf{c} \in \mathcal{C}$ can be multiplied with scalars $s \in \mathbb{Z}_N$ by an s-fold homomorphic addition of c with c. The complexity of the ciphertext-plaintext multiplication can be reduced to be polynomial in the bit-length of s by reusing the subtotals. We assume the ciphertext scalar multiplication to be a PPT algorithm and denote it as $\mathsf{c} \boxdot s$. The subtraction of two ciphertexts $\mathsf{c}_1, \mathsf{c}_2 \in \mathcal{C}$ is defined and written as $\mathsf{c}_1 \boxminus \mathsf{c}_2 := \mathsf{c}_1 \boxplus (\mathsf{c}_2 \boxdot (-1 \mod N))$. Likewise, for a ciphertext $\mathsf{c} \in \mathcal{C}$ and a message $\mathsf{m} \in \mathcal{M}$, we define $\mathsf{c} \boxminus \mathsf{m} := \mathsf{c} \boxminus \mathrm{Enc}(\mathsf{m})$ and $\mathsf{m} \boxminus \mathsf{c} := \mathrm{Enc}(\mathsf{m}) \boxminus \mathsf{c}$. Given an AHE scheme and a ciphertext $\mathsf{c} \leftarrow \mathrm{Enc}(b_1)$ with $b_1 \in \{0,1\}$, we can interpret b_1 as a boolean value and calculate the negation of b_1 on the encrypted ciphertext as $\neg\mathsf{c} := \mathrm{Enc}(1) \boxminus \mathsf{c}$. With the ciphertext negation, we define the XOR of $\mathsf{c} \leftarrow \mathrm{Enc}(b_1)$ and a plain bit $b_2 \in \{0,1\}$ as $\mathsf{c} \oplus 1 := 1 \boxminus \mathsf{c}$ and $\mathsf{c} \oplus 0 := 0 \boxplus \mathsf{c}$.

2.3.2 (Exponential) ElGamal Encryption

The ElGamal encryption scheme is an IND-CPA PKE scheme secure under the DDH assumption.

Definition 2.1 (ElGamal encryption)**.** Let $\mathcal{G}$ be a PPT algorithm that given 1^κ, for a security parameter $\kappa \in \mathbb{N}$, returns a group $\mathbb{G}$ with prime order p, generator g and group operation $\odot$. For $\kappa \in \mathbb{N}$, group $(\mathbb{G}, p, g, \odot) \leftarrow \mathcal{G}(1^\lambda)$, message space $\mathbb{G}$, key space $\mathbb{Z}_p \times \mathbb{G}$ and a ciphertext space $\mathbb{G} \times \mathbb{G}$, the ElGamal encryption scheme is a 3-tuple $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ of the following PPT algorithms:

- $\mathrm{Gen}(1^\kappa)$: Given a security parameter $\kappa \in \mathbb{N}$, $\mathrm{Gen}(1^\kappa)$ randomly draws $\mathsf{k}_{\mathsf{sk}} \leftarrow_{\$} \mathbb{Z}_p$ and outputs $\left(\mathsf{k}_{\mathsf{sk}}, g^{\mathsf{k}_{\mathsf{sk}}}\right)$.
- $\mathrm{Enc}\left(\mathsf{k}_{\mathsf{pk}},\mathsf{m}\right)$: Given a public key $\mathsf{k}_{\mathsf{pk}} \in \mathbb{G}$ and a message $\mathsf{m} \in \mathbb{G}$, $\mathrm{Enc}(\mathsf{k}_{\mathsf{pk}},\mathsf{m})$ randomly draws $r \leftarrow_{\$} \mathbb{Z}_p$ and outputs $\left(g^r, \mathsf{k}_{\mathsf{pk}}^r \odot \mathsf{m}\right)$.

- $\mathsf{Dec}(\mathsf{k_{sk}}, (\mathsf{c}_1, \mathsf{c}_2))$: Given a secret key $\mathsf{k_{sk}} \in \mathbb{Z}_p$ and a ciphertext $(\mathsf{c}_1, \mathsf{c}_2) \in \mathbb{G} \times \mathbb{G}$, the decryption algorithm Dec outputs the message $\left(\mathsf{c}_1{}^{-\mathsf{k_{sk}}} \odot \mathsf{c}_2\right)$.

For our later introduced PSI protocols, we require an AHE scheme. For ElGamal, we find no plaintext groups $(\mathbb{Z}_N, +)$ in which the DDH problem is considered hard. However, for an element $\mathsf{m} \in \mathbb{Z}_p$, we can encrypt the exponentiation g^{m} [28]. The security of the new scheme directly follows from the security of the underlying scheme (for any IND-CPA secure encryption). Remark that we can no longer (efficiently) decrypt since we would have to calculate the *discrete logarithm (*dlog) to retrieve m from g^{m}. However, for all our constructions, we only need to decrypt $\mathsf{Enc}(0)$ but can encrypt any plaintext $\mathsf{m} \in \mathbb{Z}_{|\mathbb{G}|}$.

2.3.3 (Leveled) Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) requires, in extension to AHE, the possibility to multiply ciphertexts with other ciphertexts. Given ciphertexts $\mathsf{c}_1, \mathsf{c}_2 \in \mathcal{C}$, we will write $\mathsf{c}_1 \boxdot \mathsf{c}_2$ for the homomorphic multiplication algorithm. With the BFV [17; 35] and BGV [18] schemes, two similar FHE schemes have been proposed that are based on a variation of the LWE problem over certain rings (so-called *ring* LWE). Even though these schemes allow bootstrapping [39], they are often used as so-called *leveled* FHE schemes. Leveled FHE schemes allow you to specify the number of homomorphic operations that can be correctly decrypted without a bootstrapping step. However, for an increasing number of homomorphic operations, the computational complexity of the scheme's algorithms also increases. Since homomorphic additions and scalar multiplications increase the error much less than ciphertext-ciphertext multiplications, the number of ciphertext-ciphertext multiplications is usually the crucial factor. BFV and BGV have the advantageous property that the plaintext space can be defined as $\mathbb{Z}_p^n$, for some $n \in \mathbb{N}$ and prime $p \in \mathbb{N}$. For increasing $\kappa \in \mathbb{N}$, n also increases and with it the number of messages $\mathbb{Z}_p$ that can be encrypted in one ciphertext. The decryption of homomorphically evaluated ciphertexts thus leads to component-wise additions or multiplications of the corresponding plaintext vectors. This property allows so-called *single instruction multiple data* (SIMD), where, e.g., the same scalar $s \in \mathbb{Z}_p$ can be homomorphically multiplied to all encrypted messages efficiently. Many state-of-the-art PIR protocols make use of BFV [17; 35] and BGV [18] and show how to outperform classic protocols based on the Paillier [71] or ElGamal schemes, as shown by Ali et al. [5].

2.4 Hashing

Depending on the usage, different requirements are placed on hash functions. We will use hash functions to map elements to indices of arrays, also called hash tables.

2.4.1 Universal Hashing

We will use *universal hash functions* for all our constructions and provide a simple definition based on *universal hashing families*.

Definition 2.2 (Universal Hashing Family)**.** For an $l \in \mathbb{N}$, a family of hash functions $\mathcal{H} \subseteq \{H \mid H : \{0,1\}^* \to \{0,\dots,l\}\}$ is called a *universal hashing family* if

$$\forall x, y \in \{0,1\}^*, x \neq y : \Pr_{H \in \mathcal{H}}[H(x) = H(y)] \leq \frac{1}{l}, \quad (1)$$

If we refer to a hash functions H, we assume H has been uniformly chosen from a universal hashing family $\mathcal{H}$. We then say H is a universal hash function.

2.4.2 Cuckoo Hashing

Placing each item of a set at the calculated hash index in a hash table is denoted as *simple hashing*. Different elements can map to the same index, which requires that multiple elements can be placed at the same hash table index in a so-called *bin*. Cuckoo hashing [70] makes use of multiple hash functions and, thus, multiple possible indices per element but only allows at most one item per hash table index. For k different hash functions $H_1, \dots, H_k$, that map elements to indices, Cuckoo hashing requires that every element is placed at one of the k different indices $H_1, \dots, H_k$. The challenge of Cuckoo hashing is to find a placement for the items that meets these requirements which is not always possible. However, it can be shown that the probability of Cuckoo hashing failures decreases rapidly with the hash table size as well as the number of hash functions [30; 38; 78]. We will later adjust the parameters such that failures occur with negligible probability in a statistical security parameter λ.

Blocked Cuckoo Hashing Dietzfeldinger and Weidling [31] proposed *blocked* Cuckoo hashing, a variation where instead of one item per position, for a fixed number $\delta \in \mathbb{N}$, up to δ elements are placed in the same bin. There are various strategies to create a δ-block Cuckoo hash table [31]. We use a *random-walk* approach, where for an insertion into a filled bin, a random element is swapped and reinserted into the table. Remark that blocked Cuckoo hashing is a generalization of Cuckoo hashing and thus, a 1-block Cuckoo hash table is equivalent to a standard Cuckoo hash table. With t, we refer to the table size, which is $t := (l \cdot \delta)$ for a blocked Cuckoo hash table. Blocked Cuckoo hashing can also be phrased as a compromise between Cuckoo hashing and so-called *k-choice* hashing. For blocked Cuckoo hashing no theoretical analysis of failure probabilities (for all parameters) exists [76] and is left for future work.

2.5 Hashing-based Private Set Intersection

For PSI, we cannot only use simple hashing and securely compare the elements in the filled bins because this would leak too much information. State-of-the-art PSI protocols combine 2PC building blocks like OT with customized hashing data structures, including Cuckoo hashing [30; 77], permutation-based Cuckoo hashing [74], 2D Cuckoo hashing [76], SpOT [72] and *probe-and-XOR of strings* (PaXoS) [73]. The basic idea is to use hashing structure where the size of each bin and table can be set independently of the elements.

2.5.1 Private Set Membership-based PSI

Private set membership (PSM) protocols can securely check for a client's item if it is part of a server's set and can be constructed, e.g., based on OT [77] or FHE [23]. We can efficiently extend PSM to (asymmetric) PSI protocols by using Cuckoo hashing [77]. The client places her items in a Cuckoo table $CT_{H_1,\ldots,H_k}$. The server uses the same hash functions $H_1,\ldots,H_k$ to place his elements in a simple hash table where each item is placed at the hash indices of all hash functions. Since both parties use the same hash functions, an item e that is part of the intersection will be placed in the server hash table bin corresponding to the client Cuckoo hash table index. The PSI Cuckoo hashing procedure with $k=2$ hash functions is illustrated in Figure 2. In all our figures, we illustrate items $a,\ldots,z$ as circled letters a, ..., z.

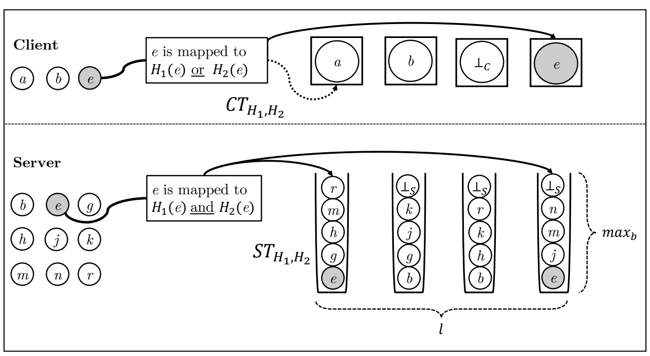


Figure 2: Structure of PSI protocols based on Cuckoo hashing and PSM. The element e and the k possible positions of e in the hashing scheme are highlighted. Dummy elements are denoted by $\perp_C$ for the client, respectively $\perp_S$ for the server.

To calculate the set intersection, the client securely checks for every Cuckoo hash table index if the placed item is contained in the corresponding server bin using a PSM protocol. For empty client table positions, dummy or random elements are used [77]. If using dummy elements, the client dummy element $\perp_C$ must be different from the server dummy element $\perp_S$. Also, PSM protocols might leak the number of items in the PSM input set. For PSM-based PSI protocols, we need to hide this information, otherwise, the client could learn the number of items per bin and, thus, extra information about the server's set. The bins of the simple table can be filled with dummy elements up to a constant maximum size $max_b \in \mathbb{N}$ to solve this problem. The maximum size max_b is chosen such that the probability that the server places more items in a bin is negligible [78]. As mentioned in subsubsection 2.4.2, Cuckoo hashing can fail. The occurrence of failures during a protocol execution can leak extra information about the items in the set. One need to choose the parameters so that the failure probability is negligible [30; 38; 77; 78].

2.5.2 PIR-based Private Set Intersection Protocols

With PIR-PSI [30], another Cuckoo hashing-based PSI approach for asymmetric PSI settings was presented. In PIR-PSI, the server uses Cuckoo hashing while the client uses simple hashing. Since the client might have multiple items per single table index, we cannot use PSM protocols as described before. To calculate the intersection, the client needs to securely compare her elements with items at the corresponding indices in the server Cuckoo hash table without leaking information about the used indices. We later formalize this functionality as *private indexed equality* (PIE) in Subsection 3.2. However, the switched hashing roles allow using other very communication-efficient MPC building blocks but also require the element comparison to hide the Cuckoo hash index on a match (from the client). Otherwise, the client would learn (too much) information about the server's Cuckoo item placement and, thus, the server items themselves. PIR-PSI utilizes FSS with at least two non-colluding semi-honest servers to perform the secure comparison.

3 Our Protocol

In this chapter, we will present our PSI protocol constructions which can be phrased as a single server PIR-PSI solution with an improved hashing structure.

3.1 Nested Cuckoo Hashing

Our hashing approach, called *nested Cuckoo* hashing, combines client Cuckoo hashing as in PSM-PSI protocols [74] with server-sided Cuckoo hashing as by Demmler et al. [30]. We can thus, reduce the number of needed element comparisons for PIR-based PSI schemes (without using the *binning* of Demmler et al. [30]). Nested Cuckoo hashing could also be used with the FSS-based multiserver approach of PIR-PSI [30]. We combine nested Cuckoo hashing with adjusted PSM protocols based on the exponential ElGamal AHE scheme and the BFV and BGV (leveled) FHE schemes. The resulting (single-server) PSI protocols can also be phrased as a Cuckoo hashing with PSM protocol, as described in subsubsection 2.5.1, but with a PSM protocol based on server-sided Cuckoo hashing and modified PIR constructions.

In the proposed PSM-based PSI protocols with Cuckoo hashing [77], the client uses k_1 hash functions $H_1, \dots, H_{k_1}$ to hash her items into a Cuckoo hash table. The server uses the same hash functions to store his items in the bins of a simple hash table. In comparison, nested Cuckoo hashing extends the server's simple hashing with an additional hashing step. For every simple table bin i, the server places the corresponding items in a (blocked) Cuckoo hash table CT_S^i using a second set of k_2 hash functions $H'_1, \dots, H'_{k_2}$. Remark, the server can use the same k_2 hash functions for every bin, which we assume for the rest of this paper. We denote the client Cuckoo table as the *outer* Cuckoo hash table with k_1 hash functions mapping to $\{1, \dots, l_1\}$. The server's Cuckoo hash tables are denoted as *inner* Cuckoo hash tables with k_2 hash functions mapping to $\{1, \dots, l_2\}$.

The nested Cuckoo hashing construction for PSI is illustrated in Figure 3 where $k_1 = k_2 = 2$, $|Y| = \frac{|X|}{2} = 6$, $l_1 = 4$, and $l_2 = 3$.

The hashing scheme of Pinkas et al. [77] guarantees that if an item e is in the intersection of the client and server's PSI input sets, then e is included in the server bins at indices $H_1(e), \dots, H_{k_1}(e)$. With nested Cuckoo hashing, we can further specify that e is placed at exactly one index $H'_1(e), \dots, H'_{k_2}(e)$.

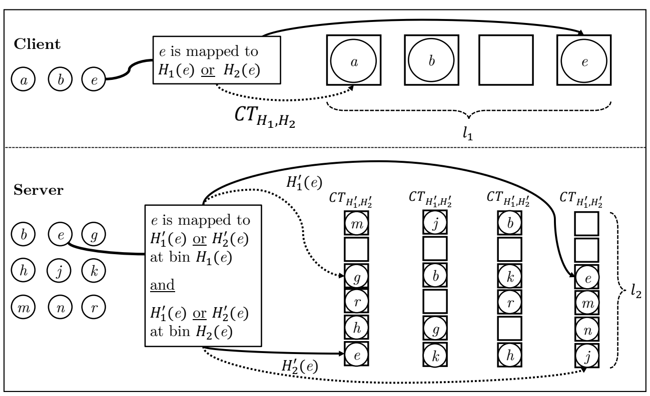


Figure 3: Client Cuckoo Hashing and server nested Cuckoo hashing for parameters $k_1 = 2$, $l_1 = 4$, $k_2 = 2$, and $l_2 = 3$.

3.2 Generic Private Set Intersection Protocol

Our nested Cuckoo hashing scheme allows PSI protocols to be based on a protocol that securely compares an item with multiple items at given indices. We call this functionality *private indexed equality* (PIE). In detail, for a given $k, N \in \mathbb{N}$, the $\text{PIE}_1^{k \times N}$ functionality is a two-party functionality between a receiver and a sender. The sender inputs an array of N items $A := (a_1, \dots, a_N)$. The receiver inputs an index set $J \subseteq 2^{\{1,\dots,N\}}$ with $|J| = k$ and an item e. As output, the receiver only learns a bit indicating whether the item e is equal to at least one item a_j at any index $j \in J$. Remark that the receiver shall not learn the index of a match. Our definition of PIE can also serve as a generalization of the approach used by PIR-PSI [30]. The PIE functionality is illustrated in Figure 4.

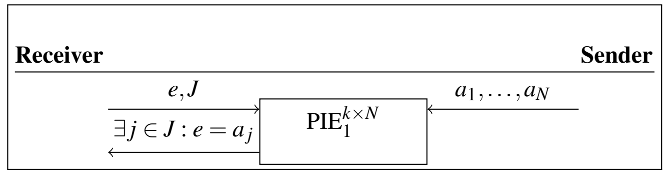


Figure 4: $\text{PIE}_1^{k \times N}$ functionality with one receiver item e, $k = |J|$ indices, and N sender elements $a_1, \dots, a_N$.

By combining nested Cuckoo hashing and a protocol for the PIE functionality, called PIE protocol, we can build efficient and secure PSI protocols as described in the following. The client uses hash functions $H_1, \dots, H_{k_1}$ to place her items in a Cuckoo hash table CT_C and initializes an empty set R. The server uses hash functions $H_1, \dots, H_{k_1}$ and $H'_1, \dots, H'_{k_2}$ to place his items in a nested Cuckoo hash table $(CT_S^1, \dots, CT_S^{l_1})$.

For each index i in the client's outer Cuckoo hash table, the client and server run a $\text{PIE}_1^{k_2 \times l_2}$ protocol with the corresponding item $e := CT_C[i]$ and the indices $J := \{H'_1(e), \dots, H'_{k_2}(e)\}$, where the server inputs the ith Cuckoo table CT_S^i. The PIE protocol outputs to the client whether e is equal to $CT_S^i[j]$ for any $j \in J$. If the PIE protocol outputs 1, the client adds e to the result set R. After the loop over all Cuckoo hash table indices, the client outputs R. Naturally, depending on the hash parameters, many Cuckoo and nested Cuckoo hash table entries are empty (cf. [72]). For empty client Cuckoo hash table positions, the client inputs a dummy element $\perp_C$ to the PIE. The server places dummy element $\perp_S$ at every empty nested Cuckoo hash table position. The dummy elements need to be different (i.e., $\perp_C \neq \perp_S$) and should also not be valid input items in order to avoid information leakage and false positives. If we use $\perp_S = 0$, we gain some performance improvements, as mentioned in Subsection 4.2. The nested hashing construction guarantees that if and only if e is in the intersection, e is placed at index j in the Cuckoo table CT^i at the server for exactly one $j \in J$. The generic PIE-based PSI protocol using nested Cuckoo hashing is presented in Figure 5.

The sizes of all hash tables (and the stash) t_1, t_2 have to be set prior to the protocol execution according to the (public set) sizes of the server and client. To build a secure PSI protocol using Cuckoo hashing or nested Cuckoo hashing approach, one needs to adjust the hashing parameters (i.e., k_1, k_2 and l_1, l_2) such that the probability of hashing failures is below a certain threshold (e.g., 2^{-40}). Related works on PSI with Cuckoo hashing [30; 74; 78] have empirically measured needed slack factors β_1 such that for $t_1 = l_1 = \beta_1 \cdot |Y|$[1], the probability of client Cuckoo hashing failures is sufficiently small. In comparison to Pinkas et al. [77], when using nested Cuckoo hashing, the server places the items of each simple hashing bin in a

[1] In the case of single-server Cuckoo hashing.

Cuckoo table. As such, we need to set $t_2 = \beta_2 \cdot max_b$ where max_b is an upper bound on the maximum simple hashing bin size. For any fixed values of t_1 and $|X|$, we can find an $m_b \in \mathbb{N}$ such that $max_b = \frac{|X|}{t_1} + m_b$ is an upper bound on the maximum simple hashing bin size with sufficiently high probability [30; 78]. Since the server creates t_1 Cuckoo tables, if we assume an independent failure probability per table, we need a failure probability of approximately $\frac{2^{-40}}{t_1}$ per table to achieve an overall nested Cuckoo hashing failure probability of 2^{-40}. However, since the failure probabilities are not independent and the average bin size is just $\frac{|X|}{t_1}$, we expect the average failure probability per nested Cuckoo hash table is much lower. A detailed analytical analysis of the failure probability of Cuckoo hashing (and thus also nested Cuckoo hashing) is still an open research question and left for future work.

3.3 Private Set Intersection from AHE

By using AHE, in this section, we will construct a PIE protocol and thus, a PIE-based PSI protocol. This protocol is represented in Figure 5 and with pseudocode in Appendix A. Assume we have an AHE scheme $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ with a message space $\mathbb{Z}_p$ for a prime p. A $\mathrm{PIE}_1^{k_2 \times N}$ scheme based on Π can be constructed as follows. The client first generates a key pair $(\mathsf{k_{sk}}, \mathsf{k_{pk}}) \leftarrow \mathrm{Gen}(1^\kappa)$ and sends the public key $\mathsf{k_{pk}}$ to the server in a one-time setup phase. For a PIE input element e, the client encrypts e as $\mathrm{Enc}(e)$. For all indices j in the index set J, the client creates an N-dimensional encrypted index vector EIV with $EIV_j = \mathrm{Enc}(1)$ and $EIV_i = \mathrm{Enc}(0)$ for all $i \in \{1, \dots, N\} \setminus \{j\}$. The client sends the public key $\mathsf{k_{pk}}$, the encrypted message $\mathrm{Enc}(e)$, and all EIVs to the server. For each EIV the server computes $\mathsf{c} \leftarrow \langle EIV, A \rangle$, $c_d \leftarrow \mathsf{c} \boxminus \mathrm{Enc}(e)$, and finally $c_f \leftarrow c_d \boxdot r$ with a random $r \leftarrow_\$ \mathcal{M} \setminus \{0\}$. All server calculations can be performed using Π's plaintext multiplication and homomorphic addition algorithms. Remark that c is an encryption of the jth server element, and c_d is an encryption of the subtraction between e and a_j. As such, multiplying c_d with a random element $r \leftarrow_\$ \mathcal{M} \setminus \{0\}$, c_f yields an encryption of 0 if $e = a_j$ and an encryption of a uniformly random element (unequal to 0) otherwise. Before sending back c_f to the client, the server shuffles the c_fs for all $j \in J$ such that the client does not learn the index of a match.

Sublinear Complexity Our AHE-based PIE protocol requires the client to send an N-dimensional encrypted index vector. Thus, the communication complexity is linear in the size of the server array. We can use an approach similar to Kushilevitz and Ostrovsky [62] to reduce the complexity. Assume $N = N_1 \cdot N_2$ with $N_1, N_2 \in \mathbb{N}$, the server can place each Cuckoo table vector $A = (a_1, \dots, a_N)$ in a $(N_1 \times N_2)$-dimensional matrix $A' = (a'_{i,j})$ where $a'_{i,j} = a_{(i-1) \cdot N_2 + j}$ for $i \in \{1, \dots, N_1\}$ and $j \in \{1, \dots, N_2\}$. The client sends the public key and her encrypted item $\mathrm{Enc}(e)$ to the server. For each hash index $i \in \{1, \dots, k_2\}$, the client computes the column index $j' := (j \mod N_1) + 1$ and sends an N_1-dimensional encrypted index vector EIV with $EIV_{j'} = \mathrm{Enc}(1)$ and $EIV_i = \mathrm{Enc}(0)$ for all $i \in \{1, \dots, N_1\} \setminus \{j'\}$. The server computes the homomorphic dot product of all EIV^i with every column in A', i.e., the homomorphic matrix-matrix product $A' \boxdot (EIV^1, \dots, EIV_2^k)$. For every resulting encrypted array entry c, the server again subtracts $\mathrm{Enc}(e)$ and multiplies with a new random $r \leftarrow_\$ \mathbb{Z}_p^\times$. The server now shuffles all $k_2 \cdot N_2$ elements before sending them back. The client needs to check whether one of the received items is an encryption of 0. By adjusting N_1 and N_2 differently, we obtain the possibility to vary between index vector size ($O(k_2 \cdot N_1)$) and response set size ($O(k_2 \cdot N_2)$). Remark, if we set $N_1 = N$ and $N_2 = 1$, we get our unimproved protocol. If we set $N_1 = 1$ and $N_2 = N$, we get a simple PSM protocol. However, if we adjust $N_1 = N_2 = \lceil \sqrt{N} \rceil$ we achieve a protocol with sublinear (square-root) communication complexity. For all $i \in \{N+1, \dots, N_1 \cdot N_2\}$, we can add server dummy elements to A. By introducing a parameter $\sigma \in \mathbb{R}^+$, called *skewness*, and set $N_1 \lceil \sqrt{N \cdot \sigma} \rceil$ and $N_2 = \lceil \sqrt{\frac{N}{\sigma}} \rceil$, we can consider varying fractions $\frac{N_1}{N_2}$. Notice, when using the sublinearity improvement, the resulting protocol is not a secure PIE protocol anymore, since the client also learns whether her items equal other server items at indices in the same matrix row (which are not in J). However, for our PSI construction, it is sufficient since the server items are distinct, J is fully determined by e, and the server only places e at $j \in J$.

Blocked Cuckoo hashing For sublinear communication complexity, instead of Cuckoo hashing each bin and placing it in a matrix, the server can directly use blocked Cuckoo hashing (as described in subsubsection 2.4.2). The server uses δ-block Cuckoo hash tables with $\delta = N_2$ to reach a similar protocol with the same sublinear communication but a smaller hashing failure probability [72]. The failure rate is empirically evaluated in Subsection D.7.

Multi-table Cuckoo hashing In the original paper on Cuckoo hashing by Pagh and Rodler [70], each hash function H_i points to a different hash table $CT[i]$. It can be beneficial to use multiple tables per Cuckoo hash table, where a hash function H_i maps an element e to $CT[i][H_i(\mathsf{m})]$. As such, the Cuckoo table is a $(k \times l)$-dimensional matrix of size $t := k \cdot l$, where $\{0, \dots, l\}$ is the range of the hash functions. The k separate hash tables can be smaller than a single-table Cuckoo hashing under comparable hashing failure probabilities (as shown in Subsection D.7). Using multi-table Cuckoo hashing, we can adjust our PIE protocols for PSI as follows. For each $q \in \{1, \dots, k_2\}$, instead of comparing an item e to the k_2 items $CT[H'_1], \dots, CT[H'_{k_2}]$ in the same single-table Cuckoo hashing vector, we compare it to the elements $CT'[1][H'_1(e)], \dots, CT'[k_2][H'_{k_2}(e)]$ in a multi-table Cuckoo

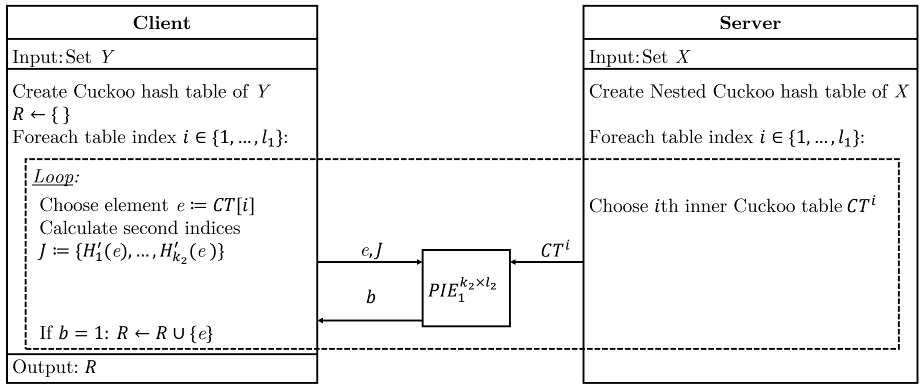


Figure 5: Generic PSI protocol using nested Cuckoo hashing and PIE.

hash table. Assuming that $|CT| = k_2 \cdot |CT'[q]|$, we can reduce the communication and computation complexity per PIE by factor k_2.

Precomputed Encrypted Index Vector As described in subsubsection 2.3.1, the server can compute the XOR of an encrypted bit and a plaintext bit. Using XOR for AHE schemes, we can shift the transfer of the encrypted index vector to an item-independent precomputation phase as follows. For each outer index i, the client generates k_2 random l_2-bit vectors r^{ERV} and sends all $ERV := \text{Enc}(r_{ERV})$ bit-wise encrypted to the server. Later in the online phase, for an element e at index i, instead of sending the large encrypted index vectors EIV, the client flips the $H'_j(e)$th bit of r^{ERV} and sends it as plaintext to the server. The server now bit-wise computes the XOR of the encrypted randomness vector ERV and the plaintext randomness r^{ERV}. Since the $H'_j(e)$th bit has been flipped in r^{ERV}, the resulting ciphertext is an encryption of 1 at position $H'_j(e)$ and an encryption of 0 otherwise. The random vectors r^{ERV} can be generated using a *pseudo random function* (PRF) thus, the client does not need to store the potentially large random vectors between the precomputation phase and the online phase.

Supporting Stashes We can include stashes for every inner Cuckoo hash table on the server side but additionally must hide if a matching item was part of the stash or the hash table (in the PIE step). The server computes the encrypted comparison of every stash element with $\text{Enc}(e)$ (by subtracting and multiplying with randomness). The encrypted comparison results are added to the result list before shuffling. The full protocol PSI protocol with stashes and δ-block Cuckoo hashing is given in Algorithm 1 on page 28.

3.4 Exponential ElGamal-based Protocol

The scheme can directly instantiate our generic construction (in Subsection 3.2). This subsection presents an improved protocol instantiation with exponential ElGamal to deal with the impossibility of efficiently decrypting arbitrary ciphertexts, computational improvements, and server privacy aspects that are not covered by our generic scheme.

Avoiding Ciphertext Decryption For exponential ElGamal, since we assume the DDH problem (and thus, the discrete logarithm) to be difficult in $\mathbb{G}$, we cannot decrypt arbitrary ciphertext efficiently. However, for our protocol, we only need to check if an encrypted element decrypts to 0. This check can be performed by computing ${\mathsf{c}_1}^{\mathsf{k_{sk}}}$ which equals to c_2, if and only if $(\mathsf{c}_1, \mathsf{c}_2) = \text{Enc}'(0; r)$.

Simultaneous Multi-Exponentiation Given an encrypted index vector $EIV = (EIV_1, \dots, EIV_N)$ and a vector $A = (a_1, \dots, a_N)$ as in our generic AHE-based PIE protocol. The homomorphic dot product is defined as

$$\langle EIV, A \rangle := (EIV_1 \boxdot a_1) \boxplus \cdots \boxplus (EIV_N \boxdot a_N). \tag{2}$$

Let $\mathsf{c} := (\mathsf{c}_1, \mathsf{c}_2) := \langle EIV, A \rangle$. If we instantiate the encryption with the exponential ElGamal scheme, we can write

$$EIV = \left((EIV_1^1, EIV_1^2), \dots, (EIV_N^1, EIV_N^2)\right) \in (\mathbb{G} \times \mathbb{G})^N$$

and the calculation of the dot product can be simplified as

$$\mathsf{c}_1 = \bigodot_{1 \le q \le N} \left(EIV_q^1\right)^{a_q} \text{ and } \mathsf{c}_2 = \bigodot_{1 \le q \le N} \left(EIV_q^2\right)^{a_q}. \tag{3}$$

Computations, as in Equation 3, are called *simultaneous multi-exponentiation* and different algorithms have been proposed to increase the efficiency compared to a naïve approach [68]. Especially for elliptic curve cryptography (ECC) systems, simultaneous multi-exponentiation algorithms can be adjusted to certain elliptic curve types to further increase the computational performance [47]. Our exponential ElGamal-based PSI scheme can directly benefit from these algorithms to improve the homomorphic dot product computation.

Extended Precomputation We can extend the precomputation approach described in Subsection 3.3 by performing additional computations on the server. The extended precomputation requires that the input set is already available on the server. Assume the server receives an encrypted randomness vector

$$ERV = \left(ERV_1 = \mathrm{Enc}(r_1^{ERV}), \ldots, ERV_N = \mathrm{Enc}(r_N^{ERV})\right)$$

(as in Subsection 3.3). Instead of waiting for the client to send the plaintext randomness, the server creates two new vectors V_0, V_1 using its input $A = (a_1, \ldots, a_N)$ as

$$V^1 := (ERV_1 \boxdot a_1, \ldots, ERV_N \boxdot a_N) \tag{4}$$

$$V^0 := \left(a_1 \boxminus V_1^1, \ldots, a_N \boxminus V_N^1\right). \tag{5}$$

Note that if the client has sent randomness $r_i^{ERV} = 1$, then $V_i^0 = \mathrm{Enc}(a_i)$ and $V_i^1 = \mathrm{Enc}(0)$. Analogue, if $r_i^{ERV} = 0$, then $V_i^0 = \mathrm{Enc}(0)$ and $V_i^1 = \mathrm{Enc}(a_i)$. After receiving the bit-flipped plaintext randomness r'^{ERV}, the server calculates

$$\mathsf{c} := V_1^{r_1'^{ERV}} \boxplus \cdots \boxplus V_N^{r_N'^{ERV}}. \tag{6}$$

If the bit $r_i'^{ERV}$ has been flipped, then $V_i^{r_i'^{ERV}} = \mathrm{Enc}(a_i)$. Otherwise, $V_i^{r_i'^{ERV}} = \mathrm{Enc}(0)$. The presented precomputation approach allows performing the homomorphic scalar multiplications for the dot product in an offline phase (with available server input). Thus, only (potentially) faster homomorphic additions must be computed in the online phase. Remark, the extended precomputation also works for the generic AHE-based PSI scheme but cannot be combined with the batched computation presented in Subsection 3.5.

Server Privacy In all our ElGamal-based constructions and implementations, we are using groups $\mathbb{G}$ for ElGamal-based schemes in which the DDH problem is assumed to be hard. The DDH assumption leads to an IND-CPA secure encryption scheme, which, however, does not imply that the client (which holds the secret key) cannot receive additional information from c_f as used in our AHE-based construction. The server might not use freshly drawn randomness in the homomorphic calculations, especially when using simultaneous multi-exponentiation algorithms, which can lead to insecure PSI protocol instantiations. For our improved ElGamal-based PSI protocol, we perform an additional so-called *rerandomization* step at the end of each PIE. The server draws randomness $r \leftarrow_\$ \mathbb{Z}_p$ and calculates $c_f \leftarrow c_f \boxplus \mathrm{Enc}(0; r)$. In Section 4, the necessity for the rerandomization step is analyzed in more detail.

3.5 BGV/BFV-based Protocol

As a second protocol instantiation, we will use the BGV and BFV (leveled) FHE schemes. In this section, with $\mathbb{Z}_p^n$, we refer to the message space of the (leveled) FHE schemes for a prime $p \in \mathbb{N}$.

Batched Computation State-of-the-art performances of 2PC protocols based on (leveled) FHE schemes heavily rely on SIMD operations. Our BGV/BFV-based PSI protocol uses a similar packed encoding as shown by Chen et al. [22; 23]. Each outer Cuckoo table (consisting of the client elements and dummy elements $\perp_C$) is encoded in one plaintext and thus encrypted to one ciphertext. So instead of encrypting $\mathrm{Enc}(CT[i])$ for every $i \in \{1, \ldots, l_1\}$, we directly encrypt $\mathrm{Enc}((CT_C[1], \ldots, CT_C[l_1]))$. We do the same for all EIVs across all outer Cuckoo hash indices. Let EIV^i be the ith sent encrypted index vector as in Subsection 3.2. For each $j \in \{1, \ldots, k_2\}$, the client sends a packed EIV', with

$$\begin{aligned} EIV_1' &= \mathrm{Enc}\left(\left(EIV_1^1, \ldots, EIV_1^{l_1}\right)\right) \\ \vdots \quad & \quad \vdots \\ EIV_{l_2}' &= \mathrm{Enc}\left(\left(EIV_{l_2}^1, \ldots, EIV_{l_2}^{l_1}\right)\right) \end{aligned} \tag{7}$$

As such, we can avoid the outer `For` loop shown in Figure 5 and perform a batched PIE computation. The batched PIE can be performed similarly to the server computation in the generic AHE-based protocol in Subsection 3.3. We will not cover the details of the plaintext encoding but refer to the rich literature [17; 18; 35]. However, for the shuffling of the server's encrypted result list, a problem arises, as shown in the next paragraph. We discuss a different packing approach beneficial for small $|Y|$ in Appendix E.

Hide Cuckoo Locations In our generic protocol, we randomly permute the vector of encrypted results per inner Cuckoo table. As such, the client does not learn the hash function index $q \in \{1, \ldots, k_2\}$ of a matching item (or if the item has been placed in the stash). When using packed encryptions, as described in the previous paragraph, we cannot (efficiently) permute the element independently across each inner Cuckoo table. However, if the server does not permute the elements independently, the client might learn the index of the hash function H_q', where an element has been placed. We propose a solution based on BGV and BFV's ciphertext-ciphertext multiplications as follows. Let $(c_d)_q$ be the result of the PIE homomorphic subtraction for hash function H_q' with $q \in \{1, \ldots, k_2\}$. For a client PIE input element e, the ciphertext $(c_d)_q$ is an encryption of $a_{H_q'(e)} - e$ for a server input $(a_1, \ldots, a_{l_2})$. Before multiplying with a random $r \leftarrow_\$ \mathbb{Z}_p^\times$, the server computes

$$\mathsf{c}_d' := (c_d)_1 \boxdot \cdots \boxdot (c_d)_{k_2}. \tag{8}$$

Remark, c'_d is an encryption of 0, if and only if e equals to $a_{H'_q(e)}$ for one $q \in \{1, \ldots, k_2\}$. For small k_2, this approach is efficient and reduces the number of decryptions for the client.

When using blocked Cuckoo hashing, the server again needs to randomly permute the bins (before the homomorphic computations) to hide matching bin indices from the client.

Circuit Privacy As for the exponential ElGamal-based Protocol, using an IND-CPA secure HE scheme does, in general, not hide the server input from the client. In the literature, hiding the server input and the (circuit) structure of the computation, is referred to as *circuit privacy*. Circuit privacy can be achieved by an additional *bootstrapping* step or *noise flooding* [40; 43]. However, for BGV and BFV, more efficient constructions for circuit privacy have been proposed[15]. For PSI, we might even construct more efficient approaches that only hide the server inputs but not the (public) circuit structure. In theory, if the PIE server result c_f encrypts a random value, we will assume that a PPT simulator exists such that the client cannot distinguish c_f from a simulated ciphertext (that has no access to the server input).

Arbitrarily Sized Plaintexts HE schemes only support evaluation on constant predetermined plaintext bitlengths. When evaluating our protocol with longer plaintexts, we need to adjust our protocol. We proceed analogously to the Microsoft APSI library based on Cong et al. [27]. This proceeds roughly as follows. Each word is separated into multiple subwords that are each short enough to be processed by our regular PSI scheme. We use our normal scheme to inform the client about partial matches, which they can then use to check complete matches. This leaks extra information about the server's set and can lead to false positives. In order to alleviate security concerns, an OPRF is applied to the words before they are separated into separate subwords. This means that the client no longer learns anything about the server's set from partial matches, and false positives now happen at random, but with very small probability. For performance reasons, the words can also be hashed and truncated beforehand. Runtime analysis of this protocol is simple, since its scales linearly with the length of the words after truncation. If each word is separated into, for example, three subwords, then it is equivalent to running our normal protocol on client and server sets that is three times the size. For a more detailed overview, we refer to the work of Cong et al. [27].

4 Analysis

With π, we will refer to our generic AHE-based PSI protocol as described in Subsection 3.2). Remark, in theory, X and Y are also bit-string that encode sets of fixed cardinality consisting of ρ-bit strings. Let $\mathbb{Z}_p$ be the plaintext space of Π.

4.1 Security

Our PSI protocol is secure against semi-honest adversaries in the standalone model without *random oracles*. We do not provide a security proof for the more generic PIE-based construction in Subsection 3.2, which, however, would be straightforward using parts of the correctness proof of subsubsection 4.1.1 and simulators for the views of the underlying semi-honest secure PIE protocol. For simplicity, we omit notations for the universes of indices for indexed probability ensembles. So, e.g., we write X instead of $X \in 2^{\{x|x\in\{0,1\}^\rho\}}$. We separate the semi-honest security into independent claims for *correctness*, *client privacy*, and *server privacy*. The details of the analysis and the proofs can be found in Appendix B.

4.1.1 Correctness

Correctness says that the output of the PSI protocol shall not be (computationally) distinguishable from the ideal output $X \cap Y$ as follows.

[Correctness] Our PSI protocol π provides correctness, meaning that,

$$\{\text{output}^\pi(X,Y,\kappa)\}_{X,Y,\kappa} \overset{c}{\equiv} \{(\emptyset, X \cap Y)\}_{X,Y} \ .$$

The correctness of BGV/BFV-based protocol also directly follows from subsubsection 4.1.1 and the description in Subsection 3.5. As mentioned in subsubsection 2.3.3, leveled FHE schemes allow you to specify the number of homomorphic operations that can be performed such that an operated ciphertext can still be correctly decrypted. We will not go into detail about how to choose the correct parameters for BGV and BFV but refer to the rich literature [9; 17; 18; 35; 58]. Remark that the number of needed homomorphic scalar multiplications and additions can be deduced from the public parameters k_1, k_2, l_1, l_2.

4.1.2 Client Privacy

In our PSI construction, client privacy follows from the security of the underlying encryption scheme (similarly to [23]). Loosely speaking, we can efficiently simulate server protocol views such that if an attacker could distinguish the real protocol view of the server from our simulated one, the attacker could break the underlying encryption scheme. To model the PSI input set sizes as public parameters, the simulator for the server's view additionally receives the client's set size $|Y|$ as input.

[Client Privacy] Let Π be an IND-CPA secure AHE scheme, then, our generic PSI protocol π instantiated with Π (as shown in Algorithm 1) provides *client privacy*, i.e., there exist a *PPT* algorithm Sim_S, such that

$$\{view_S^\pi(X,Y,\kappa)\}_{X,Y,\kappa} \overset{c}{\equiv} \{\text{Sim}_S(1^\kappa, X, \emptyset, |Y|)\}_{X,Y,\kappa} \ .$$

4.1.3 Server Privacy

The simulator for the client's view of our PSI protocol π additionally receives the server's set size $|X|$ as input. Remark, server privacy is not implied by the IND-CPA security of the underlying AHE encryption scheme. We need to assume that the client does not learn anything else from a ciphertext $c_f := (\langle EIV, A\rangle \boxminus \mathrm{Enc}(e)) \boxdot r$ than a bit b indicating $e = a_j$ (for an index vector EIV with $\mathrm{Enc}(1)$ at position j and $\mathrm{Enc}(0)$, otherwise). We treat this problem theoretically for any AHE scheme (according to our definition in subsubsection 2.3.1) but mention that this can be solved by *rerandomization* or *circuit privacy*. More formally, we assume that simulators Sim'_b for $b \in \{0,1\}$ exist that output an encrypted ciphertext. If $e = a_j$, Sim'_1 shall be computationally indistinguishable from the correct c_f (as in Algorithm 1 with $\mathrm{Dec}(c_f) = 0$). Likewise, for $e \neq a_j$, Sim'_0 shall be computationally indistinguishable from the c_f (with $\mathrm{Dec}(c_f) = r$ for $r \leftarrow_{\$} \mathbb{Z}_p^{\times}$). Loosely speaking, Sim'_0 simulates server results that decrypt to randomness $r \neq 0$ for non-matching elements, while Sim'_1 simulates server results that decrypt to 0 for matching client elements.

[Server Privacy] Assume simulators $\mathrm{Sim}'_0, \mathrm{Sim}'_1$ (as described in subsubsection 4.1.3) exist for the *PSI* protocol π (as shown in Algorithm 1), then, π provides *client privacy*, i.e., a *PPT* algorithm Sim_C exists, such that

$$\{\mathrm{view}_C^{\pi}(X,Y,\kappa)\}_{X,Y,\kappa} \overset{c}{\equiv} \{\mathrm{Sim}_C(1^{\kappa}, Y, X \cap Y, |X|)\}_{X,Y,\kappa} .$$

4.2 Complexity

To ease the understanding of our complexity analysis, we recall the most important parameters. The size of client and server's sets are $|Y|$ and $|X|$, respectively. The number of outer and inner hash functions are k_1 and k_2, respectively. For multi-table Cuckoo hashing, the table sizes of the (δ-block) Cuckoo hash tables are $t_1 = \beta_1 \cdot k_1 \cdot l_1$ and $t_2 = \beta_2 \cdot k_2 \cdot l_2$ for the outer and inner Cuckoo hash tables, respectively.

The sizes of parameters like $\beta_1, \beta_2, k_1, k_2, l_1, l_2$ depend on other parameters and on the security parameter $\kappa \in \mathbb{N}$. However, we will omit an explicit parameterized notation (e.g., $l_1(\kappa, k_1, \beta_1)$) and refer to the parameter description in Subsection 3.1. With t_1 and t_2, we denote the number of elements (including dummy elements) in the outer Cuckoo table and each inner Cuckoo hash table, respectively.

We assume that the server uses multi-table δ-block inner Cuckoo hash tables, and thus, $t_2 = k_2 \cdot \delta \cdot l_2$. An analysis for single-table hash tables is analogous. Let γ denote the bit-length of an encoded ciphertext. For simplicity, assume the encoded parameters, keys, and set sizes $|Y|$, $|X|$ have been exchanged in a precomputation/setup phase.

[Complexity] The complexities of our PSI protocol π can be summarized as follows:

- The client has a computation complexity of
$$O\left(\frac{1+\sigma}{\sqrt{\sigma}} \cdot \sqrt{\cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \qquad (9)$$
- The server has a computation complexity of
$$O(\beta_1 \cdot \beta_2 \cdot |Y| \cdot m_b + \beta_2 \cdot k_1 \cdot |X|) \qquad (10)$$
- The communication complexity is
$$O\left(\gamma \cdot \frac{1+\sigma}{\sqrt{\sigma}} \cdot \sqrt{\cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \qquad (11)$$

5 Implementation and Evaluation

PSI protocols are custom 2PC and, thus, MPC protocols with a high practical value. Considering the practical relevance, we have implemented and empirically evaluated the performance of our protocol.

5.1 Implementation

The implementation of our protocols is based on the `libscapi` [11] framework and the OpenFHE library [9]. As an underlying universal hash function family, we have implemented tabulation hashing [79], which has been shown to provide reasonable failure rates for Cuckoo hashing. For the δ-block Cuckoo hashing insertion step, a random-walk strategy is used [31], i.e., if a bin exceeds the maximum bin size δ, a randomly chosen element inside that bin gets replaced.

Exponential ElGamal We have implemented the exponential ElGamal scheme based on `libscapi`'s ElGamal implementation, which can be used with various underlying DDH-secure groups. Our implementation uses `libscapi`'s wrappers for `OpenSSL`'s ECC implementations [86], including the *wNAF-based interleaving exponentiation method* [68] for simultaneous multi-exponentiation. For the PRF used in our precomputation extension, we use `libscapi`'s PRF implementation based on the *advanced encryption standard* (AES) [32].

BGV/BFV OpenFHE provides a generic interface for the (leveled) FHE schemes with batched computations which we have used to implement our BGV/BFV-based protocol. As mentioned before, to achieve circuit privacy and, thus, server privacy, we need an additional bootstrapping, noise-flooding, or OPRF step. We have omitted circuit privacy in our implementation and evaluation and assume a preliminary OPRF step as in [27]. The BGV and BFV implementations in OpenFHE use variations that have been presented in subsequent works [42; 58] and improve the computational performance of the original constructions [17; 18; 35].

More details about the implementation are given in Appendix C.

5.2 Performance Evaluation

In this section, we show that our protocols are practical for large server sets of a million server items and client set sizes of thousands of elements. We evaluate our protocols on an *Amazon Web Services, Inc.* (AWS) cloud instance powered by an Intel Xeon Scalable processor (Skylake 8151) with 24 virtual cores, up to 4.0 gigahertz (GHz) clock speed, and 192 gigabytes (GB) of random-access memory (RAM). The parameters of the cloud instance are chosen to be comparable to previous work on FHE-based protocol [27]. We use the *Ubuntu 20.04 LTS* Linux distribution as operating system and run the client and server on the same system. The client and server communicate over a *virtual network loopback interface* with unrestricted bandwidth. The client uses a single thread for all evaluations. Except the evaluation in subsubsection 5.2.2 and D.5, the server uses one main thread (as described in Subsection 5.1) and an additional thread for the homomorphic computations. However, the main thread one proceeds if at least one other thread has terminated and thus, we also do not count the main thread.

We evaluate the performance for each phase (e.g., online phase) in terms of communication and computational costs. The communication costs are measured as the median of transmitted data in megabytes (MB). The transmitted data denote the sum of sent and received data by the client (except in Figure 13, where we differentiate between incoming and outgoing communication costs). We only consider application data and omit the overhead of underlying transmission control protocol (TCP) protocol. To measure the joint computation cost of the client and server per phase (including the communication), at the end of each phase, the server signals the client that he has finished the computation. The computation costs are measured by the client as median running times in seconds (s).

5.2.1 Different Encryption Schemes

$\lvert X \rvert$	2^{16}	2^{20}	2^{21}	2^{22}
Elgamal	187.51	720.23	—	—
BFV	1.38	6.89	10.25	25.24
BGV	1.61	8.66	14.27	20.79

Table 6: Total PSI computation costs of our BGV/BFV and ElGamal-based PSI schemes. For ElGamal, $\rho = 128$, while for BGV and BFV, $\rho = 32$ with $\lvert Y \rvert = 4096$.

In this subsection, we will compare the BGV/BFV-based protocol implementation with the exponential ElGamal-based version. Due to implementation restrictions (as also in [23]), we compare item bit-lengths $\rho = 32$ for BGV/BFV with $\rho = 128$ for ElGamal. Figure 6 shows the total running times of our PSI protocol version, and Figure 7 shows the communication costs. For BGV and BFV, we make use of packed ciphertexts and SIMD operations (as described in Subsection 3.5).

$\lvert X \rvert$	2^{16}	2^{20}
Elgamal	55.88	143.94
BFV	25.19	50.39
BGV	26.51	51.70

Table 7: PSI communication costs in MB of our BGV/BFV and ElGamal-based PSI schemes. For ElGamal, $\rho = 128$, while for BGV and BFV, $\rho = 32$.

The computation time is about 100 times lower when using BGV/BFV instead of exponential ElGamal and also show that for our PSI protocol, BFV is faster than BGV. However, we compare bit sizes of $\rho = 32$ for BGV/BFV with $\rho = 128$ bits for exponential ElGamal. Larger bit sizes for BGV/BFV would increase the computation time and communication size. For Elgamal, the impact of item bit length on computation time is minor and considered in Subsection D.4. Also, achieving server privacy for BGV/BFV would require, e.g., an additional *noise-flooding* or *OPRF* step which would increase the communication and computation costs. As such, this evaluation serves as an overview for the used schemes and does not provide a fair comparison between our ElGamal and BGV/BFV-based instantiations.

5.2.2 Parallelization

Figure 8 shows the improvements using a varying number of threads. We can observe that for 8 threads, the total running time decreases by up to $\approx 74\%$ (for $\lvert Y \rvert = 128$ and $\lvert X \rvert = 2^{20}$). We expect the minimal improvement for a larger number of server threads is due to the unchanging effort for the client and the client-server interaction.

$\lvert X \rvert$	128	512	2048	128	512	2048
$\lvert Y \rvert$		65536			1048576	
1	25.7	35.5	62.5	317.8	347.8	415.7
8	6.7	11.5	25.5	52.1	67.4	102.7
16	5.2	9.7	23.8	36.5	56.0	87.3
24	5.4	10.1	23.1	37.7	53.0	86.1

Table 8: Online computation costs (s) of our ElGamal-based PSI scheme for different numbers of server threads.

Overall, the figures show that, depending on the application and input set sizes, our protocols achieve practical performance, especially for small client set sizes. The communication for $\lvert Y \rvert = 32, k_1 = 3$, and $\lvert X \rvert = 2^{20}$ is less than 84 bits per server element. We expect that for an more unbalanced case, the bits per server element decreases, as asymptotically shown in Subsection 4.2. As we will discuss in Subsection D.6, the implementation offers potential for better ciphertext encodings and thus, smaller communication costs. A more in depth discussion of the benefits of this technique in comparison to other techniques as well as some suggestions for future work can be found in Appendix E.

References

[1] Marius A. Aardal and Diego F. Aranha. 2DT-GLS: Faster and exception-free scalar multiplication in the GLS254 binary curve. *IACR Cryptol. ePrint Arch.*, page 748, 2022.

[2] Mehmet Adalier and Antara Teknik. Efficient and secure elliptic curve cryptography implementation of curve p-256. In *Workshop on elliptic curve cryptography standards*, volume 66, pages 2014–2017, 2015.

[3] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR : Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, avril 2016:155–174, April 2016.

[4] Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 94–125, Cham, 2021. Springer International Publishing.

[5] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–computation trade-offs in PIR. pages 1811–1828, 2021.

[6] Ramiro Alvarez and Mehrdad Nojoumian. Comprehensive survey on privacy-preserving protocols for sealed-bid auctions. *Comput. Secur.*, 88, 2020.

[7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. Pir with compressed queries and amortized query processing. pages 962–979, 2018.

[8] Diego F. Aranha, Chuanwei Lin, Claudio Orlandi, and Mark Simkin. Laconic private set-intersection from pairings. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’22, pages 111–124, New York, NY, USA, November 2022. Association for Computing Machinery.

[9] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022.

[10] Saikrishna Badrinarayanan, Peihan Miao, and Tiancheng Xie. Updatable private set intersection. *Cryptology ePrint Archive*, 2021.

[11] Bar Ilan University Cryptography Research Group. Libscapi - the secure computation api.

[12] Aslı Bay, Zekeriya Erkin, Jaap-Henk Hoepman, Simona Samardjiska, and Jelle Vos. Practical multi-party private set intersection protocols. *IEEE Transactions on Information Forensics and Security*, 17:1–15, 2022.

[13] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[14] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe DM de Souza, Vinodh Gopal, et al. Intel HEXL (release 1.2). https://github.com/intel/hexl, 9 2021.

[15] Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. Fhe circuit privacy almost for free. Cryptology ePrint Archive, Paper 2016/381, 2016.

[16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, Lecture Notes in Computer Science, pages 337–367, Berlin, Heidelberg, 2015. Springer.

[17] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. Cryptology ePrint Archive, Paper 2012/078, 2012.

[18] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):13:1–13:36, July 2014.

[19] Dung Bui and Geoffroy Couteau. Private set intersection from pseudorandom correlation generators. Cryptology ePrint Archive, Paper 2022/334, 2022.

[20] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT ’99*, Lecture Notes in Computer Science, pages 402–414, Berlin, Heidelberg, 1999. Springer.

[21] Robit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. Parallel programming in openmp, 2001.

[22] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’18, pages 1223–1237, New York, NY, USA, October 2018. Association for Computing Machinery.

[23] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1243–1255, New York, NY, USA, 2017. Association for Computing Machinery.

[24] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, November 1998.

[25] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks*, Lecture Notes in Computer Science, pages 464–482, Cham, 2018. Springer International Publishing.

[26] Charles J Clopper and Egon S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[27] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication, 2021.

[28] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, Lecture Notes in Computer Science, pages 103–118, Berlin, Heidelberg, 1997. Springer.

[29] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 143–159, Berlin, Heidelberg, 2010. Springer.

[30] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 4:159–178, 2018.

[31] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1):47–68, June 2007.

[32] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. Advanced encryption standard (aes), 2001.

[33] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, Lecture Notes in Computer Science, pages 10–18, Berlin, Heidelberg, 1985. Springer.

[34] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, WPES'19, pages 14–25, New York, NY, USA, November 2019. Association for Computing Machinery.

[35] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012.

[36] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1):115–155, January 2016.

[37] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, Lecture Notes in Computer Science, pages 1–19, Berlin, Heidelberg, 2004. Springer.

[38] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, Lecture Notes in Computer Science, pages 395–425, Cham, 2021. Springer International Publishing.

[39] Robin Geelen and Frederik Vercauteren. Bootstrapping for bgv and bfv revisited. Cryptology ePrint Archive, Paper 2022/1363, 2022.

[40] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, May 2009. Association for Computing Machinery.

[41] Craig Gentry and Shai Halevi. Compressible fhe with applications to pir. Cryptology ePrint Archive, Paper 2019/733, 2019.

[42] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the aes circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 850–867, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[43] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32nd international*

conference on Automata, Languages and Programming, ICALP'05, pages 803–815, Berlin, Heidelberg, July 2005. Springer-Verlag.

[44] O Goldreich, S Micali, and A Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, 1987.

[45] Shay Gueron. Intel's new AES instructions for enhanced performance and security. In Orr Dunkelman, editor, *Fast Software Encryption*, Lecture Notes in Computer Science, pages 51–66, Berlin, Heidelberg, 2009. Springer.

[46] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. All the numbers are us: Large-scale abuse of contact discovery in mobile messengers. In *NDSS*, 2021.

[47] Darrel Hankerson, Scott Vanstone, and Alfred J. Menezes. *Guide to Elliptic Curve Cryptography*. Springer New York, January 2004.

[48] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Information Security and Cryptography. Springer, Berlin, Heidelberg, 2010.

[49] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, page 78–86, New York, NY, USA, 1999. Association for Computing Machinery.

[50] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for bgv and bfv. Cryptology ePrint Archive, Paper 2021/315, 2021.

[51] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 370–389. IEEE, 2020.

[52] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Paper 2017/738, 2017.

[53] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[54] Jonas Janneck, Anselme Tueno, Jörn Kußmaul, and Matthew Akram. Private computation on set intersection with sublinear communication. Cryptology ePrint Archive, Paper 2022/1137, 2022.

[55] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[56] Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 85–86, New York, NY, USA, May 2012. Association for Computing Machinery.

[57] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. Faster secure comparisons with offline phase for efficient private set intersection, 2022.

[58] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 608–639, Cham, 2021. Springer International Publishing.

[59] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proc. Priv. Enhancing Technol.*, 2017(4):177–197, 2017.

[60] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 818–829, New York, NY, USA, 2016. Association for Computing Machinery.

[61] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1257–1272, New York, NY, USA, October 2017. Association for Computing Machinery.

[62] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[63] Yang Li, Kee Siong Ng, and Michael Purcell. A tutorial introduction to lattice-based cryptography and homomorphic encryption, 2022.

[64] Jack P. K. Ma and Sherman S. M. Chow. Secure-computation-friendly private set intersection from oblivious compact graph evaluation. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, pages 1086–1097, New York, NY, USA, May 2022. Association for Computing Machinery.

[65] Moxie Marlinspike. Technology preview: Private contact discovery for signal, 2017.

[66] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134, April 1986. ISSN: 1540-7993.

[67] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 2292–2306, New York, NY, USA, November 2021. Association for Computing Machinery.

[68] Bodo Möller. Algorithms for multi-exponentiation. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 165–180, Berlin, Heidelberg, 2001. Springer.

[69] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, Lecture Notes in Computer Science, pages 678–708, Cham, 2021. Springer International Publishing.

[70] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms — ESA 2001*, Lecture Notes in Computer Science, pages 121–133, Berlin, Heidelberg, 2001. Springer.

[71] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, Lecture Notes in Computer Science, pages 223–238, Berlin, Heidelberg, 1999. Springer.

[72] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, Lecture Notes in Computer Science, pages 401–431, Cham, 2019. Springer International Publishing.

[73] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 739–767, Cham, 2020. Springer International Publishing.

[74] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, 2015.

[75] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, Lecture Notes in Computer Science, pages 122–153, Cham, 2019. Springer International Publishing.

[76] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–157. Springer, 2018.

[77] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 797–812, San Diego, CA, 2014. USENIX Association.

[78] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), 1 2018.

[79] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM*, 59(3):14:1–14:50, June 2012.

[80] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 859–870, 2018.

[81] Amanda C. D. Resende and Diego F. Aranha. Faster unbalanced private set intersection. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 203–221, Berlin, Heidelberg, 2018. Springer.

[82] Amanda C. D. Resende and Diego F. Aranha. Faster unbalanced private set intersection in the semi-honest setting. *Journal of Cryptographic Engineering*, 11(1):21–38, April 2021.

[83] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, Lecture Notes in Computer Science, pages 901–930, Cham, 2021. Springer International Publishing.

[84] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[85] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 1166–1181, New York, NY, USA, November 2021. Association for Computing Machinery.

[86] The OpenSSL project. Openssl — cryptography and ssl/tls toolkit.

[87] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 1556–1571. USENIX Association, 2019.

[88] Zhusheng Wang, Karim Banawan, and Sennur Ulukus. Private set intersection: A multi-message symmetric private information retrieval perspective. *IEEE Transactions on Information Theory*, 68(3):2001–2019, 2022.

[89] Christian Weinert. *Practical Private Set Intersection Protocols for Privacy-Preserving Applications*. PhD thesis, Technical University of Darmstadt, Germany, 2021.

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

A Protocol Details

Algorithm 1 shows the pseudocode for an PSI protocol instantiation using AHE and blocked Cuckoo hashing.

B Full Analysis

First, we will mention some details we have omitted for simplicity. The set sizes $|X|$ and $|Y|$ are disclosed to the parties before the computation. The decision to use multi-table or single-table Cuckoo hashing and the number of hash functions k_1 and k_2 are hyperparameters of the protocols. Depending on the set sizes and hyperparameters, the parties deduce parameters for the hashing families (e.g., l_1 and l_2). We assume that all our protocol constructions have an item-independent precomputation phase in which necessary cryptographic key material, hash function descriptions, and randomness are exchanged. Since we consider semi-honest behavior, we simply specify that the client chooses all necessary parameters and keys and sends them to the server. Emerging problems of parameter choices by malicious clients are out of scope of this work.

We have not specified the PSI protocol behavior if Cuckoo hashing fails. In contrast to Cuckoo hashing used in non-cryptographic algorithms [70], in the case of failures, we must not choose other hash functions and repeat the hashing. Repeating the hashing would implicitly make the hash functions item-dependent, which could leak information about used inputs and makes it impossible to simulate indistinguishable *views* in the proofs. Instead, we specify that if failures occur, the parties send a failure signal to the other party, immediately stop the protocol execution, and output $\emptyset$. However, we will assume hashing to fail with negligible probability in a statistical security parameter $\lambda \in \mathbb{N}$. Thus, the concrete behavior in the case of failures does not influence the (asymptotic) security as shown in the following proofs. For simplicity, we consider one security parameter $\kappa \in \mathbb{N}$ and assume $\kappa = \lambda$ to avoid dealing with an additional security parameter. In practice, variants to handle client hash failures might be interesting, like skipping and dropping items e that lead to failure in the hashing insertion step. Thus, the PSI output would become $(X \cap Y) \setminus \{e\}$. However, we remark that hashing parameters with non-negligible failure probability lead to an insecure PSI protocol.

We assume that the parameters of the used AHE scheme Π are chosen such that homomorphically evaluated ciphertexts decrypt to the correct result (as mentioned in Subsection 2.3). Again, the parameter selection for Π has to be item-independent and solely deduced from public parameters.

B.1 Security Proofs

[Correctness] Our PSI protocol π provides correctness, meaning that,

$$\{\text{output}^{\pi}(X,Y,\kappa)\}_{X,Y,\kappa} \overset{c}{\equiv} \{(\emptyset, X \cap Y)\}_{X,Y} \ .$$

Proof. Let E_1 be the event that hashing succeeds and $E_2 = \neg E_1$. For any fixed X, Y, and κ, let $\mathcal{D}$ be an PPT distinguisher with output in $\{0,1\}$, let $D_1 := \mathcal{D}(1^{\kappa}, \text{output}^{\pi}(X,Y,\kappa))$, $D_2 := \mathcal{D}(1^{\kappa}, (\emptyset, X \cap Y))$, and

$$\text{Adv}^{\pi}_{\texttt{corr}} := \Pr[D_1] - \Pr[D_2] := \Pr[D_1 = 1] - \Pr[D_2 = 1],$$

then,

$$\begin{aligned}\text{Adv}^{\pi}_{\texttt{corr}} &= \Pr[E_1]\cdot\Pr[D_1 \mid E_1] + \Pr[E_2]\cdot\Pr[D_1 \mid E_2] - \Pr[D_2] \\ &\le \Pr[D_1 \mid E_1] + \text{negl}(\kappa) - \Pr[D_2].\end{aligned}$$

It is left to show that, in the event of E_1, $\text{output}^{\pi}(X,Y,\kappa) = (\emptyset, X\cap Y)$ and thus, $\Pr[D_1 \mid E_1] = \Pr[D_2]$. Since the server outputs nothing, we are only interested in the client's output in the event of E_1. Let $H_1,\ldots,H_{k_1}$ and $H'_1,\ldots,H'_{k_2}$ be the used hash functions in π. Assume $e \in X\cap Y$, then $e \in Y$ and e is placed at one $i \in \{H_1(e),\ldots H_{k_1}(e)\}$ in the outer client hash table. Likewise, $e \in X$, thus, e is placed in all inner Cuckoo tables CT^i for $i \in \{H_1(e),\ldots H_{k_1}(e)\}$. The client and server use the same indices $J := H'_1(e),\ldots,H'_{k_2}(e)$ to create the encrypted index vectors and insert e into the inner Cuckoo hash table. Thus, the description of our construction in Subsection 3.3 shows that the AHE-based PIE outputs `True` to the client.

For $e \notin X\cap Y$, either $e \in Y \wedge e \notin X$ or $e \notin Y$. If $e \in Y \wedge e \notin X$, e is placed at a Cuckoo index i, but e is unequal to every server element and $e \neq \perp_S$, thus, the AHE-based PIE outputs `False` (as described in Subsection 3.3). If $e \notin Y$, either $e = \perp_C$ or the client does not place e in the Cuckoo table. If a dummy element $\perp_C$ is placed at a Cuckoo position i, since we assume that $\perp_C$ is unequal to the server dummy element $\perp_S$ and also no valid input element, the corresponding AHE-based PIE comparison outputs `False`. □

ElGamal-based Protocol Correctness of the ElGamal-based protocol with the improvements of simultaneous multi-exponentiation and the extended precomputation follows from subsubsection 4.1.1 and the constructions in Subsection 3.4. Since ElGamal provides ciphertext freshness and soundness, we can do arbitrarily many homomorphic evaluations without worrying about the correctness of the decryption. However, the cardinality of the underlying group $|\mathbb{G}| = p$ has to be larger than 2^{ρ} to encode all possible elements $e \in \{0,1\}^{\rho}$ as exponents $\mathsf{m} \in \mathbb{Z}_p$.

BGV/BFV-based Protocol The correctness of BGV/BFV-based protocol also directly follows from subsubsection 4.1.1 and the description in Subsection 3.5. As mentioned in subsubsection 2.3.3, leveled FHE schemes allow you to specify the number of homomorphic operations that can be performed such that an operated ciphertext can still be correctly decrypted. We will not go into detail about how to choose the correct parameters for BGV and BFV but refer to the rich literature [9; 17; 18; 35; 58]. Remark that the number of needed homomorphic scalar multiplications and additions can be deduced from the public parameters k_1,k_2,l_1,l_2.

[Client Privacy] Let Π be an IND-CPA secure AHE scheme, then, our generic PSI protocol π instantiated with Π (as shown in Algorithm 1) provides *client privacy*, i.e., there exist a *PPT* algorithm Sim_S, such that

$$\{view^{\pi}_S(X,Y,\kappa)\}_{X,Y,\kappa} \overset{\text{c}}{\equiv} \{\text{Sim}_S(1^{\kappa},X,\emptyset,|Y|)\}_{X,Y,\kappa}\,.$$

Proof. We construct the simulator Sim_S as follows:

- Uniformly and randomly choose r^S (as simulation of the servers random tape).
- Generate all parameters and keys based on the hyper parameters and set sizes $|X|,|Y|$ (as in π).
- For every possible outer Cuckoo table index i, generate a simulated encrypted element $\text{Enc}(r_0)$ and for every $q \in \{1,\ldots,k_2\}$ generate an simulated encrypted index vector $\text{Enc}(r_1),\ldots\text{Enc}(r_{l_2})$ with $r_i \leftarrow_\$ \mathbb{Z}_p$ for all $i \in \{0,\ldots,l_2\}$.
- Encode the generated parameters, keys, and encrypted values as bit-string $\mathsf{m_1}^S$.
- Output $\left(X, r^S, \mathsf{m_1}^S\right)$.

Remark, $view^{\pi}_S$ contains the hashing and encryption parameters and keys, as well as the encrypted client Cuckoo table entries and corresponding encrypted index vectors. The input set, random tape, parameters, and keys of $view^{\pi}_S$ and Sim_S are identically distributed. If the scheme does not provide client privacy, a PPT $\mathcal{D}$ exists that successfully distinguishes a set of encrypted random values from another set of encrypted concrete (potentially not uniformly distributed) values. Since we assume Π to be IND-CPA secure, such a distinguisher cannot exist, which concludes the proof by contradiction. □

We assume ElGamal, BGV, and BFV to be IND-CPA secure. Thus, the client privacy of the concrete constructions directly follows from subsubsection 4.1.2.

Precomputation The protocol with (extended) precomputation yields a slightly different server's view, but the proof is similar. A simulator Sim'_S for the server's view $view^{\pi}_S(X,Y,\kappa)$ for the PSI protocol with (extended) precomputation can be constructed as follows. Let Sim_S be the simulator with output $\left(X, r^S, \mathsf{m_1}^S\right)$ as described in the proof of subsubsection 4.1.2. Sim'_S modifies $\mathsf{m_1}^S$ and adds a random bit $b \leftarrow_\$ \{0,1\}$ for each outer Cuckoo table index, each $q \in \{1,\ldots,k_2\}$, and each possible index $j \in \{1,\ldots,l_2\}$. Assume π uses a PRF to generate the random values, in the proof, we first replace the PRF generated values by values drawn uniformly and randomly. Now, again, the security follows from the IND-CPA security of the underlying AHE scheme as in the proof of subsubsection 4.1.2.

[Server Privacy] Assume simulators $\text{Sim}'_0, \text{Sim}'_1$ (as described in subsubsection 4.1.3) exist for the *PSI* protocol π (as shown in Algorithm 1), then, π provides *client privacy*, i.e., a *PPT* algorithm Sim_C exists, such that

$$\{\text{view}^{\pi}_C(X,Y,\kappa)\}_{X,Y,\kappa} \overset{\text{c}}{\equiv} \{\text{Sim}_C(1^{\kappa},Y,X\cap Y,|X|)\}_{X,Y,\kappa}\,.$$

Proof. We construct the simulator Sim_C as follows:

- Choose r^C (as simulation of the servers random tape) uniformly at random.
- Generate all parameters and keys based on the hyper-parameters and set sizes $|X|,|Y|$ (as in π). Encode the parameters as $\texttt{params} \in \{0,1\}^*$.
- Create a Cuckoo hash table CT with the items Y using the deduced parameters $\texttt{params}$.
- For every possible Cuckoo table index i, use $\mathrm{Sim}'_0(\texttt{params})$ to generate a list L of simulated encrypted random results with the same number of ciphertexts as L in π. If $CT[i] \in X \cap Y$, for one $l \in L$, use $\mathrm{Sim}'_1(\texttt{params})$ instead of $\mathrm{Sim}'_0(\texttt{params})$.
- Shuffle each L (independently and) uniformly at random.
- Concatenate the encode parameters $\texttt{params}$ and the shuffled list L as bit-string $\mathsf{m_1}^C$.
- Output $\left(Y, r^C, \mathsf{m_1}^C\right)$.

As in subsubsection 4.1.2, the simulated parameters, random tape, and input are identically distributed as in view_C^π. By assumption, for every $b \in \{0,1\}$, the output of Sim'_b is indistinguishable from the server result c_f. Thus, by shuffling at random and since c_f decrypts to 0 at most once per Cuckoo index i, the simulated output and the view of π are computationally indistinguishable. □

ElGamal-based Protocol For the ElGamal-based protocol, we do not need the general assumption of simulating server results c_f. Server privacy can be achieved by rerandomizing $c_f \leftarrow c_f \boxplus \mathrm{Enc}(0)$ (as mentioned in Subsection 3.4). The rerandomized ciphertexts $\mathrm{Enc}(e;r)$ are distributed like fresh ciphertexts but for an integer r chosen uniformly at random by the server. We can thus instantiate $\mathrm{Sim}'_0(\texttt{params}) := \mathrm{Enc}(r)$ for $r \leftarrow_\$ \mathbb{Z}_p$ and $\mathrm{Sim}'_1(\texttt{params}) := \mathrm{Enc}(0)$. Remark the server privacy for our ElGamal-based protocol does not require computational assumptions and, thus, provides perfect indistinguishability.

BGV/BFV-based Protocol Server privacy for the BGV/BFV-based protocol requires different techniques than for the ElGamal-based construction. Since homomorphic computations increase the error term (as mentioned in subsubsection 2.3.3), fresh encryptions $\mathrm{Enc}(0)$ are not indistinguishable from an c_f that decrypts to 0. We can solve this problem by establishing circuit privacy [15]. However, maybe more efficient approaches are possible if the requirement to hide the FHE circuit structure is omitted. For our later evaluation, we assume an additional OPRF masking step as in [27], which also allows simulating the server results without the input of $X \setminus Y$.

B.2 Complexity Proofs

[Complexity] The complexities of our PSI protocol π can be summarized as follows:

- The client has a computation complexity of

$$O\left(\frac{1+\sigma}{\sqrt{\sigma}} \cdot \sqrt{\cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \tag{9}$$

- The server has a computation complexity of

$$O(\beta_1 \cdot \beta_2 \cdot |Y| \cdot m_b + \beta_2 \cdot k_1 \cdot |X|) \tag{10}$$

- The communication complexity is

$$O\left(\gamma \cdot \frac{1+\sigma}{\sqrt{\sigma}} \cdot \sqrt{\cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \tag{11}$$

Computation For each outer Cuckoo table index i, the client encrypts k_2 index vectors and the item (or dummy element) at the Cuckoo table position i. Each encrypted index vector EIV contains l_2 elements. Thus, the client encrypts $O(t_1 \cdot (1 + k_2 \cdot l_2))$ elements.

The server multiplies each entry of EIV with the corresponding inner δ-block Cuckoo hashing table entry for each bin $d \in \{1,\dots,\delta\}$, which leads to $O(t_1 \cdot k_2 \cdot l_2 \cdot \delta)$ homomorphic scalar multiplications. Remark, if we set $\perp_S = 0$, the server does not need to perform a homomorphic scalar multiplication for dummy elements.

For each $d \in \{1,\dots,\delta\}$ and $k' \in \{1,\dots,k_2\}$, the result is (homomorphically) summed up, subtracted, and scalar multiplied. We ignore the complexity of the shuffle step, which can be performed very efficiently (and also use precomputed random permutations). Remark, the homomorphic subtraction can be avoided if the client instead sends $Enc(-e)$ for each element (or dummy value) e at an outer Cuckoo table position. Thus, overall, the server needs to perform $O(t_1 \cdot k_2 \cdot \delta \cdot l_2)$ homomorphic scalar multiplications and additions.

Considering parameters β_1, β_2, this leads to $t_1 = \beta_1 \cdot |Y|$ and $t_2 \in O\left(\beta_2 \cdot \left(\frac{k_1 \cdot |X|}{t_1} + m_b\right)\right)$ for $m_b \in \mathbb{N}$. Thus, the server computational complexity simplifies to

$$\begin{aligned} &O\left(\beta_1 \cdot |Y| \cdot \beta_2 \cdot \left(\frac{k_1 \cdot |X|}{\beta_1 \cdot |Y|} + m_b\right)\right) \\ &= O(\beta_1 \cdot \beta_2 \cdot |Y| \cdot m_b + \beta_2 \cdot k_1 \cdot |X|). \end{aligned} \tag{12}$$

Remark, $|Y| \ll |X|$ and m_b is small (as shown in Subsection D.1). Thus, in practice, $k_1 \cdot |X|$ is the dominating factor. For simplicity and as in related work [30; 78], in the following, we assume $m_b \in \mathbb{N}$ to be of constant size.

For each outer Cuckoo table index i, the client receives $O(k_2 \cdot \delta)$ ciphertexts c_f. If the client has placed a dummy element at index i, the client can skip the decryption step. Also, if a server result c_f has decrypted to 0, the client does not need to decrypt any other c_f for the same outer Cuckoo table entry anymore. However, this might introduce side-channel leakage if an attacker can observe the computation time of the client (as discussed in Subsection D.6). The expected computational complexity for these decryption improvements is omitted in our big O notation.

Let $t_2 = \beta_2 \cdot \left(\frac{k_1 \cdot |X|}{t_1} + m_b\right)$ with constant m_b, we can adjust $l_2 = \left\lceil \sqrt{\frac{t_2 \cdot \sigma}{k_2}} \right\rceil$ and $\delta = \left\lceil \sqrt{\frac{t_2}{\sigma \cdot k_2}} \right\rceil$ to achieve sublinear complexity for a *skewness* $\sigma \approx \frac{l_2}{\delta}$, as described in Subsection 3.3. Thus, overall, if we omit m_b, the client encrypts

$$O(t_1 \cdot k_2 \cdot l_2) = O\left(\beta_1 \cdot |Y| \cdot k_2 \cdot \left\lceil \sqrt{\frac{\beta_2 \cdot k_1 \cdot |X| \cdot \sigma}{\beta_1 \cdot |Y| \cdot k_2}} \right\rceil\right) \quad (13)$$

elements. For l_2 larger than 1, which follows for $|Y| \ll |X|$ (and adequate k_2 and σ), we can simplify the term as

$$O\left(\beta_1 \cdot |Y| \cdot k_2 \cdot \sqrt{\frac{\beta_2 \cdot k_1 \cdot |X| \cdot \sigma}{\beta_1 \cdot |Y| \cdot k_2}}\right) = O\left(\sqrt{\sigma \cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \quad (14)$$

Likewise, for δ larger than 1, we can do the same simplifications, which shows that the client performs

$$O\left(\sqrt{\frac{1}{\sigma} \cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \quad (15)$$

decryptions. Thus, we have shown the claim of sublinear computation complexity (in $|X|$) for the client (assuming $|Y| \ll |X|$).

Communication The communication complexities of the client and server can be deduced from the number of performed encryptions and decryptions of the client. In the online phase, the client sends, and the server receives

$$O\left(\gamma \cdot \sqrt{\sigma \cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \quad (16)$$

bits. Likewise, the server sends, and the client receives

$$O\left(\gamma \cdot \sqrt{\frac{1}{\sigma} \cdot \beta_1 \cdot |Y| \cdot k_2 \cdot \beta_2 \cdot k_1 \cdot |X|}\right) \quad (17)$$

bits. Thus, we have shown the claim of sublinear communication complexity (in $|X|$) of our protocol (assuming $|Y| \ll |X|$).

Precomputation The precomputation extension generally increases the computation complexity of the server and client. However, the computationally expensive generation of the encryption index vectors by the client can be performed in an item-independent precomputation phase. The online communication can be reduced since instead of γ bits for the entries in EIV, only a single bit is sent in the online phase, as evaluated in Figure 17.

ElGamal-based Protocol In practice, the number of group operations $\odot$ and, thus, the computational complexity can be reduced by using simultaneous multi-exponentiation. More detailed complexity analyzes of simultaneous multi-exponentiation, also denoted as *simultaneous multiple point multiplications*, can be found in the literature [47; 68]. However, the computational complexity of simultaneous multi-exponentiation depends on the distribution of the used exponents and, thus, in the case of PSI, the server elements X. This might introduce side-channel leakage, as discussed in Subsection D.6.

The extended precomputation for the ElGamal-based protocol also increases the computational complexity of the server and client. In comparison to the simple precomputation, as described in Subsection 3.3, the extended precomputation reduces the number of homomorphic scalar multiplications in the online phase by an additional offline phase (with available server input). However, this comes at the cost of computing and storing twice as many homomorphic scalar multiplications on the server.

BGV/BFV-based Protocol To hide the server's inner Cuckoo table position in the BGV/BFV-based protocol with batched computation, additional $k_2 \cdot \delta$ homomorphic ciphertext-ciphertext multiplications are computed, as described in subsubsection 5.2.1. This increases the server computation complexity but decreases the number of client decryptions. If $u \in \mathbb{N}$ plaintexts can be packed into the same ciphertext, the batched computation reduces the server and client complexities by up to factor u (as also discussed in subsubsection 5.2.1). However, for larger u, in general, also γ and the complexity of homomorphic operations increase.

C Implementation Details

We have separated the execution of our protocols into *precomputation*, *offline*, and *online* phases as follows.

Precomputation Phase In the precomputation phase, no inputs are available to the parties, but they can exchange item-independent materials such as randomness, keys, or hash functions. For our PSI protocol, in the precomputation phase, the client sends the public encryption keys and the hash functions to the server. Using the precomputation extension (as described in Subsection 3.3 and Subsection 3.4), the client also sends the encrypted random vectors ERV. In this phase, the server receives the AHE public key and the hash functions to the server. For the ElGamal-based scheme, the server computes encryptions of zero for the later rerandomization of ElGamal ciphertexts (as described in Subsection 3.4).

Offline Phase In the offline phase, the parties cannot communicate but make computations on their inputs and received data from the precomputation phase. In our implementation, the offline phase includes creating the outer Cuckoo hash table and encrypting each outer Cuckoo hash table entry and all index vectors EIV.

Online Phase In the online phase, the client sends the encrypted outer Cuckoo table entries and EIVs to the server. The server computes the encrypted results c_f (as described in Algorithm 1) and sends them back to the client.

Parallelization The practical performance of our PSI protocol implementation is improved through parallel computations. We implement parallelization only for the server in accordance with our setting of a client with limited computational resources. However, parallelization for the client could be added. We implement multi-threading for the nested Cuckoo hashing creation on the server by using OpenMP [21]. Likewise, OpenMP can be used within the OpenFHE framework to parallelize the computation of the BGV/BFV-based protocol implementation. For our ElGamal-based version, we have implemented a more fine-granular multi-threading approach, described as follows. The server uses one main thread to read the data from the client, distributes the homomorphic computations workloads to other threads, gathers the results c_f, and sends them back to the client. As such, with more than one thread, the server can perform homomorphic computations before the client has sent all encrypted items. Likewise, depending on the server thread scheduling, the client can already decrypt server results before the server has performed all homomorphic operations.

D Further Evaluations and Details

D.1 Parameters

For the evaluation, we use different (not equidistant) client set sizes $|Y| \in \{32, 128, 512, 1024, 2048, 4096\}$ and server sets of size $|X|$ of $2^{16} = 65536$ and $2^{20} = 1048576$ filled with random elements of fixed bit length ρ. Unless otherwise specified, we use $\rho = 128$ bits for the ElGamal-based protocol and $\rho = 32$ bits for the BGV/BFV-based version considered in subsubsection 5.2.1. For a practical evaluation, we first have to select many parameters accordingly to our assumption for hashing failures and security levels. Selecting the parameters for our hashing scheme and the BGV/BFV-based protocol is highly non-trivial. We use the slack factors formulas interpolated by Demmler at al. [30] for β_1, where $t_1 := \beta_1 \cdot k_1 \cdot l_1$ are the number of possible entries in the client's Cuckoo hash table. Remark, as in the evaluation of PIR-PSI [30], we choose β_1 to reach a client Cuckoo table failure rate of $\leq 2^{-20}$. However, we could decrease the failure probability, e.g., to 2^{-40} with an increase of β_1 logarithmically in the failure probability [30]. For our BGV/BGV-based protocol, we can even show that larger β_1s lead to better performance results for small client set sizes due to the batched computation (as discussed in subsubsection 5.2.1). However, for fixed BGV/BFV parameters, the outer Cuckoo table size t_1 and thus, β_1 can only be increased to a limited extent.

For the server's nested Cuckoo hash table, we first use the empirically interpolated formula of Demmler et al. [30] to calculate a needed maximum bin size $max_b = \frac{k_2 \cdot |X|}{t_1} + m_b$ such that the simple hashing into bins, with maximum bin size max_b, fails with probability $\leq 2^{-40}$. To determine the size of the Cuckoo hash tables t_2, max_b is multiplied with another slack factor $\beta_2 = 1.1$ (according to Subsection D.7) and the resulting value is rounded up. The used parameter combinations are given in Table 9.

For the inner Cuckoo hash tables (inside the nested Cuckoo hash table), we use large $\delta = \left\lceil \sqrt{\frac{t_2}{\sigma \cdot k_2}} \right\rceil$ and thus, $l_2 = \left\lceil \sqrt{\frac{t_2 \cdot \sigma}{k_2}} \right\rceil$. We set $\sigma = 1$, if not stated otherwise (like in Subsection D.3). For our evaluations, we always use $k_2 = 2$ multi-table Cuckoo hashing without a stash. Note, we have not observed any hashing failures during the evaluation when using these parameters.

For our evaluations with exponential ElGamal, we use the P-256 elliptic curve [2] which offers an expected security level of 128 bits. The parameter selection for the BFV/BGV schemes uses OpenFHE to set the security level to 128 bits and to adjust the number of packed ciphertext for the batched computation accordingly [9]. Further, we use other OpenFHE default parameters [9]. We have not activated hardware acceleration for the lattice-based FHE computations which can be added for OpenFHE using Intel's *homomorphic encryption acceleration library* (HEXL) [14].

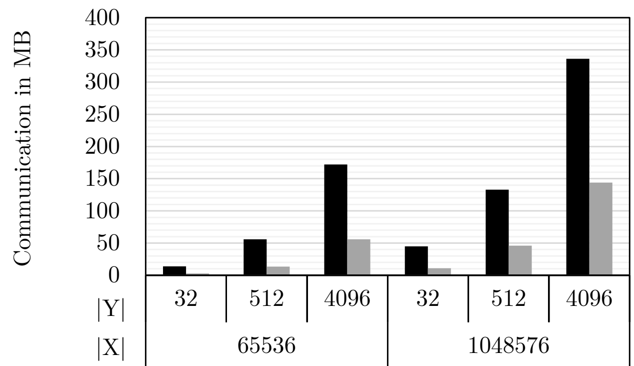
D.2 Different Number of Outer Cuckoo Hash Functions

We investigate the influence of parameters $k_1 \in \{2, 3\}$ on the performance of our ElGamal-based protocol. The differences of Cuckoo hashing for $k_1 \in \{2, 3\}$ has also been subject to analyses of other PSI protocols like PIR-PSI [30]. Subsection 4.2 has shown that the computation and communication costs (theoretically) increase with an increasing k_1. However, as also mentioned in Section 4, the parameters cannot be chosen independently. For smaller k_1, we need to increase β_1 to reach the same Cuckoo hashing failure probability [30]. Figure 10 shows that the communication costs for $k_1 = 2$ is larger than $k_1 = 3$ (for all other parameter combinations).

For most parameter combinations, Figure 11 shows that the online running time for $k_1 = 3$ is smaller (than for $k_1 = 2$). This observation is counter-intuitive but can be explained with the higher slack factors β_1 for $k_1 = 2$ (given in Table 9). Remark, that in comparison to $k_1 = 2$, for $k_1 = 3$, the server needs to insert almost 50% more items into the nested Cuckoo hash table. We expect that for larger server set sizes (and other

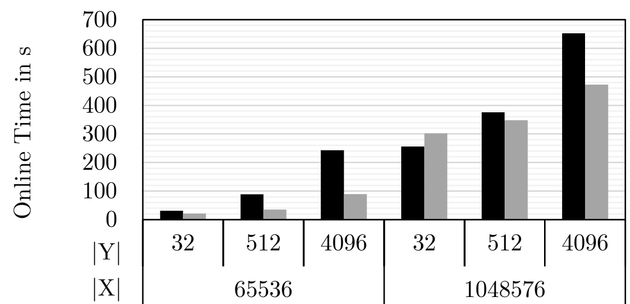
Table 9: Hashing parameters used for our evaluations. Cuckoo hashing factor β_1 is chosen according to the interpolated formulas by Demmler at al. [30] and $\beta_2 = 1.1$, as argued in Subsection D.7.

$\|Y\|$	$\|X\|$	k_1	β_1	l_1	max_b	t_2
32	65536	2	27.62	442	256	282
128	65536	2	18.15	1162	128	141
512	65536	2	11.92	3053	70	77
1024	65536	2	9.66	4949	54	60
2048	65536	2	7.83	8022	42	47
4096	65536	2	6.35	13004	34	38
32	1048576	2	27.62	442	2770	3047
128	1048576	2	18.15	1162	1156	1272
512	1048576	2	11.92	3053	507	558
1024	1048576	2	9.66	4949	344	379
2048	1048576	2	7.83	8022	237	261
4096	1048576	2	6.35	13004	166	183
32	65536	3	1.26	14	5201	5722
128	65536	3	1.27	55	1468	1615
512	65536	3	1.29	220	445	490
1024	65536	3	1.30	443	256	282
2048	65536	3	1.30	890	154	170
4096	65536	3	1.31	1791	97	107
32	1048576	3	1.26	14	76950	84645
128	1048576	3	1.27	55	20139	22153
512	1048576	3	1.29	220	5322	5855
1024	1048576	3	1.30	443	2766	3043
2048	1048576	3	1.30	890	1466	1613
4096	1048576	3	1.31	1791	795	875



k = 2	13.7	55.8	172.1	44.9	132.8	336.1
k = 3	2.9	13.5	55.9	10.9	46.0	143.9

Figure 10: PSI communication costs for $k = k_1 = 2$ and $k = k_1 = 3$ using ElGamal.



k = 2	30.9	88.3	242.5	255.7	375.9	652.0
k = 3	21.7	35.5	89.6	302.2	347.8	472.6

Figure 11: PSI online computation costs for $k = k_1 = 2$ and $k = k_1 = 3$ using ElGamal.

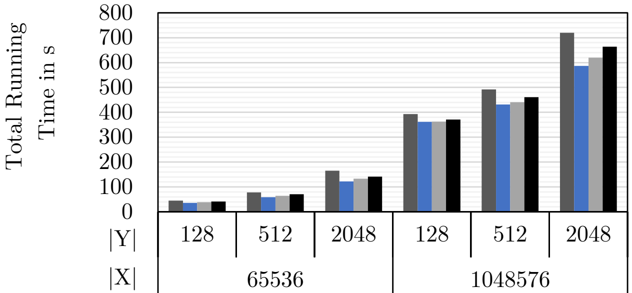
parameters held constant), using $k_1 = 2$ will at some point offer better performance results.

D.3 Different Cuckoo Table Skewness

In the rest of the evaluation, we will only consider $k_1 = 3$. Recall, for a blocked (inner) Cuckoo table, the parameter σ specifies the ratio between the number of hash indices l_2 and the size of the bins δ. The theoretical complexity analysis in Section 4 shows that a parameter $\sigma = 1$ provides the best asymptotic communication and computation complexity. We want to test different values $\sigma \in \{0.5, 1, 2, 3\}$ for the performance of the practical implementation.

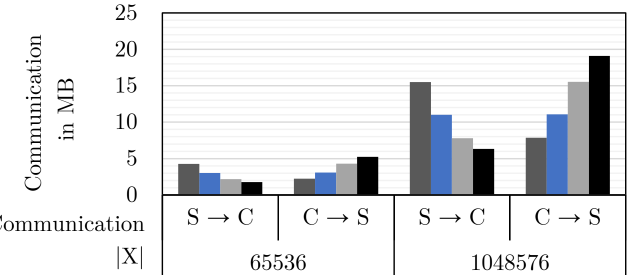
Figure 12 shows that $\sigma = 1$ also provides the best practical performance over (almost) all parameter combinations. If we look at the online and offline phases separately, the picture is somewhat different. Remember, for large values of σ, the size of the encrypted index vector EIV increases and thus also the amount of elements the client has to encrypt. At the same time, the number of server results c_f, and thus, the number of decryptions required by the client decreases. Larger values of σ allow more computational effort to be shifted to the offline phase. The relationship becomes clear in Figure 13. The data the client has to encrypt corresponds to the outgoing bytes, whereas the server results correspond to the incoming bytes. Figure 13 also shows that the total communication is minimal for $\sigma = \frac{l_2}{\delta} = 1$. E.g., for $|X| = 2^{20}$, the communication for $\sigma = 1$ is $22.10 = 11.02 + 11.08$, while for the other $\sigma \in \{0.5, 2, 3\}$, the total communication is 23.35 or 25.42.

For $\sigma = 1$, Figure 12 also shows that the total running time of the ElGamal-based scheme is only around 10% – 25% higher (for $|X| = 2^{20}$) than the online time (compared to Figure 11). Thus, the Cuckoo hashing and encryption of the client is a minor part of the total computation costs. Remark for $\sigma = 1$, the client needs to check whether $\mathrm{Dec}(c_f) = 0$ for as many ciphertexts as the client sends as encrypted index vectors EIV. If we assume the comparison $\mathrm{Dec}(c_f) = 0$ to be as efficient as encryption, we can underpin the claim that our protocol is suitable for clients with limited resources.



	|X|	65536			1048576		
	|Y|	128	512	2048	128	512	2048
Skewness	0.5	44.6	77.8	165.2	393.3	492.3	719.4
Skewness	1	36.3	58.8	122.5	361.7	431.2	586.8
Skewness	2	38.5	64.5	132.8	363.0	441.1	619.8
Skewness	3	41.2	70.5	141.7	371.2	460.8	664.3

Figure 12: Total PSI computation costs for a varying skewness $\sigma \in \{0.5, 1, 2, 3\}$ using ElGamal with a client set size $|Y| = 128$.



	|X|	65536		1048576	
	Communication	S → C	C → S	S → C	C → S
Skewness	0.5	4.26	2.24	15.50	7.85
Skewness	1	3.02	3.07	11.02	11.08
Skewness	2	2.18	4.32	7.80	15.55
Skewness	3	1.77	5.25	6.34	19.08

Figure 13: PSI communication costs for a varying skewness $\sigma \in \{0.5, 1, 2, 3\}$ using ElGamal with a client set size $|Y| = 128$.

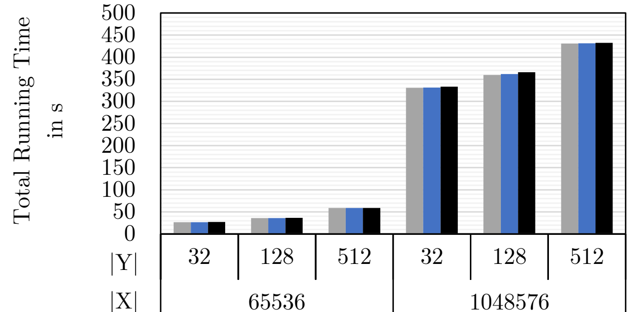
D.4 Item Lengths

In this subsection, we evaluate the influence of the items bit-length $\rho \in \{32, 128, 255\}$ on the performance of the exponential ElGamal-based protocol. To retain a security level of 128 bits for the evaluation, we use the P-256 elliptic curve also for small ρ, e.g., $\rho = 32$. Naïvely, for $\rho > 256$, we would require using elliptic curves with more elements. However, we could use a collision-resistant hash function to compute the PSI protocol on hashed fingerprints with a negligible collision probability in the resulting bit-length [78].

Figure 14 shows that the computational costs are almost independent of the used input bit-lengths ρ. However, we can clearly observe that for all parameter combinations, a larger ρ leads to a slightly higher running time. From a performance viewpoint, the running time differences are insignificant. However, different running times for varying bit-length indicate side-channel leakage, as we will discuss in Subsection D.6. Remark, informally, since we use the P-256 elliptic curve for all different ρ, and all sent communication data is either item-independent or encrypted, the communication costs are independent of ρ.

D.5 Precomputation

In the following, we will analyze the performance of our extended precomputation variation for our exponential ElGamal-based PSI protocol (as described in Subsection 3.4). Remark, the client protocol is the same for the extended precomputation and simple precomputation described in Subsection 3.3). Likewise, the communication costs are identical. To avoid long evaluations and allow testing more parameter combinations, we have used 24 server threads for the following protocol executions.

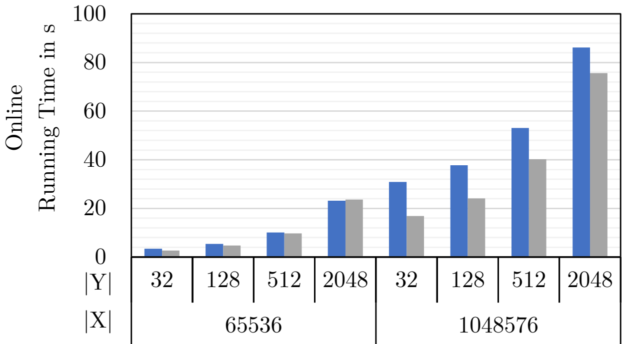


	|X|	65536			1048576		
	|Y|	32	128	512	32	128	512
Bit-length	32	26.8	36.2	58.7	331.0	359.8	431.0
Bit-length	128	26.9	36.3	58.8	331.2	361.7	431.2
Bit-length	255	27.0	36.3	58.8	333.1	366.1	432.2

Figure 14: Total computation costs of our ElGamal-based PSI scheme for different item bit-length ρ.

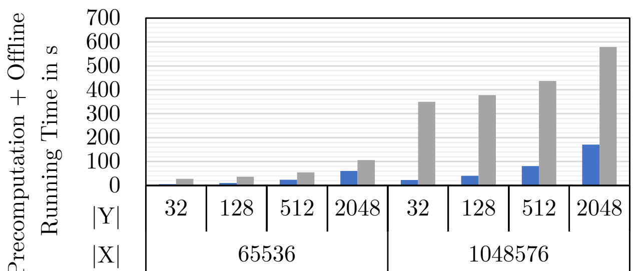
In Figure 15, we can indeed show that (at least for larger server set sizes) the online running time decreases when using the extended precomputation variant. However, the computation costs for the precomputation + offline phase highly increase, as shown in Figure 16. Especially for $|Y| \ll |X|$, the precomputation + offline phase takes up to ≈ 15 times as long as for the standard exponential ElGamal-based protocol. Remark, in the offline phase, the extended precomputation variant can utilize all 24 threads on the server. In our standard exponential ElGamal-based protocol, the computational effort of the precomputation and offline phase lies almost completely on the single-threaded client.

In comparison to the standard variant, Figure 17 shows that the online communication of the extended precomputation variant is almost halved. In more detail, Figure 17 shows



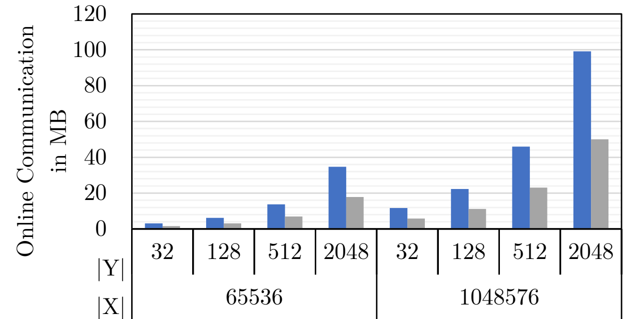
|X|	65536				1048576			
|Y|	32	128	512	2048	32	128	512	2048
Standard	3.5	5.4	10.1	23.1	30.8	37.7	53.0	86.1
Precomp	2.7	4.7	9.8	23.6	16.9	24.1	40.2	75.7

Figure 15: Online computation costs of our extended precomputation and standard ElGamal-based PSI schemes.



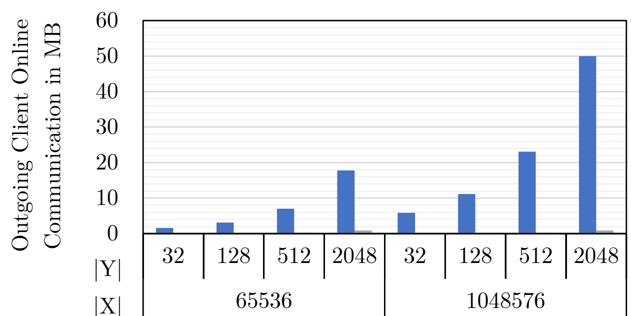
|X|	65536				1048576			
|Y|	32	128	512	2048	32	128	512	2048
Standard	5	11	24	60	23	41	81	170
Precomp	28	36	54	106	350	378	437	579

Figure 16: Precomputation + offline computation costs of our extended precomputation and standard ElGamal-based PSI schemes.



|X|	65536				1048576			
|Y|	32	128	512	2048	32	128	512	2048
Standard	3.1	6.2	13.8	34.7	11.7	22.3	46.0	99.1
Precomp	1.6	3.1	7.0	17.8	5.9	11.2	23.1	50.0

Figure 17: Online communication costs of our extended precomputation and standard ElGamal-based PSI schemes.



|X|	65536				1048576			
|Y|	32	128	512	2048	32	128	512	2048
Standard	1.56	3.12	6.99	17.79	5.85	11.17	23.08	49.98
Precomp	0.02	0.05	0.22	0.87	0.02	0.06	0.22	0.88

Figure 18: Outgoing online communication costs for the client using our extended precomputation scheme.

that especially the outgoing client communication is reduced by a factor ≈ 290 (for $|Y| = 32$ and $|X| = 2^{20}$). Since we use $\sigma = 1$ for these evaluations, for the standard exponential ElGamal-based variant, the amount of sent and received bytes by the client are almost equal (as shown in Figure 13). Thus by increasing σ, we could reduce the online communication of the extended precomputation variant. However, the total communication costs, including the precomputation phase, are always higher when using the extended precomputation in comparison to the standard variant.

D.6 Practical Aspects

More and more PSI protocols are actually implemented and deployed [51; 89]. We will therefore discuss some aspects that are relevant in practice.

Practical Security A practical security problem is *side-channel leakage*, which is not considered in our security model. If the running times for certain input item distributions differ, an adversary might gain additional information about the parties' inputs. The (non-constant) multiplications of plaintext with encrypted data are prone to introduce side-channel leakage since, generally, larger plaintexts increase the computation time. We remark that side-channel leakage is an actual problem of our implementations (as indicated in Figure 14). However, in practice, the running time can be randomly increased to obfuscate the exact value. A more secure approach could avoid timing side-channels by using randomized input elements by an additional OPRF masking step.

Implementation Performance Our current exponential ElGamal implementation supports all elliptic curves offered by the OpenSSL ECC interface [86]. However, other non-supported elliptic curves like Curve25519 [13] or GLS254 [1; 8] provide faster group operations and might improve practical performance. The libscapi library provides a nice interface to access different PKE schemes. However, when using ECC-based ElGamal, using libscapi comes at the cost of many datatype conversions during the computation. An implementation that directly links to and uses the same datatypes as OpenSSL [86] could, thus, improve the performance of the ElGamal-based implementation. Likewise, libscapi's encoding of ECC-based ElGamal ciphertexts could be improved by using elliptic curve point compression [55] or, at least, binary encoding of a curve point's $(x,y) \in \mathbb{Z}_p^2$ coordinates. With minor implementation adjustments, we expect to decrease the communication overhead by at least 50%.

Server Updates For applications like contact discovery [30], it is desirable to support updates of the input sets and compute the updated intersection without executing the whole PSI protocol again. PSI protocols that efficiently support updates have been considered by Kiss et al. [59] and Badrinarayanan et al. [10]. The complexity of an update step should thereby only increase with the number of additions and deletions, but not the set size itself. For our ElGamal-based scheme, we can use the same server update approach as presented by Janneck et al. [54].

D.7 Hashing Failure Evaluation

As mentioned, theoretical analyses of the hashing failure probabilities for Cuckoo hashing and especially blocked Cuckoo hashing are missing and left for future work. However, similar to other works on hashing-based PSI [30; 76; 78], we empirically measure the failure probabilities of the used δ-block Cuckoo hashing as a foundation for the later performed evaluation. We will use the interpolated formulas of Demmler et al. [30] for the adjustment of the slack factor β_1 of the outer Cuckoo hash table. For the nested Cuckoo hash table and thus, the δ-block inner Cuckoo hashing we evaluate the failure probability with $k = k_2 = 2$ hash functions using the random-walk reinsertion strategy with tabulation hashing [31; 79].

For each parameter combination, we perform at least 10000 hashing attempts. The number of hashing attempts it not high enough to empirically validate failure probabilities of, e.g., 2^{-40}, as also discussed in related works [30; 38; 78]. For a failure probability of 2^{-40}, we need expected 2^{40} hashing attempts to observe on hash failure which is infeasible due to limited computational resources. However, to strengthen the validity of our empirical evaluation, we provide 99.99% confidence intervals [26] for the estimated failure probabilities. As in related work [30; 78], we use an input set chosen uniformly at random. In the following evaluation, we call β the *ratio between the table size and the number of elements*. Since we want to adjust $\delta = O(\sqrt{|X|/|Y|})$ for sublinear communication, we want to use large δ (assuming $|Y| \ll |X|$). Figure 19 shows that with already $\delta = 8$, the failure probability is so low that for $\beta > 1.008$, we observe no failure. For the following evaluation, we will use δ-block Cuckoo hashing with $\beta_2 = 1.1$, which is far higher than the interpolated $\beta \approx 1.012$ for a failure probability of $\leq 2^{-40}$ However, from a theoretical point of view, this argumentation is questionable and requires a closer analysis in future work. Remark, that we do not use Cuckoo table stashes in our evaluation, since blocked Cuckoo hashing already decreases the failure probability sufficiently.

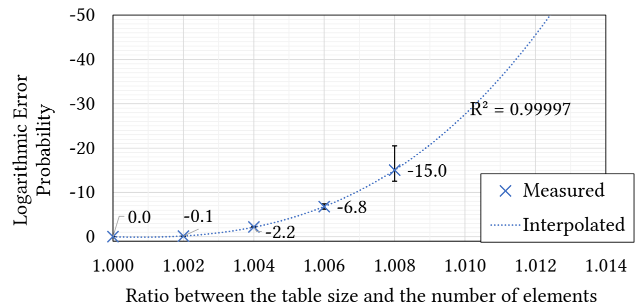


Figure 19: Failure Rate ($\log_2$ of mean failure probability) for 8-block Cuckoo Hashing using $k = 2$ hash functions and inserting 8000 elements. The error bars show the 99.99% Clopper-Pearson confidence intervals [26]. The interpolation uses polynomials of degree ≤ 3 for which the *coefficient of determination* R^2 is shown.

E Discussion

We expect our proposed nested Cuckoo hashing scheme to be of independent interest for other PSI-related 2PC protocols. Likewise, the formalization of *private indexed equality* (PIE) can serve as a basis for future work and be directly applied to PIR-PSI [30]. The AHE-based PSI protocol (described in Subsection 3.3) with δ-block Cuckoo hashing scheme achieves sublinear communication complexity for any AHE scheme. Thus, our protocol could also be used with other AHE schemes like Paillier [71].

A problem remains in the complex analysis of failure rates for the nested Cuckoo scheme, especially the combination of failure rates for the maximum bin size (as discussed by Pinkas et al. [78]) and the δ-block Cuckoo hashing. Even if we rely on empirical bounds, selecting the correct parameters for the nested Cuckoo hashing is non-trivial. However, given empirically interpolated failure probability equations like in PIR-PSI, this step could be automated like for parameters of (leveled) FHE schemes [9].

Exponential ElGamal-based Protocol Our exponential ElGamal-based constructions in Subsection 3.4, only relies on the simple ElGamal encryption and DDH assumption. Remark, additively homomorphic exponential ElGamal is easy to implement given practical libraries like OpenSSL [86].

With total running times of less than 7 seconds and communication of $\approx$6 MB (for $|Y| \leq 128, |X| = 2^{16}$), our evaluations in Subsection 5.2 attest to practical performance when using parallelization. For applications like private contact discovery, smaller online running times (e.g., <1 second) for larger sets are desirable and achieved by other protocols [23; 61; 72]. However, due to the familiarity of the underlying encryption scheme, the small communication costs, and the simple implementation, we expect our exponential ElGamal-based protocol to be actually relevant for certain practical applications.

BGV/BFV-based Protocol Our evaluation shows that for many input sizes, the BGV/BFV-based protocol variant outperforms our ElGamal-based approach, e.g., by a factor of ≈ 100 for $|Y| = 4096$. Using hardware acceleration for lattice-based schemes like Intel's HEXL extension might further improve the practical performance. However, the comparison is only valid for small bit sizes of $\rho = 32$, which might be unsuitable for many practical applications. Likewise, to ensure server privacy, we would require an additional OPRF masking step. Remark, the underlying cryptographic ring LWE assumption is well-established (in the meanwhile) and might be secure against quantum computers (in contrast to DDH).

E.1 Comparison to FHE-based PSI [22; 23; 27]

Chen et al. [23] have presented another PSI protocol based on (leveled) FHE (referred to as FHE-PSI). This work has been later improved by Chen et al. [23] and Cong et al. [27] to achieve state-of-the-art performance. As mentioned in Subsection 1.1, FHE-PSI uses homomorphic evaluations of polynomials to check item membership. Depending on the ciphertext packing, our BGV/BFV-based protocol needs only a constant number of ciphertext-ciphertext multiplications. The many improvements of FHE-PSI allow to perform a server online computation in ≈ 2.34 s (for $|Y| = 4096$ and $|X| = 2^{20}$). This comes at the cost of an offline computation time taking ≈ 29 s. Remark, to achieve semi-honest security, the offline time cannot be reused for other client's [22]. For the same parameters $|Y|, |X|$ we achieve a total running time of 6.89 s which additionally includes the offline computation, client computation and communication. Remark, that Cong et al. allow bit-lengths $\rho > 80$ where ours are only $\rho = 32$ as in Chen et al. [23]. However, we think our scheme can support larger bit-lengths and so-called labeled PSI analogous to the improvements of Cong et al. [27] and Chen et al. [22]. In contrast to Cong et al., our BGV/BFV-based protocol could also benefit from many PIR improvements as outlined in Subsection E.3.

E.2 Comparison to DH-based PSI [82; 85]

Rosulek and Trieu have presented improvements to the original DH-PSI protocol [49] leading to a fast protocol for small set sizes. Depending on the application, our ElGamal-based protocol is also only practical for small server set sizes $|X| \leq 2^{16}$. In comparison to DH-PSI, our protocol is adapted to a unbalanced asymmetric setting assuming a client with less computational resources. An asymmetric execution of DH-PSI would require that the client performs $O(|X|)$ exponentiations (whereas we require $O(\sqrt{|X|})$). For $|Y| = 32$ and $|X| = 2^{20}$), we only need communication costs of 10.9 MB (as shown in Figure 10 in the appendix). A similar (asymmetric) execution using DH-PSI would require at least ≈ 30 MB. Resende and Aranha have presented another protocol based on DH for the (asymmetric) unbalanced setting which achieves performance comparable to our FHE-based protocol [82]. However, the used filter techniques introduce false-positives and require a large client state (linear in $O(|X|)$) that is transferred in an item-dependent precomputation phase. Further, in contrast to other DH-based protocols [82; 85], our homomorphic encryption based approach allows extensions like labeled PSI [22] and so-called PSI-CA (outlined in Subsection E.3).

E.3 Future Work

In this section, we provide an overview of potential future work based on our PSI protocol and the described extensions. We expect our nested Cuckoo hashing construction also offers a basis for potential future work, which is not discussed in this work.

Generic Post-Computation Different variations of the PSI functionality have been proposed [27; 37; 51; 52; 76]. In the so-called PSI-CA problem, the client shall only learn the cardinality of the set intersection [37]. Our AHE-based PSI protocol (without FHE ciphertext packing) can be adjusted to solve the PSI-CA problem with almost no extra computational effort. Instead of sending back the server results for each outer Cuckoo table position, the server shuffles all ciphertext results c_f before sending them back. If a shuffled c_f decrypts to 0, the client only learns that one item $x \in X$ equals one $y \in Y$ and can, thus, compute the intersection set cardinality $|X \cap Y|$. However, we are also interested in computing any fixed but arbitrary function on the intersection as considered by some PSI protocols [54; 64; 69; 75]. For BFV/BFV, instead of simply subtracting and randomizing in our AHE-based comparison step, the server could use improved homomorphic comparisons [50] that yield an encryption of one if the elements are equal and an encryption of zero otherwise. The encrypted zeroes and ones can be used to run (leveled) FHE circuits for arbitrary functionalities.

Hashing Guarantees The problem with all Cuckoo hashing-based PSI protocols is a missing analytical bound for the hashing failure probability. Recently, Garimella et al. [38] have presented a construction based on several Cuckoo hashing tables that can provably reduce the failure probability given a higher (empirically validated) failure rate per table. However, the construction of Garimella et al. [38] is based on an encodings for masked elements that does not fit our comparison approach. The question of whether a variation of our AHE-based scheme can be combined with the constructions of Garimella et al. [38] is left for future work. The δ-blocked Cuckoo hashing on the server side also suffers from missing theoretical failure analyses. However, the theoretical results mentioned by Pinkas et al. [72] might offer implications for our constructions for large values of δ.

Improvements from PIR schemes Our protocol uses ideas many ideas from PIR protocol constructions, especially the bit-wise encryption of the index vectors and the construction for square-root complexity [5; 62]. As such, we expect that our protocol can directly benefit from a compressions of the encrypted index vector as proposed by Angel et al. [7]. Likewise, *compressible* FHE [41] could be used to reduce the computation complexity. State-of-the-Art PIR protocols also reduce the [5; 7; 67] server result size. However, combining our AHE-based comparison with the compression of the server result is not straightforward. Future work for the BGV/BFV-based variant could consider a different ciphertext packing approach. Instead of packing together all client elements, for each outer Cuckoo table entry, the corresponding client element and *EIV* could be packed into one (or more) ciphertexts. The resulting protocol would benefit from very small client set sizes in comparison to our packed batched computation (as shown in subsubsection 5.2.1). However, the alternative packing approach requires homomorphically rotating ciphertexts, which might increase the running times for larger $|Y|$.

Algorithm 1 PSI from nested Cuckoo hashing and AHE

Require: $H_1, \ldots, H_{k_1} : \mathcal{M} \to \{1, \ldots, l_1\}$ and $H'_1, \ldots, H'_{k_2} : \mathcal{M} \to \{1, \ldots, l_2\}$

procedure SERVER-PSI(X)
    $(CT^1, stash^1), \ldots, (CT^{l_1}, stash^{l_1}) \leftarrow$ CREATENESTEDCUCKOOHASHTABLE(X)
    for $i \leftarrow 1, \ldots, l_1$ **do**
        $L \leftarrow [\,]$
        $c_C \leftarrow$ RECEIVEENCRYPTEDELEMENT()
        for $j \leftarrow 1, \ldots, k_2$ **do**
            $EIV^j \leftarrow$ RECEIVEENCRYPTEDINDEXVECTOR()
            for $d \leftarrow 1, \ldots, \delta$ **do** ▷ For each bin in the inner δ-block CT
                $c_S \leftarrow \langle EIV^j, CT^i[d] \rangle$ ▷ Homomorphically evaluated dot product
                $r \leftarrow_{\$} \mathcal{M} \setminus \{0\}$ ▷ Fresh randomness
                $c_f \leftarrow r \boxdot (c_S \boxminus c_C)$
                PUSHTOLIST(L, c_f)
            end for
        end for
        for $e \in stash^i$ **do**
            $r \leftarrow_{\$} \mathcal{M} \setminus \{0\}$
            $c_f \leftarrow r \boxdot (c_S \boxminus e)$
            PUSHTOLIST(L, c_f)
        end for
        $L \leftarrow$ SHUFFLE(L) ▷ Random permutation
        SENDRESULTLIST(L)
    end for
end procedure
procedure CLIENT-PSI(Y)
    $CT \leftarrow$ CREATECT(Y)
    $R \leftarrow \{\}$
    for $i \leftarrow 1, \ldots, l_1$ **do**
        $e \leftarrow CT[i]$
        $\mathsf{c} \leftarrow \mathrm{Enc}(e)$
        SENDENCRYPTEDELEMENT(c)
        for $j \leftarrow 1, \ldots, k_2$ **do**
            $EIV^j \leftarrow$ CREATEENCRYPTEDINDEXVECTOR($H'_j(e)$)
            SENDENCRYPTEDINDEXVECTOR(EIV^j)
        end for
        $L \leftarrow$ RECEIVERESULTLIST()
        for $\mathsf{c} \in L$ **do**
            if DECRYPT(c) = 0 **then** $R \leftarrow R \cup \{e\}$
            end if
        end for
    end for
    return R
end procedure