On Deploying Secure Computing Commercially: Private Intersection-Sum Protocols and their Business Applications

Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, Moti Yung

Google, LLC 1600 Amphitheatre Pkwy, Mountain View, CA 94043
{mion, benkreuter, anergiz, sarvar, marianar, shobhitsaxena, karn, dshanahan, moti}@google.com

Abstract. In this work, we describe how to deploy a cryptographic secure computation protocol for routine use in industry. Based on our experience, we identify major preliminaries and enabling factors which we found to be critical to the successful deployment of such technology as a practical, and uniquely positioned method for solving the task at hand.

The specific technical problem that we tackled is that of computing Private Intersection-Sum. In this setting two parties hold datasets containing user identifiers, and one of the parties additionally has an integer value associated with each of its user identifiers. The parties want to learn (1) the number of users they have in common and (2) the sum of the integer values associated with these users, without revealing any more information about their private inputs. Private Intersection-Sum is not an arbitrary question, but rather arose naturally and was concretely defined based on a given central business need: computing aggregate conversion rate (or effectiveness) of advertising campaigns. This problem has both great practical value and important privacy considerations, and represents a type of analysis that occurs surprisingly commonly.

Among the factors that enabled our deployment, in this work we consider in more depth the technical issue of protocol choice and its performance implications. Specifically, we present a study involving three novel protocols for computing Private Intersection-Sum, which leverage three different basic protocol techniques including Random Oblivious Transfer, encrypted Bloom filters, and Diffie–Hellman style (Pohlig–Hellman specifically) double masking. We compare the three protocols under different instantiations of an additive homomorphic encryption, which is used as a building block in each protocol. We implement our constructions and compare their actual communication and computation overheads. Finally, we analyze the advantages of the DDH-based protocol which make it the solution of choice for our deployment setting.

Keywords: secure computation · private intersection-sum · secure aggregate ad conversion

1 Introduction

Secure computation protocols, where parties combine their inputs to generate output while keeping privacy of the inputs, were invented soon after public-key cryptography was conceived (the "Mental Poker" 1979 protocol by Shamir, Rivest, and Adleman [49] was perhaps the first such protocol). In this work, we report on a current industrial application, which led us to deploy secure computing technology to serve a concrete, routine, and central business area.

Modern cryptography can be essentially viewed as composed of three generations of technology: (1) Symmetric Cryptography, started in 1973 with the design of DES, motivated by the need of the expanding banking industry need for inter-branch secure communication; (2) Public Key cryptography from 1976-77, where the development of the Internet made large scale authenticated secure communication (i.e. authenticated key exchange) essential and prevalent; and (3) Secure computation protocols, which have enjoyed a very rich 40 year history of development of many basic and fundamental notions, protocols, techniques, and experiments, yet with much less actual deployments and use.

We argue that current modern computational settings involving companies' massive data and their need on the one hand to leverage their data for insights, while on the other hand having legitimate privacy needs, leads itself to potentially implementing secure computing technology. Indeed, the consideration given to privacy by users and governments around the world is growing rapidly. Yet, attempting deployment in an existing and operationally successful business requires care, and as a community, the cryptography and security community does not have extensive experience in how one adapts theoretical and even experimental protocols into deployed systems. With actual deployment in mind, we will identify, based on our project's experience, the major properties and enabling factors which facilitate this potential and make the use of the technology and its actual deployment possible, acceptable to business, and even uniquely qualified for the critical task at hand.

Concretely, the specific family of sharing problems we deal with gives rise to what we call **"the Intersection-Sum problem"**: two parties (companies) hold datasets containing user identifiers based on users active on their sides. The first party has a list of users associated with first activity, activity like, for example, viewing advertisements on the Internet, and the second side has a list of users who transacted on a second activity such as: transacted at the advertised entity. The second party additionally has an integer value associated with each user identifier (representing, say, that user's spending). A party wants to learn the number of users they have in common, and the *sum* of the associated integer values, but essentially "nothing more." These outputs are, in fact, crucial to assessing the effectiveness of the first activity: First, we learn how many users the first activity drives towards the second activity, and secondly, we learn how much the first activity increases the average spending of the users who are driven to the second activity. The need for such effectiveness measures over correlated activities are common in business, but privacy is an obvious countervailing con-

cern: businesses would prefer to keep additional data (besides upper bounds on the sizes of their lists) from being revealed.

1.1 Contributions

The contributions of our work are twofold: First, we found criteria for a business problem to be suitable and acceptable to using secure computation for solving it, while facing various business (and other real world) constraints. Secondly, we actually developed and deployed a large scale solution leveraging secure computation techniques. From both steps there are issues to learn and we touch upon them. Our major contributions here are technical and consist of the development of three novel protocols for secure computation of Private Intersection-Sum, a study that was needed in order to justify the choice of a highly probable successful path to deployment.

While initially this functionality looks very close to the Private Set Intersection (PSI) functionality, which securely computes the intersection of two input sets, it turns out that achieving the additional aggregation while hiding the intersection comes with efficiency costs. At the same time the PSI capability is an implicit required building block in the secure protocols for the extended functionality. Thus, our protocols present approaches that extend techniques for private set intersection in a way that allows to hide the actual intersection and only compute an aggregated functionality over corresponding attributes. These protocols are based on three different underlying techniques, which we chose in order to explore the whole available space of existing PSI solutions, and to determine which one of these approaches extends most efficiently to the private intersection-sum functionality.

There have been many works on the problem of PSI in recent years. We chose two approaches among these solutions and developed private intersection-sum constructions leveraging their techniques. The first one is based on Random Oblivious Transfer, and builds on techniques developed by [43] and [34]. This approach leverages oblivious PRF techniques (OPRF), which we extended in a two step oblivious evaluation that allows secret permutation of the evaluated inputs. This enables us to hide the identity of the elements in the intersection. In order to facilitate the aggregation functionality we leverage additive homomorphic encryption. The second protocol is based on Encrypted Bloom Filters and is inspired by several recent PSI solutions [16, 19, 20]. We construct an oblivious protocol for evaluating membership in an encrypted Bloom filter under additive homomorphic encryption. We also use additive homomorphic encryption for the aggregation functionality. In addition to the two protocols based on recent PSI techniques, we present a third protocol, which we call the DDH-style protocol. It is based on the classic set-intersection protocol of [50, 37], which uses the Pohlig–Hellman cipher (based on the hardness of Decisional Diffie–Hellman (DDH)) – this functionality can be also viewed as an oblivious PRF with shared key [31]. Similarly to the first construction the aggregation property is achieved through the use of additively homomorphic encryption.

In the three Private Intersection-Sum protocols that we summarize above the output recipient is the party who has input set of pairs of identifiers and values. We also describe "reverse" variants of the protocols, which change the output recipient to be the party that has input set consisting only of identifiers. All our constructions implement secure computation functionalities with security against semi-honest adversaries (in the next section we refer to the business constraints which allowed us to safely consider this model, and the advantage this path offers).

All our protocols use additively homomorphic encryption as a building block. We currently have three main ways to instantiate this encryption primitive: Paillier [42], Exponential (additive) ElGamal [21], or lattice-based constructions [27, 9, 8, 23]. We analyze the efficiency of our protocols under these different instantiations and compare the trade-offs that they offer. We prove the correctness and security of each of the protocols. We implement and benchmark each of these protocols, and compare their performance in both computation and communication. The DDH-style protocol achieves best communication efficiency among the three protocols, requiring 9.28MB of communication on input sets of 100,000 elements. Surprisingly, it also achieves best computation efficiency in the implementation variants that we measured – the execution of the DDH-style protocol for Private Intersection-Sum using RLWE homomorphic encryption for the associated values takes 395.78 seconds on sets of size 100,000 with at most 32-bit associated values. Although the underlying PSI solutions for the first two protocols (based on ROT and Bloom Filters) have been optimized for computation efficiency, we found that the dominant cost in the Private Intersection-Sum setting is the operations related to the homomorphic encryption required to guarantee the extended intersection hiding and aggregation properties. To exemplify this we analyze our protocol under ideal efficiency settings for the homomorphic encryption, which do give computational advantage to ROT and BF protocols.

Based on our efficiency evaluation the DDH-style protocol was the clear choice for practical deployment under the constraints that we have identified. The other two protocols have potential to be good alternative for settings where computation is the bottleneck, by developing further optimizations of homomorphic encryption such as packing, or instantiating the aggregation functionality with more communication-expensive MPC techniques. They also outperform the communication costs of existing constructions that consider generic private computation on the set intersection leveraging garbled circuits and other similar techniques [11, 22, 44] (though with significantly higher computation than those works).

Other immediate advantages of the choice have been that the computation of the protocol is parallelize-able and one can exploit multi-core and cloud computing efficiently. We also comment that we will argue in the next section that flexibility of the protocol is a business advantage since it may easily be reused with small tweaks to solve other emerging tasks: the protocol of choice can be easily modifies to reveal the set intersection if needed, and other modifications are also easy to extract by small program modifications.

1.2 Roadmap

In Section 2 we discuss our experience of bringing Private-Intersection-Sum from theory into practice in the context of business environment. In Section 3 we describe our approach and how it compares to existing works. In Section 5 we present our three protocols which use several cryptographic primitives defined in Section 4 and claim their correctness and security. In Section 6 we discuss the tradeoffs of using different encryption schemes to instantiate our protocols. In Section 7 we discuss some simple tweaks that can be made to each of our protocols to modify them for slightly different settings. In Section 8 we present measurements from our protocol implementations. In Section 9, we revisit our protocols from the lens of the deployment requirements discussed in Section 2.

In Appendices A, B and C we give the full security proofs for our protocols. Appendix D discusses "reverse" variants of our protocol. In Appendix E, we present additional measurements.

2 Deploying Secure Computation: A View from a Practical Perspective

Secure computation was long considered largely a theoretical tool, too inefficient for practice, but research has yielded numerous implementations achieving orders of magnitude efficiency improvements. A growing number of real-world applications of multiparty computation protocols have been reported in the last decade. One of the first was the widely-cited Danish Sugar Beet Auction [7]. Others include a financial application in Estonia [6], and a secure survey of faculty salaries at universities in Massachusetts [36]. More recently, Lindell et al. [3, 26] deploy secure computation for cryptographic operations running between machines owned by a single organization. Even more recently, startup companies aiming at performing machine learning on secure data have been established as well. However, routine use by organizations of secure computing between multiple entities or businesses remains rare.

Our goal in the deployment efforts were a few: we wanted to learn what problems and issues will be acceptable to the business to the point that they can risk relying on secure computing, we wanted to prove that a successful fully operational deployment is possible. The goal of this work is to discuss some of the issues, and cover a technical part of what led us to a successful deployment.

In this section, we overview our take-aways from this process in terms of the nature of the *problems* which make compelling candidates for the adoption of cryptographic solutions as well as the characteristics of the cryptographic *solutions* that are easiest to deploy. We add the caveat that our experience is reflective of a period of time when cryptographic solutions based on secure computation techniques are relatively novel in practice. This adds some unique factors which may no longer be relevant when MPC becomes a mainstream security approach.

Properties of a business problem: We start with a discussion about what makes a problem a good candidate for cryptographic secure computing based

solution, with references to the application we first employ the technology, when appropriate:

Strong business benefit: Given that cryptographic solutions add significant complexity and performance costs, a clear business benefit is a prerequisite to exploring such solutions. In this case the potential benefit outweighed the cost in development effort, as well as computation and communication costs, for executing a private solution.

Strong privacy need: For our application, the data needed to compute the analytics is split across different parties, each of which have a strong desire to keep their data private. This is a powerful motivation to find solutions with strong privacy properties.

Alternatives pose undesirable risks: The parties may have constraints related to the underlying data that make alternatives, like using a trusted third party or developing isolated execution environments, less palatable or not acceptable. When no other solution is acceptable, a relatively expensive cryptographic solution becomes more attractive, since it provides the desired end-to-end privacy.
Performance Issues: The underlying problem must be amenable to efficient solutions. The volume of data and the planned frequency of execution translate into performance requirements. For example, if a report is required every day, its production cannot take two days.

Not Real-Time: An application whose output is not needed in real-time (like analytics, learning, etc.) can much more easily tolerate the overhead costs associated with a cryptographic solution.

How to choose among possible solutions: Next we turn to the properties which guided our construction for the private set intersection-sum problem. We note that our context is secure computation across multiple businesses (over WAN). The characteristics for secure computing within a single business, or between a server and a mobile application both owned by the same company may be different. With this caveat, we made our solution decision based on the following characteristics:

Communication Efficiency: Somewhat surprisingly, for the offline 'batch computing' scenarios we consider, communication costs are far more important than computation. This is especially the case for a secure protocol involving multiple businesses, where servers cannot be co-located (Wide area network solutions). Networks are inherently shared, and it is much less expensive to add CPUs to a shared network than to expand network capacity. A rule of thumb we encountered is that doubling the communication cost of a solution is equivalent to increasing the computational cost by a factor of $20 \times -40\times$.

Well-understood tools and assumptions: Companies tend to be highly risk-averse when it comes to deploying novel cryptography, so it is very helpful to have protocols that use well-understood underlying assumptions and widely distributed implementations.In this respect our chosen solution is based on the classical hardness assumption of discrete logarithm (applied in the ECC setting) and a factoring related assumption. We note that in fundamental research, technical

novelties are always appreciated, yet in deployment, conservative, time-tested assumptions help mitigate the risk and are easier to accept. Yet, proving security and correctness of protocols, (as in fundamental research) is, in fact, a great tool in clarifying the situation to the designers themselves, and making explanations easier, convincing decision makers of the guarantees that are offered.

Simplicity: It is difficult to overstate the importance of simplicity in a practical deployment, especially one involving multiple businesses. A simple protocol is easier to explain to the multiple stakeholders involved, and greatly eases the decision to use a new technology. It is also easier to implement without errors, test, audit for correctness, and modify. It is also often easier to optimize by parallelizing or performing in a distributed manner. Simplicity further helps long-term maintenance, since, as time passes, a constantly increasing group of people needs to understand the details of how a solution works. Conversely, the more complex a solution gets, the more barriers appear in each context mentioned above. In theoretical work, simplicity and clarity is often just an aesthetic goal, but in implementation and deployment, simplicity has real and underappreciated practical advantage.

Flexibility: Flexibility is the fact that we need high suitability for the primary goal, yet at the same time, the solution needs to be easy to modify and reuse for related problems. Any successful deployment may drive demand for solutions to sub-problems or related problems. This implies a modular design (that can be easily adapted and will eventually lead to an API that can be applied to a class of problems) is always desirable.

Adversarial Behavior. Another important axis in considerations is the class of misbehavior the secure computation should protect against. Existing cryptographic protocols provide different levels of security guarantees, where the two extremes are the *honest-but-curious* (or semi-honest) and the *malicious* adversary models. Protocols that provide security against "honest-but-curious" adversaries, assume that the participants will follow the protocol steps honestly, but may try to learn as much as possible from the protocol messages. Honest-but-curious security means that such participants should learn nothing more than the protocol prescribes. On the other hand, malicious adversaries are allowed to deviate arbitrary from the protocol, and malicious security for a construction guarantees that even such adversaries cannot learn more about the private inputs. There also exist adversarial models that achieve intermediate guarantees. While protection against stronger adversaries is naturally more appealing, it does come with substantial efficiency cost (especially communication), which in the setting of our application was not acceptable when not essential. A semi-honest construction (modeled in [52], on the other hand, does provide meaningful guarantees in scenarios where the parties are large institutions and have additional non-cryptographic incentives to follow the protocol. We targeted such settings (at least, to start with) due to their performance and simplicity advantages (yet we look for solutions that with some added functionality can be extended when possible to counter more malicious parties). Note that the nature of the adver-

sary in real life is actually determined by **risk management** considerations which determine the nature of the interaction and the level of trust among the transacting parties (and in the range of possible partners there are enough that can be considered honest-but-curious due to the business/ legal context and arrangements).

In summary, identifying a good application for practical deployment of secure computation techniques involves solving technical, business, organizational, and human challenges which are often intertwined in the context of real applications. Thus, a viable solution cannot target just one subset of these areas of concern, but needs to address all of them.

The bulk of this paper does not cover all these subareas, rather it concentrates in some length on technically justifying the choice of a solution among possible available solutions. We demonstrate how the choice was based mainly on performance and simplicity, and also some other criteria like flexibility.

3 Related Work and Our Approach

3.1 Private Set Intersection: an Overview

Private set intersection is a specialized problem for secure two-party computation that has been studied extensively in a long sequence of works [30, 15, 35, 48, 45, 43, 19, 47, 34, 25, 12, 20, 16, 29]. Several works limit the parties to learning only the *cardinality* of the intersection [25, 1, 33, 51, 14, 40, 30].

The most prevalent technique among recent specialized protocols for set intersection, which also underlies the most efficient PSI construction currently [34], is based on the concept of oblivious pseudorandom functions (OPRF) [24]. The high level idea in this approach is to replace the items in the two input sets with corresponding PRF evaluations where one party has the OPRF key and can obtain locally its pseudorandom evaluations, and the oblivious evaluation property of the PRF is used to enable the other party to obtain its PRF values. Now any of the two parties can use the pseudorandom representations of the input sets to compute the intersection and it will also have the mapping of the intersection PRF values to the original input set values. Different papers have proposed different approaches for instantiation of the OPRF where currently the most efficient approach has been using random oblivious transfer (ROT), which is a modification of the oblivious transfer OT extension protocol [45], as an OPRF that allows a single oblivious evaluation. Additionally the set intersection protocol is reduced to many smaller set intersection protocols in which one of the parties always has a single element input set, these can be implemented using the limited functionality of the ROT-based OPRF. This is achieved by leveraging Cuckoo hashing [17] at one of the parties, which distributes its items in bins of size at most one, and simple hashing at the other party using all Cuckoo hash functions for each input element. At this point the parties run a mini-PSI protocol for the items sent to each bin in the parties' inputs.

Another approach for computing set intersection, which was introduced in the work of Dong et al. [19], and was later strengthened by Rindal et al. [47]

in the context of malicious set intersection leverages the notion of Bloom filters (BF) [5]. The two parties agree on the hash functions parameters for a Bloom filter. The first party computes a garbled Bloom filter which contains random values in its locations with the constraint that if an element is present in the set the values in the Bloom filter corresponding to its hash locations are additive shares of the element. The second party constructs a regular Bloom filter with binary values for its input set. Then, the two parties run oblivious transfer protocols where the second party retrieves the elements from the garbled BF of the first party only for locations where the value in its own BF is one. The second party can now check which of its input elements is in the intersection represented by the partial garbled BF that it has obtained through the OTs.

An approach for PSI computation based on the Decisional Diffie–Hellman (DDH) computational assumption, which comes from some older works [30, 15, 31, 35, 48], has the following idea. The two parties hash their values in a group where DDH holds. Each party has a secret exponentiation key. The parties use their keys to exponentiate their hashed input elements and exchange the results. They further exponentiate the elements in each of the received sets and send back the results. Now each party can compute the intersection locally. Technically this approach can be considered as an instantiation of the OPRF approach where the OPRF key is shared [31], but since the construction has different properties we discuss it independently. This technique has not been considered in more recent PSI works, which focus on computational efficiency, however, it brings desirable advantages for our setting where the limiting factor is communication complexity.

The PSI problem has also been solved using directly general two party computation techniques. Works using garbled circuit techniques allow computing the cardinality of the intersection, as well as more general functions of associated values of items in the intersection associated values. Huang et al.'s work [29] leverage the Sort-Compare-Shuffle approach which uses a $O(n \log n)$- depth circuit to perform set intersection. The Phasing technique of Pinkas et al. [43] hashes items to bins and uses garbled circuits to perform intersection bin-wise. More recently, several partly concurrent works [11, 22, 44] combine modern PSI tools such as phasing and sort-shuffle-compare garbled-circuits to create novel protocols that allow parties to output *shares* of the items in the intersection. This allows arbitrary two-party computations over associated values of items in the intersection, including learning sums of associated values. These works offer very competitive computational efficiency, at the cost of increased communication due to the use of techniques (including garbled circuits) that require bit-wise encryption and secure comparison of inputs. However, these works have the powerful advantage that they allow computing arbitrary functions over the intersection.

There are also works that target specialized settings to gain additional efficiency. In the setting of unbalanced input sets, Chen et al. [10] leverage fully-homomorphic encryption to create PSI protocols which offer improved communication efficiency, while at the same time allowing retrieval of associated data

for items in the intersection. [32] explore offline precomputation to maximize online efficiency for PSI.

3.2 Transforming PSI into Private Intersection Sum: Tag, Shuffle and Aggregate

As we discussed in the introduction the problem of private intersection-sum can be viewed as an extension of the functionality of private set intersection, and moreover the PSI functionality is an implicit component of our functionality. Thus a natural starting point for our constructions are existing PSI protocols. Our high level recipe for going from PSI to Private Intersection Sum is as follows:

1. Each of our protocols takes an existing PSI approach, and incorporates a shuffle step. This idea, first seen in [30], has the effect of turning the PSI protocol into a Private Intersection-Cardinality protocol: shuffling lets the participants count how many items are in the intersection, but not learn which specific item was in common.
2. Together with the identifiers, we insert homomorphic encryptions of associated values as "tags". These tags accompany the identifiers through the PSI protocol and the shuffle step. When one party is eventually computing the shuffled-intersection, that party can also homomorphically add together all "tags" to compute an encrypted intersection-sum of the associated values, which can subsequently be decrypted.

This recipe, which we refer to as "Tag, Shuffle, Aggregate", is conceptually straightforward and immediately appealing. However, it turns out the detailed application to each protocol can differ significantly, and a naive application of the recipe can lead to significant impact to protocol communication and computation costs. As an example, all of our protocols make use of an underlying additively homomorphic encryption scheme in various ways. It turns out the specific choice of additively homomorphic encryption chosen has a huge impact on efficiency, especially the availability of features like slotting. These tradeoffs are discussed in more detail in Section 6. Indeed, we view the careful optimizations and comparison of different approaches as a key contribution of our work.

A positive feature of the tag-shuffle-aggregate approach is that each protocol we obtain naturally degrades into a protocol for computing intersection-size, by simply skipping the "tag" and "aggregate" steps.

Given this recipe, our approach was to identify the most promising main techniques underlying existing PSI approaches and modify them to the setting of private intersection-sum, comparing the resulting protocols in terms of efficiency.

The Random-OT set intersection technique is the basis of our first private intersection-sum protocol described in Section 5.1, which extends it using homomorphic encryption and shuffling to hide the exact intersection while aggregating over it.

Our second construction for private intersection-sum presented in Section 5.2 is inspired by the techniques using encrypted Bloom Filters.

Our final solution, which is perhaps the most natural application of the recipe above, is our private intersection-sum solution based on DDH-style PSI. We extend the [30] protocol with homomorphic encryption for aggregation purposes, which we present in Section 5.3. This solution, though perhaps the technically simplest, turns out to be the protocol most appealing for deployment.

Revealing the Intersection Size: An important side effect of applying the recipe above is that all our protocols additionally reveal the *size* of the intersection, in addition to the intersection sum. In the business applications we consider, this is actually a *desirable* feature: parties actually *want* to learn how many items were in common. However, it is possible that in other settings, revealing the intersection size is undesirable leakage. In those settings, this additional leakage would be a drawback of our recipe and our protocols.

3.3 Why not Garbled Circuits?

Garbled circuits [52] and other generic secure multiparty computation protocols hold a lot of promise in computing a generic function over data held by two parties. There have even been a sequence of works applying tailored protocols based on Garbled Circuits and similar technologies to the problem of computing arbitrary functions privately over an intersection [29, 11, 22, 44]. These works offer very competitive computational efficiency, and have the additional advantage that they allow computing arbitrary functions over the intersection, without revealing the intersection size, which is a required side-effect of our recipe. A natural question therefeore arises: Why not use garbled-circuit protocols directly for computing the intersection size and intersection sum?

The reason why we do not deeply explore garbled-circuit style solutions in this work is due to our focus on communication complexity. Garbled-circuit-style protocols incur significantly increased communication cost due to the use of techniques that require bit-wise encryption and secure comparison of inputs. In particular, the communication cost of these protocols depends on the product of the set sizes, the security parameter, and also the *bit length of each input.* Ciampi et al [11] provide an excellent analysis of communication costs of garbled circuit protocols, which we partially reproduce in Appendix F. By comparison, the protocols in this paper have communication cost dependent only on the product of set sizes and the security parameter (see Table 1 which analyzes communication costs for our protocols).

This additional dependence on the bit length of each input leads, for example, to a 60× increase in communication cost for inputs that are 60 bits long. This is a significant increase in communication, and turns out to be infeasible for our specific application.

Because communication cost turns out to be crucially important to our business application, and the high communication cost of garbled-circuit-style approach is already clear from theoretical analysis, we do not explicitly implement and compare garbled circuit style approaches to our protocols in this work. However we note that in settings where communication is not a bottleneck, garbled-

circuit style solutions are likely to offer much improved computation costs and enable richer functionality compared to the protocols we present.

The comparison to garbled circuits gives another lens through which to view our recipe for transforming PSI into Private Intersection Sum: Our recipe allows computing over the intersection without using communication-inefficient bitwise encryption, but comes at the cost of needing to reveal the intersection size.

4 Preliminaries, Cryptographic Primitives and Hardness Assumptions

4.1 Notation

$\mathbb{Z}$ represents the integers, and $\mathbb{Z}^+$ represents the nonnnegative integers. The notation $[k]$ for $k > 0$ to donate the set $\{1, 2, ..., k\}$. The notation $[a, b]$ for $a < b$ represents the set of integers, $\{a, a + 1, ..., b\}$, while $[a, b)$ represents the set $\{a, a + 1, ..., b - 1\}$.

Let Π be a randomized interactive protocol between two parties P_1 and P_2, with P_1 holding input X and P_2 holding input Y. For security parameter λ, we use the notation $\mathsf{REAL}_{\Pi}^{i,\lambda}(X, Y)$ to represent the random variable corresponding to the *view* of P_i in the protocol for $i \in \{1, 2\}$. The view of a party is the input of that party, its randomness, and all messages received by that party during the execution of Π, including aborts by the other party. The random variable varies over the randomness of P_1 and P_2.

4.2 Decisional Diffie–Hellman

Definition 1 (Decisional Diffie–Hellman assumption (DDH)) *[18] Let $\mathcal{G}(\lambda)$ be a group family parameterized by security parameter λ. For every probabilistic adversary M that runs in time polynomial in λ, we define the advantage of M to be:*

$$|Pr[M(\lambda, g, g^a, g^b, g^{ab}) = 1] - Pr[M(\lambda, g, g^a, g^b, c) = 1]| - \frac{1}{2}$$

Where the probability is over a random choice $\mathcal{G}$ from $\mathcal{G}(\lambda)$, random generator g of $\mathcal{G}$, random $a, b, c \in [1, |\mathcal{G}|]$ and the randomness of M. We say that the Decisional Diffie–Hellman assumption holds for $\mathcal{G}$ if for every such M, there exists a negligible function ϵ such that the advantage of M is bounded by $\epsilon(\lambda)$.

In other words, the distributions (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are computationally indistinguishable. Through this paper we will write group operations using multiplicative notion.

4.3 Random Oracle Model

Definition 2 (Random Oracle) *[4] A random oracle RO is a map from $\{0, 1\}^*$ to $\{0, 1\}^\infty$ chosen by selecting each bit of $RO(x)$ uniformly and independently, for every x. $\{0, 1\}^\infty$ means an output of infinite length, which can be truncated to any fixed appropriate size depending on the setting.*

We say a protocol is secure when modeling a hash function H as a Random Oracle if we can prove security when each invocation of H in the protocol is replaced by a call to a Random Oracle RO on the same input.

4.4 Additively Homomorphic Encryption Scheme

Definition 3 (Additively Homomorphic Encryption) *An additively homomorphic encryption scheme consists of the following probabilistic polynomial-time algorithms:*

AGen: *Given a security parameter λ, $\mathsf{AGen}(\lambda)$ returns outputs a public-private key pair (pk, sk), and specifies a message space $\mathcal{M}$.*

AEnc: *Given the public key pk and a plaintext message $m \in \mathcal{M}$, one can compute a ciphertext $\mathsf{AEnc}(pk, m)$, an encryption of m under pk.*

ADec: *Given the secret key sk and a ciphertext $\mathsf{AEnc}(pk, m)$, one can run ADec to recover the plaintext m.*

ASum: *Given the public key pk and a set of ciphertexts $\{\mathsf{AEnc}(pk, m_i)\}$ encrypting messages $\{m_i\}$, one can homomorphically compute a ciphertext encrypting the sum of the underlying messages*[1]*:*

$$\mathsf{AEnc}(pk, \sum_i m_i) = \mathsf{ASum}(\{\mathsf{AEnc}(pk, m_i)\}_i)$$

We rely on the standard notion of CPA security of encryption, meaning, informally, that without knowledge of the private key sk, encryptions of different messages are computationally indistinguishable.

In addition, we will make use of the property that one can randomized ciphertext using a randomized procedure denoted as REFRESH function:

$REFRESH(\mathsf{ASum}(\ \{\mathsf{AEnc}(\ pk,\ m_i)\}))$ and $REFRESH(\mathsf{AEnc}(pk, \sum_i m_i))$ have statistically close distributions, even against an adversary that holds the secret key sk. In popular schemes such as Paillier [42] and Exponential ElGamal [21], this REFRESH can be achieved by always adding an additional fresh encryption of 0 in the ciphertexts to be summed by ASum, whereas in lattice based systems randomizing of error location by shuffling is performed as well.

We note also that homomorphic multiplication by a plaintext scalar can be trivially achieved by repeated homomorphic addition. All homomorphic encryption schemes we consider also support homomorphic multiplication by a plaintext scalar.

4.5 Oblivious Pseudorandom Function (OPRF)

Definition 4 (Oblivious Pseudorandom Function) *[39, 24] A two-party protocol P is said to be an Oblivious Pseudorandom Function (or OPRF) if there exists some Pseudorandom Function family f_k, such that P securely computes the following functionality.*

[1] If the sum is large, it can wrap around in the message space $\mathcal{M}$. In this work, we only consider messages and sums that are too small to wrap around.

Inputs: *Party 1 holds an evaluation point x; Party 2 holds holds a key k.*
Outputs: *Party 1 outputs $f_k(x)$; Party 2 outputs nothing*

In other words, an OPRF protocol allows a party to receive the output of a PRF f_k on an input x, using a key k held by the other party, while hiding the input x from the other party. Furthermore, the evaluation of the PRF f_k on all other inputs remains pseudorandom.

4.6 Random Oblivious Transfer (ROT)

Definition 5 (Random Oblivious Transfer) *[34] A two-party protocol P is said to be an Random Oblivious Transfer Protocol (or ROT) if it securely computes the following functionality*

Inputs: *Party 1 holds m evaluation point $x_1, ..., x_m$; Party 2 has no input.*
Outputs: *Party 2 outputs m lookup tables L_i; Party 1 outputs $\mathsf{L}_i[x_i]$.*

In the implementation of [34], the lookup tables L_i can be efficiently described using a PRF family f_k, where each lookup table $\mathsf{L}_i$corresponds to a different key k_i. (Technically, the PRF family is a "relaxed" PRF family with related keys. We refer the reader to [34] for details, and omit them here since the distinction does not have a significant impact on our protocol or proofs).

In this view, ROT can be viewed as multiple OPRF protocols executed simultaneously, where the keys for the underlying PRF are dynamically generated during the ROT execution.

4.7 Bloom Filter

Definition 6 (Bloom Filter) *[5] A Bloom Filter is a probabilistic data structure that supports insertion and membership checking. A Bloom filter is parameterized by a size N and a sequence of k randomly chosen hash functions $h_1, ..., h_k : \{0,1\}^* \to [N]$. An empty Bloom filter BF consists of N bits, each set to 0. inserting an item x into a Bloom Filter is implemented by setting the $h_i(x)$-th bit of the Bloom Filter to 1 for all i. Checking if an item x is in a Bloom Filter is achieved by checking that the $h_i(x)$-th bit of the Bloom Filter is 1 for all i.*

Bloom filters can possibly give false-positives on the membership test. The probability that an element x yields a false positive membership check is dependent on N, k and the number of items inserted into the Bloom Filter. [38] is a good reference on setting the Bloom Filter parameters.

4.8 Cuckoo Hash

Definition 7 (Cuckoo Hash Table) *[41] A Cuckoo Hash Table is a data structure supporting insertion and membership tests. It is parameterized by a number*

of bins N, a stash size s, and by k randomly chosen hash functions. An empty Cuckoo Hash Table has N empty bins. When inserting an item x into the table, if any of the bins $\{h_i(x)\}_{i=1}^{k}$ is empty, then x is placed in one of those bins. Otherwise, a bin in $\{h_i(x)\}_{i=1}^{k}$ is randomly chosen, and the item in that bin is replaced with x. The evicted item is then recursively inserted. If this process does not terminate after a fixed set of iterations, then the final evicted element is placed in a special bin called the stash. If the stash already contains s items, the insertion algorithm fails.

To check if an item x is in the Cuckoo Hash Table, one checks each of the bins in $\{h_i(x)\}_{i=1}^{k}$ for the item.

[17] show through extensive experiments that when inserting n items into a cuckoo hash for $n \geq 512$, $N = 1.5n$ bins, $k = 3$ hashes, and $s = 0$ stash size is sufficient to get a 2^{-40} probability of cuckoo hash failure.

5 Protocols for Private Set Intersection-Sum

In this section we present our cryptographic protocols for secure private set intersection-sum which use as a starting point the PSI protocols described in the previous section.

We first describe our two constructions leveraging recent ROT and BF techniques for set intersection. Following that we describe our DDH-based protocol, which is the protocol that we choose for deployment based on its communication efficiency, simplicity and underlying classical hardness assumption as we discuss in more detail in Section 2.

We note that the protocols presented in the next three sections assume that the party who has input pairs of identifiers and values, is the output receiver. In Appendix D we consider the setting where the party with only identifiers as input is the output receiver.

5.1 Random-OT-based Protocol

In this section, we describe a protocol for intersection-sum based on Random Oblivious Transfer, which is used in existing PSI solutions [45, 43, 19, 47]. We believe this is the first construction of the Private Intersection-Sum functionality from Random OT. Note that our construction can be naturally modified to a protocol for privately computing PSI-cardinality.

We use the following primitives defined in Section 4:

- Random OT with the following functionality: P_1 has no input and obtains a look-up table L that can be evaluated on some known domain. P_2 has input index IND and receives $\mathsf{L}[\mathsf{IND}]$. Additionally, any polynomial subset of entries of L (including or excluding $\mathsf{L}[\mathsf{IND}]$) appears pseudorandom to P_2.
- An additively homomorphic encryption scheme (AGen, AEnc, ADec).

We present our protocol Π_{ROT} in Fig. 1. At a high level, the protocol Π_{ROT} is similar to the Batched-OPRF based PSI of [34]. However, since we additionally want the intersection to be hidden, we introduce another step where we shuffle and re-mask the outputs of the batched-OPRFs. This shuffling is executed by the OPRF receiver encrypting the batched-OPRF outputs, and sending the encryptions to the OPRF sender. The sender then homomorphically adds a random value r_i to each ciphertext, and shuffles the ciphertexts among each other before sending them back. This implicitly changes the OPRF output $L_i(x_i)$ to $L_i(x_i) + r_i$, which is also pseudorandom, and can be viewed as a "new" OPRF. When the receiver decrypts, it therefore learns the "new" OPRFs evaluated on each of its inputs, but in shuffled order. Once we have this type of shuffled OPRF, it is natural to build a protocol for both intersection-sum and intersection-cardinality.

We note that [22] and [11] have some high-level similarities with our protocol, in that they adapt [34]-style bucketing with additional techniques to hide which specific items were in the intersection. While we hide the intersection by shuffling the buckets amongst each other, [22] and [11] achieve it by performing a tailored MPC-subprotocol for each bucket.

Correctness of the protocol follows immediately from inspection, assuming cuckoo hashing fails with negligible probability and that the Random OT outputs collide with negligible probability.

The security of our protocol follows from the security of the encryption scheme and the security properties of the Random-OT protocol. We state the theorems here and present the formal security proof of the protocol from Figure 1 in Appendix A.

Theorem 1 (Honest But Curious Security against P_1 in the DDH-based Protocol Π_{DDH}). *There exists a PPT simulator SIM_1 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

Where m_2 is the size of Party 2's input, $J = \{j : w_j \in \{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $|J|$ is its cardinality.

Theorem 2 (Honest But Curious Security against Party 2 in the ROT-based Protocol Π_{ROT}). *There exists a PPT simulator SIM_2 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\mathsf{REAL}^{2,\lambda}_{\Pi_{\mathsf{ROT}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_2(1^\lambda, \{(w_j, t_j)\}_{j=1}^{m_2}, m_1, S_J)$$

Where m_1 is the size of Party 1's input, $J = \{j : v_j \in \{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $S_J = \sum_{j \in J} t_j$ is the intersection-sum.

Randomized-OT based Private Intersection-Sum protocol

– **Inputs**:
  - P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$
  - P_2: Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$
– **Setup**:
  - P_1 generates an additively homomorphic encryption key-pair (pk_1, sk_1), and sends pk_1 to P_2.
  - P_2 generates an additively homomorphic encryption key-pair (pk_2, sk_2), and sends pk_2 to P_1.
– **Protocol Steps**:
  1. P_1 and P_2 choose Cuckoo hash table size n and stash size s, and choose k hash functions: $h_1, ..., h_k$ mapping inputs to bins $[]1, ..., n]$.
  2. P_1 hashes its items using Cuckoo hashing with $h_1, ..., h_k$ into n bins and a stash of size s, where each bin contains at most one item. P_1 fills in all empty bins and the empty positions in the stash with a dummy item. We refer to P_1's items (including both dummy and real values) as $v_1, ..., v_{n+s}$, where the items $v_{n+1}, ..., v_{n+s}$ are the items in the stash. P_1 aborts if the protocol fails.
  3. For each $1 \leq i \leq n+s$ the parties run a random OT where P_2 obtains a lookup table L_i and P_1 obtains $\mathsf{L}_i[v_i]$.
  4. P_1 encrypts and sends to P_2
  $$\{\mathsf{ct}_i = \mathsf{AEnc}(pk_1, \mathsf{L}_i[v_i])\}_{i=1}^{n+s}$$
  5. P_2 chooses random values $\{r_i\}_{i=1}^{n+s}$, and, for each $i \in [1, n+s]$, computes:
  $$\mathsf{ct}'_i = \mathsf{ct}_i + \mathsf{AEnc}(pk_1, r_i)$$
  P_2 sends P_1 the set of ct'_i in shuffled order.
  6. For each $(w_j, t_j) \in W$, P_2 computes:
  $$\begin{aligned} W_j =& \{\mathsf{L}_{h_i(w_j)}[w_j] + r_{h_i(w_j)}\}_{i=1}^{k} \\ & \cup \{\mathsf{L}_i[w_j] + r_i \ : \ i \in \{n+1, ..., n+s\}\} \\ \mathsf{ct}_j^t =& \mathsf{AEnc}(pk_2, t_j) \end{aligned}$$
  If any W_j has size $< k+s$, P_2 pads W_j with dummy random elements until it has size $k+s$. P_2 sends P_1 the set $\{(W_j, \mathsf{ct}_j^t)\}_{j=1}^{m_2}$, with the order of the tuples randomly permuted, in order to hide which elements of each W_j correspond to which hash function or stash index.
  7. P_1 decrypts all values ct'_i. Let V^* be the set of decrypted values. P_1 computes
  $$\mathsf{CT} = \sum_{j \ : \ |W_j \cap V^*| > 0} \mathsf{ct}_j^t$$
  P_1 rerandomizes and sends CT to P_2.
  8. P_2 decrypts CT to recover d.
– **Output**: P_2 outputs d, equal to $\sum_{i:w_i \in V} t_i$, the intersection-sum.

Fig. 1: Π_{ROT} : Randomized-OT based Private Intersection-Sum protocol.

Formal proofs appear in Appendix A.

5.2 Bloom-Filter-based Protocol

In this section, we describe a Private Intersection-Sum protocol based on the use of encrypted Bloom Filters, extending PSI approaches [16, 20, 19].

We use the following primitives defined in Section 4: a Bloom filter BF, an additively homomorphic encryption scheme (AGen, AEnc, ADec). We refer the reader to Section 4 and specifically 6 for definitions and notation for Bloom Filters.

We present the protocol in Figure 2. At a high level, one party inserts its input database into a Bloom Filter, and encrypts each bit of the Bloom Filter using an additively homomorphic encryption scheme under its key pk_1. That party then sends the encrypted Bloom Filter to the other party, who homomorphically computes membership of its elements in the Bloom Filter. If an item is in the Bloom Filter, that party will homomorphically compute an encryption of zero, and if it is not in the Bloom Filter, the party will compute an encryption of a uniformly random value. It sends these "membership" ciphertexts to the first party in shuffled order, who can then decrypt to learn how many items were in the intersection. The second party also sends homomorphically encrypted associated values with the membership ciphertexts under its own key pk_2, allowing the first party to compute the intersection-sum. The first party can additively mask the final intersection sum and then unmask the value decrypted by the second party.

The correctness of the above protocol can be seen directly, except in the case when:

- There are collisions in the Bloom Filter, and
- In step 4b, $r_j \cdot (\sum_j b_{\mathsf{IND}_j} - k) = 0$ but $\sum_j b_{\mathsf{IND}_j} \neq k$.

The probability of Bloom Filter collisions can be made negligible by appropriate choice of Bloom Filter size and number of hash function. The probability that $r_j \cdot (\sum_j b_{\mathsf{IND}_j} - k) = 0$ but $\sum_j b_{\mathsf{IND}_j} \neq k$ can be made negligible by making the message space of the encryption scheme exponentially large.

The security of this protocol follows from the security properties of the additive homomorphic encryption scheme, as well as the negligible probability of collisions in a Bloom-Filter. We discuss security in Appendix B.

5.3 DDH-based Protocol

In this section we present our DDH-based intersection-sum protocol which extends the corresponding PSI protocol. It uses the following primitives defined in Section 4: a group $\mathcal{G}$ in which the DDH assumption holds, an additively homomorphic encryption scheme (AGen, AEnc, ADec) and a hash function H modeled as a random oracle for the security proofs. We present our construction in Figure 3.

Encrypted-Bloom-Filter-based Intersection-Sum Protocol

– **Inputs:**
  - P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$.
  - P_2: Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$.
– **Setup**:
  - P_1 generates an additively homomorphic encryption key-pair (pk_1, sk_1), and sends pk_1 to P_2.
  - P_2 generates an additively homomorphic encryption key-pair (pk_2, sk_2), and sends pk_2 to P_1.
– **Protocol Steps**:
  1. P_1 and P_2 choose a size N for a Bloom filter, and k hash functions: $h_1, ..., h_k$ mapping identifiers to $[1, ..., N]$.
  2. P_1 computes a Bloom filter for its items, which we refer to as BF_1.
  3. P_1 sends to P_2 its Bloom filter entries encrypted under key pk_1:
  $$\{\mathsf{ct}_i^1 = \mathsf{AEnc}(pk_1, \mathsf{BF}_1[i])\}_i.$$
  4. For each of its elements w_i, P_2 computes the following:
     (a) $\mathsf{IND}_j = h_j(w_i)$ for $1 \leq j \leq k$;
     (b) Define ct_i^2, which encrypts 0 for a match or a random value for non-match, as follows:
     $$\begin{aligned}\mathsf{ct}_i^2 &= r_j.(\sum_j \mathsf{ct}^1_{\mathsf{IND}_j} - \mathsf{AEnc}(pk_1, k)) \\ &= \mathsf{AEnc}(pk_1, r_j(\sum_j b_{\mathsf{IND}_j} - k))\end{aligned}$$
     where r_j is random.
  5. P_2 sends to P_1 the following values in a permuted order:
  $$\{\mathsf{ct}_i^2, \mathsf{ct}_i^t = \mathsf{AEnc}(pk_2, t_i)\}_i.$$
  6. P_1 decrypts all values ct_i^2 and computes
  $$\mathsf{CT} = \sum_{j:\mathsf{ADec}(sk_1, \mathsf{ct}_j^2)=0} \mathsf{ct}_j^t$$
  P_1 rerandomizes CT and sends it to P_2.
  7. P_2 decrypts CT to recover d.
– **Output**: P_2 outputs $d = \sum_{i:w_i \in V} t_i$.

Fig. 2: Π_{BF}: Encrypted-Bloom-Filter-based Private Intersection-Sum protocol.

At a high-level, the two parties interact to hash and then exponentiate each entry in their datasets, using a multiplicatively shared secret exponent. During this interaction the values that are hashed and exponentiated are also shuffled. Thus, at the end, the output receiver is comparing pseudorandom representation of the input sets to compute the intersection without being able to relate it to the original input values. Our construction can be viewed as computing obliviously a ROM PRF $H(x)^k$ [31] where the key is multiplicatively shared, or evaluating deterministic Pohlig–Hellman cipher [28] with shared key. The group $\mathcal{G}$ in which we apply the Pohlig–Hellman cipher can be any group in which the DDH assumption holds, and that allows hashing to a random generator. We assume for simplicity in Figure 3 that $\mathcal{G}$ has prime order.

Our protocol extends ideas from the work of of Meadows [37] on private matching and Huberman [30] on intersection cardinality to also support intersection-sum. To achieve this, the party holding associated values sends along additive-homomorphic encryptions of the values under AEnc, and the other party homomorphically computes the sum of values in the shuffled-intersection. The sum is sent back to the first party, who then decrypts to recover the intersection-sum.

Correctness of the protocol follows immediately from inspection.

We state the theorems proving security below, and present a formal security analysis of the construction in Figure 3 in Appendix C.

Theorem 3 (Honest But Curious Security against P_1 in the DDH-based Protocol Π_{DDH}). *There exists a PPT simulator SIM_1 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

Where m_2 is the size of P_2's input, $J = \{j : w_j \in \{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $|J|$ is its cardinality.

Theorem 4 (Honest But Curious Security against P_2 in the DDH-based Protocol Π_{DDH}). *There exists a PPT simulator SIM_2 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\mathsf{REAL}^{2,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_2(1^\lambda, \{(w_j, t_j)\}_{j=1}^{m_2}, m_1, S_J)$$

Where m_1 is the size of P_1's input, $J = \{j : w_j \in (\{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $S_J = \sum_{j \in J} t_j$ is the intersection-sum.

The proofs appear in Appendix C.

DDH-based Private Intersection-Sum Protocol

– **Inputs**:
  - Both parties: A group $\mathcal{G}$ of prime order, and an identifier space $\mathcal{U}$. A hash function $\mathsf{H} : \mathcal{U} \to \mathcal{G}$, modeled as a random oracle, that maps identifiers to random elements of $\mathcal{G}$.
  - P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$, where $v_i \in \mathcal{U}$.
  - P_2: Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$, with $w_i \in \mathcal{U}$, $t_i \in \mathbb{Z}^+$.
– **Setup**:
  - Each P_i chooses a random private exponent k_i in the group $\mathcal{G}$.
  - P_2 generates a fresh key-pair $(pk, sk) \leftarrow \mathsf{AGen}(\lambda)$ for the additive homomorphic encryption scheme and sends the public key pk with P_1.
– **Round 1** (P_1):
  1. For each element v_i in its set, P_1 applies the hash function and then exponentiates it using its key k_1, thus computing $\mathsf{H}(v_i)^{k_1}$.
  2. P_1 sends $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ to Party 2 in shuffled order.
– **Round 2** (P_2):
  1. For each element $\mathsf{H}(v_i)^{k_1}$ received from P_1 in the previous step, P_2 exponentiates it using its key k_2, computing $\mathsf{H}(v_i)^{k_1 k_2}$.
  2. P_2 sends $Z = \{\mathsf{H}(v_i)^{k_1 k_2}\}_{i=1}^{m_1}$ to P_1 in shuffled order.
  3. For each item (w_j, t_j) in its input set, P_2 applies the hash function to the first element of the pair and exponentiates it using key k_2. It encrypts the second element of the pair using the key pk for the additive homomorphic encryption key. It thus computes the pair. $\mathsf{H}(w_j)^{k_2}$ and $\mathsf{AEnc}\,(t_j)$.
  4. P_2 sends the set $\{(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))\}_{j=1}^{m_2}$ to P_1 in shuffled order.
– **Round 3** (P_1):
  1. For each item $(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))$ received from P_2 in Round 2 Step 4, P_1 exponentiates the first member of the pair using k_1, thus computing $(\mathsf{H}(w_j)^{k_1 k_2}, \mathsf{AEnc}(t_j))$.
  2. P_1 computes the intersection set J:

$$J = \{j : \mathsf{H}(w_j)^{k_1 k_2} \in Z\}$$

     where Z is the set received from P_1 in Round 1.
  3. For all items in the intersection, P_1 homomorphically adds the associated ciphertexts, and computes a ciphertext encrypting the intersection-sum S_J:

$$\begin{aligned}\mathsf{AEnc}(pk, S_J) &= \mathsf{ASum}\left(\{\mathsf{AEnc}(t_j)\}_{j \in J}\right) \\ &= \mathsf{AEnc}\left(\sum_{j \in J} t_j\right)\end{aligned}$$

  4. P_1 sends this ciphertext to Party 2.
– **Output** (P_2): P_2 decrypts the additive homomorphic ciphertext received in Round 3 using the secret key sk to recover the intersection-sum S_J.

Fig. 3: Π_{DDH}: DDH-based Private Intersection-Sum protocol.

6 Instantiating the Homomorphic Encryption scheme

Each of the thee Private Intersection-Sum protocols that we presented, requires an additive-homomorphic encryption scheme in order to encrypt the associated values and homomorphically sum them. The Random-OT-based protocol and the Bloom-filter-based protocol additionally rely on an additive homomorphic encryption scheme in order to encrypt and intersect the identifiers themselves. The choice of homomorphic encryption scheme has a strong impact on both the communication and computation costs of each protocol. In this section, we discuss three possible additive homomorphic encryption schemes that we can use, namely Paillier encryption [42], Exponential ElGamal encryption [21], and schemes based on Ring-LWE [27, 9, 8, 23]. We discuss the various characteristics of each of these schemes, together with several optimizations that could be applied to each of them. These differences are summarized in Table 4.

Encryption Scheme	Unslotted-Expansion	Efficient Decryption	Slotting	Slot-Shuffle
Paillier/Damgard-Jurik encryption	High	✓	✓	X
Exponential ElGamal encryption	Medium	X	X	X
Ring-LWE encryption	High	✓	✓	✓ (Expensive)

Fig. 4: Comparison of the properties of various additively-homomorphic encryption schemes. Unslotted-Expansion is a qualitatiative comparison of the size of the ciphertext to the plaintext. Efficient Decryption denotes whether the scheme has computationally efficient decryption. Slotting denotes whether the scheme is compatible with encrypting multiple values into different "slots" of a single ciphertext. Slot-Shuffle denotes whether the scheme is compatible with homomorphically shuffling slots within a ciphertext and between ciphertexts.

6.1 Paillier Encryption

Paillier encryption [42] is one of the most well-known additively homomorphic encryption schemes, with security based on the Decisional Composite Residiuosity Assumption.

Paillier encryption requires relatively expensive modular exponentiation (*"public-key"*) operations in order to encrypt and decrypt. Paillier ciphertexts also have relatively large plaintext and ciphertext spaces, which leads to a large communication expansion when the values being encrypted are small. For example, using typical security parameters, a Paillier ciphertext will have ciphertext size 4096 bits, with plaintext space 2048 bits. However, if the associated values to be encrypted and summed are in the range of 20 bits, then using Paillier results in approximately 200× ciphertext expansion compared to the plaintext.

However, the Paillier scheme's large plaintext space is amenable to *slotting*. That is, one can divide up a 2048 bit plaintext space into multiple slots of, say, k bits each, with the first slot being the lowest order k bits of the plaintext space, the next slot being the next highest k bits, and so on. Each slot is then individually additively homomorphic: homomorphically adding two ciphertexts also homomorphically adds each slot in the ciphertexts. Finally, slots can be homomorphically rotated by homomorphically multiplying ciphertexts by the scalar 2^k.

Slotting helps make Paillier encryption much more efficient for encrypting associated values as follows: the party holding the associated values can encrypt a different associated value t_j into each slot. Once the other party determines which slots must be added together, it can rotate those values into a single privileged slot (namely the highest slot), and homomorphically add the rotated ciphertexts to compute the encrypted sum. The adding party can then mask the other slots with random values to hide any residual sums in those slots, and send the final ciphertext back to the first party to decrypt.

We note that the slots need to be large enough to accomodate the associated values, with some extra bits to allow summing without overflowing into the next slot, and some further additional bits to allow random masking. Additionally, only half the plaintext space can be used for slots, since otherwise there is no way to rotate the lowest slot into the highest slot's position without overflowing the ciphertext space. Slotting can also be combined with Damgard-Jurik optimizations [13], which allows increasing the plaintext space of the Paillier scheme with a proportionally smaller increase in the ciphertext size.

As an example, consider associated values of 20 bits each, with 1024 values in the intersection-sum (requiring 10 bits), and with 40 additional bits per slot for random masking. This leads to $k = 20 + 10 + 40 = 70$ bits per slot. Consider also using Damgard-Jurik optimizations with $s = 4$, which means each ciphertext has $4 \cdot 2048 = 8192$ bits of plaintext space, and the ciphertext itself has size $5 \cdot 2048 = 10240$ bits. Each ciphertext can accomodate at most $\lceil 8192/(70 \cdot 2) \rceil = 59$ slots with slack for masking and rotation. Therefore, the expansion of each ciphertext over the size of the associated values it can accomodate is $(10240/(20 \cdot 59))$, which corresponds to a $8.67\times$ expansion, a clear improvement over the $200\times$ originally considered.

A downside of slotting is that it is impossible to shuffle the slots both within a ciphertext and between different slotted ciphertexts. This makes the slotting optimization a poor fit for the "reverse" variant of the DDH-protocol (Appendix D) where encrypted associated values must be permuted, and similarly a poor fit for the $\mathsf{L}_i[v_i]$ values in the Randomized-OT based protocol (Section 5.1).

6.2 Exponential ElGamal

Exponential ElGamal encryption [21] also requires public-key operations for encryption, but has relatively smaller ciphertexts than Paillier encryption. A typical choice of parameters for ElGamal over elliptic curves will result in ciphertexts of length 512 bits, which for a 20 bit plaintext value would have an expansion

of 25×. A major downside of Exponential ElGamal is that decryption is expensive, involving solving DDH in the plaintext space. This limits the sizes of plaintexts that can be decrypted to about 60 bits, and also blocks optimizations like slotting.

However, if the values to be encrypted are small, on the order of 20 bits, and the number of them we expect to add together is also small, e.g. around 2^{16} elements, then after summing the plaintext size will be 36 bits, which is small enough to decrypt. This makes exponential ElGamal a good alternative to (unslotted) Paillier for encrypting associated values.

This approach trades communication for computation, as the decryption requires inverting a discrete logarithm; however, a trade-off between memory and CPU time can be made, if the range is small enough to permit a lookup table. For a 36 bit exponent the lookup table would be relatively large but well within the storage capacity of typical server-class machines.

Exponential ElGamal is definitely an improvement over Paillier in the Bloom Filter-based protocol (Section 5.2), where we use additive homomorphic encryption to encrypt each bit of the Bloom filter, and only need to test if decrypted values are zero or nonzero. For that protocol, we get smaller ciphertexts with no loss of computational efficiency, since testing if a ciphertext decrypts to 0 is very cheap in the exponential ElGamal scheme.

6.3 Ring-LWE-Based Cryptosystems

Another option is to use an encryption scheme based on the hardness of lattice problems. The most efficient schemes of this type are based on the hardness of Ring-Learning-With-Errors [9, 8, 27, 23].[2] Ring-LWE-based encryption schemes are also usually quite computationally efficient compared to schemes like Paillier and Exponential ElGamal, since they do not involve expensive modular-exponentiation operations to encrypt and decrypt, and their polynomial operations can be sped up using number-theoretic transforms (NTT).

Like Pailler encryption, RLWE-based schemes have ciphertexts with large plaintext spaces, but the plaintext space can be naturally decomposed into multiple 'slots', where each slot is individually additively homomorphic. In these schemes, plaintexts are high-degree polynomials, and each coeffecient of the polynomial can be viewed as an additively-homomorphic slot. Furthermore, slots can be homomorphically rotated by multiplying ciphertexts with the plaintext monomials corresponding to single powers of x. Therefore, the party holding the associated values can put one associated value into each slot of a plaintext, and encrypt and send the ciphertexts to the other party, who can homomorphically rotate the ciphertexts so that values to be added all end up in a specially designated slot (for example, the slot corresponding to the constant term of the polynomial), then homomorphically add the rotated ciphertexts, and finally, mask the other slots with random values.

[2] These schemes are actually "fully"-homomorphic, but here we consider only their additive homomorphism

However, one subtlety with RLWE-based encryption schemes is that the error in the resulting ciphertexts (needed for security) may reveal the sequence of operations used to create them. This leakage is even more pronounced to the party holding the decryption key, and may, in the worst case, reveal the individual items in the intersection to the party performing decryption. Unlike Paillier and Exponential ElGamal, it is not sufficient to simply add a fresh encryption of zero to the sum, since the error in a fresh encryption would be too small to mask the accumulated error from the homomorphic operations. Rather, the ciphertexts produced after homomorphic operations must be rerandomized with a large amount of error (proportional to the security parameter), in order to statistically hide which ciphertexts were used to produce it. This induces some additional communication overhead.

As a concrete example, consider if we had 20 bit values, and we expected 2^{10} of them to be added together, meaning the sum is bounded by 30 bits. Assume also that each polynomial coefficient has 3 bits of error added to it for RLWE-security. When 2^{10} values are added together, the error could grow to 13 bits. If we wanted 2^{-40} statistical hiding of the added error, we would need 40 additional bits per coefficient. This would lead to polynomial coefficients of size $30 + 13 + 40 = 83$ bits, to hold each 20 bit associated value, an expansion of $4.15\times$.

Another advantageous property of RLWE schemes is that the slots can be unpacked, shuffled and repacked homomorphically, thereby allowing slots to be arbitrarily shuffled. ([2] give one example of how to do so.) This could make RLWE-based schemes potentially a good fit for the "reverse" variants of the protocols, where slots must be shuffled, and also for the Random-OT variant of the protocol, where encrypted $\mathsf{L}_i[v_i]$ values must be masked and shuffled. However, the homomorphic unpack-and-repack operations add significant additional error to the ciphertexts, meaning the parameters have to be sufficiently increased to accommodate the error. The unpack-and-repack operation also has a significant additional computational cost. From our estimates, there are some choices of parameters for which one can get a marginal (less than 10%) decrease in total communication in the Random-OT protocol by using homomorphic unpacking, but at significant computational cost. We concluded that in the Random-OT variant, it is more effective to encrypt each $\mathsf{L}_i[v_i]$ value in a separate ciphertext.

7 Flexibility via tweaking

The recipe we apply of 'tag' a user with its associated integer, 'shuffle' and then 'aggregate' on matched shuffled elements in the intersection provides various extensions that allow us to tweak the solution to solve other related problems with minimal software updates to the protocol. This is an advantage when considering emerging changes and uses. Here are a few examples:

1. It is easy to ignore the tags and perform only set intersection size.
2. If we skip the shuffle phase, we get actual set intersection rather than computing its size only.

3. If we want to hide the size of the inputs, but we know a global upper bound on the sizes, we can always pad the solutions with 'dummy elements.' Namely, each user randomly chooses user identifiers (outside the domain of user names) and also adds a tag with value zero. This uniform bound is run always and will not affect that results since these random dummies are not going to intersect with extremely overwhelming probability.
4. If we want to hide the size of the intersection, we can do it by squaring the size of the sets artificially: The two parties can decide on a common seed, derive $O(n^2)$ common dummy elements (which are tagged with zero integer values); each side randomly selects half of these elements, and the size of their intersection is still $O(n^2)$. The resulting intersection over the original and the common dummy elements statistically hide the intersection size.
5. If we want to compute other statistics we can add tags, we can tag with different homomorphic encryption scheme that allows the computations over the tags. One simple example is tagging also with the square of the value, and computing the sum and the sum of squares which allows up to compute the first two moments of the tag values.

8 Measurements

In this section, we present the measurements of the communication and computation costs of our implementations of the three protocols for private intersection-sum presented in Sections 5.1,5.2 and 5.3 with different homomorphic encryption schemes.

	Exponentiations	Homomorphic Ops. (including Enc and Dec)	Misc.
DDH-based Protocol	$4M$	$2M + 1$	–
Random-OT-based Protocol	–	$5.9M + 1$	ROT$(1.3M)$ + Cuckoo(M)
Bloom-filter-based Protocol	–	$M(104 + 2.44\log_2(M)) + 1$	2BF(M)

Table 1: Computational operations in each Private Intersection-Sum protocol. Numbers presented are totals across both parties, each having M inputs. "Exponentiations" refers to the number of group exponentiations (sometimes called "public-key" operations). Homomorphic Ops. corresponds to the number of homomorphic encryptions, decryptions and additions performed. We assume each homomorphic operation has the same cost. Misc. includes other costs. ROT(x) refers to the computation cost of x Random OTs. Cuckoo(x) is the cost of cuckoo-hashing x items. BF(x) is the cost of inserting x items into a bloom Filter.

	Group Elts.	HE Ciphertexts	Misc.
DDH-based Protocol	$3M$	$M+1$	–
Random-OT-based Protocol	–	$3.6M+1$	ROT$(1.3M)$ + $96M$Bytes
Bloom-filter-based Protocol	-	$M(61 + 1.44\log_2(M)) + 1$	-

Table 2: Communication in each Private Intersection-Sum protocol. Numbers presented are totals across both parties, each having M inputs. Group Elements refers to the number of EC curve elements transferred. HE Ciphertexts corresponds to the number of homomorphic encryptions transferred, not taking into account optimizations like slotting. In miscellaneous costs, ROT(x) refers to the communication cost of x Random OTs.

In Tables 1 and 2, we present the computation and communication costs of each protocol, in terms of counts of different types of operations and different types of elements transferred. Based on these numbers we expect that the DDH protocol will have best communication while the most efficient protocol in terms of computation will depend on the relative costs of exponentiation and homomorphic operations, which we investigate through our experiments.

All computational cost measurements are in terms of total wall-clock runtime for both parties, running on a single-thread of a desktop workstation with an Intel Xeon CPU E5-1650 v3 (3.50 GHz) and 32 GB of RAM. Computational cost excludes the time spent transferring files between parties over the network.

	DDH Protocol + Paillier		Random-OT + Paillier		Bloom Filter + Paillier	
Input Size	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]
1000	5.27	0.45	11.31	1.64	19.64	8.55
2000	10.46	0.90	21.82	3.20	39.34	17.43
3000	15.65	1.35	32.31	4.76	59.74	26.42
4000	20.82	1.79	43.28	6.32	80.96	35.50
5000	25.96	2.24	53.83	7.89	100.62	44.63
10000	51.94	4.49	107.91	15.75	205.16	90.85
20000	104.49	8.90	216.57	31.51	417.49	184.89
30000	157.65	13.45	325.42	47.36	628.96	280.14
40000	208.84	17.93	434.27	63.22	843.16	376.16
50000	259.50	22.43	543.68	79.09	1,057.17	472.78
100000	519.35	44.84	1,115.10	158.74	2,159.75	961.54

Table 3: Comparison of Private Intersection-Sum protocol and variants, when using Paillier encryption to encrypt the associated values.

	DDH Protocol + RLWE		Random-OT + RLWE		Bloom Filter + RLWE	
Input Size	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]
1000	3.89	0.14	9.89	1.33	18.24	8.25
2000	7.88	0.22	19.61	2.52	36.60	16.76
3000	11.88	0.33	29.66	3.74	56.22	25.41
4000	15.78	0.41	39.33	4.94	74.28	34.11
5000	19.72	0.51	48.72	6.16	93.29	42.90
10000	39.33	0.97	97.67	12.22	191.34	87.33
20000	77.91	1.89	196.86	24.55	383.75	177.82
30000	117.36	2.82	294.95	36.73	582.56	269.51
40000	157.28	3.75	399.50	49.02	782.62	361.98
50000	195.68	4.68	492.17	61.34	987.30	455.04
100000	395.78	9.28	989.88	123.21	1988.47	925.98

Table 4: Comparison of Private Intersection-Sum protocol and variants, when using slotted Ring-LWE-encryption to encrypt the associated values.

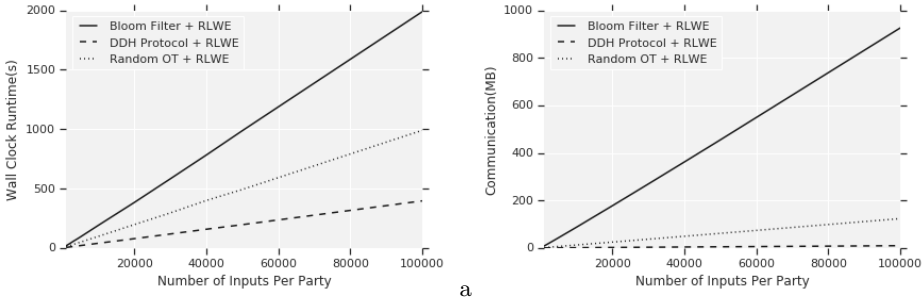
8.1 Choice of Parameters and Encryption Schemes

For all schemes and database sizes, we assume the input domain is the set of 128-bit strings, with associated values being at most 32 bits long. We additionally assume that the sum of associated values is bounded by 32 bits. All computation costs are presented assuming that each entry in each party's input is in the set intersection (which, in all protocols, maximizes computation).

We now describe the choice of encryption schemes and parameters for each of the protocols.

DDH-based protocol We use an elliptic curve with 224 bit group elements as the group $\mathcal{G}$. For the random oracle, we use SHA-256 applied to the input, and re-applied until the resulting output lies on the elliptic curve. In order to simulate a new random oracle for each execution, we choose a random seed and prepend it to each input before hashing.

Bloom Filter-based protocol We use Exponential ElGamal in order to encrypt the bits of the Bloom Filter. We implement Exponential ElGamal over an elliptic curve with 224 bit group elements. As noted in Section 6 Exponential ElGamal is a good fit for encrypting the Bloom Filter because it has relatively small ciphertexts compared to Paillier or Ring-LWE. The only cost is decryption, which requires computing a discrete logarithm, however in the Bloom-Filter based protocol, we only need to check if the decrypted value is 0, which can be done very efficiently. We also dynamically choose the number of hash functions and the Bloom Filter size based on the number of items held by each party, based on a 2^{-40} probability of incorrect Bloom Filter collision over all items. That is, we set the parameters so that the intersection is guaranteed to be correct, except with probability 2^{-40}.



a

(a) Comparison of Computation Costs for Intersection Sum.

(b) Comparison of Communication Costs for Intersection Sum.

Fig. 5: Compares computation and communication costs for the different variants of intersection-sum, on different input sizes. All protocols use slotted Ring-LWE based encryption to encrypt associated values. Wall-clock running times are the totals across both parties, excluding network transfer time. Communication costs are also totals for both parties.

Random-OT based protocol We use Paillier encryption to encrypt the $\mathsf{L}_i[v_i]$ values. We implement Paillier encryption with 768 bit primes, so that each ciphertext is 3072 bits, with 1536 bits of plaintext space. For the Random Oblivious Transfer variant, we choose the following parameters:

Cuckoo Hash. We use the heuristics provided in Appendix B of Demmler et al. [17] for the Cuckoo hash parameters. For the input sizes that we consider, these heuristics require $k = 3$ hash functions, no stash, with a Cuckoo hash table that can hold n elements with $\leq 1.5n$ bins.

Random-OT. We use the Batched Random OT protocol of Kolesnikov et al. [34], with 128 base public-key OTs and pseudorandom code output length 448 bits. For the actual pseudorandom code, we use a PRG based on SHA-256, sped up by native instructions. We also use SHA-256 as the hash function for the [34] protocol.

Finally, to encrypt the associated values, we present measurements for each of our three protocols using two different additive homomorphic encryption schemes.

"Unslotted" Paillier encryption with 768 bit primes, where, as mentioned earlier, each ciphertext is 3072 bits, with 1536 bits of plaintext space. The use of unsloted encryption means each associated value is encrypted in a separate ciphertext.

"Slotted" Ring-LWE encryption – each associated value is encrypted into a different slot. We use RLWE with an 80 bit ciphertext modulus and a 32 bit plaintext modulus, with 2048 coefficients per ciphertext, and 3 bits of error per coefficient. We mask the error in the sum ciphertext with 47 bits of randomness, which leads to at least 2^{-30} statistical-hiding of the error (even with our largest

input size of 100,000 entries). Each ciphertext is 20 KB and can hold up to 2048 values of 32 bits each.

8.2 Discussion of Measurements

We present our measurements in Tables 3, 4 and 5.

In Table 3, we show the costs for the 3 different variants of the protocols, using unslotted Paillier as the homomorphic encryption scheme for the associated values. As expected from our count estimates the DDH protocol has most succinct communication, whereas the ROT-protocol requires three times more communication and the BF-based protocol has 20 times more communication. The DDH protocol also turns out to be the most computationally efficient protocol, two and four times faster than the ROT and the BF protocols respectively. In Table 4, we show the costs when slotted Ring-LWE is used for the associated values, which leads to savings in both computation and communication, which are most significant for the DDH-based protocol. In this setting the DDH protocol continues to dominate the other two in both communication and computation efficiency. In Figure 5, we present the comparison graphs of the computation and communication costs of the three intersection-sum protocols using Ring-LWE based encryption for the associated values.

Using an "Ideal" Homomorphic Encryption Scheme We note that, as can be seen in both tables above, both the Random-OT-based and the Bloom Filter-based protocols have communication cost and computation costs far larger than the DDH-based one. In both of these protocols, computation and communication costs are dominated by the overhead of the additive homomorphic encryption scheme, especially in encrypting $\mathsf{L}_i[v_i]$ in the ROT variant, and the bits of the Bloom Filter in the BF variant. The effect of the homomorphic encryption schemes is particularly clear in Tables 1 and 2, which show large numbers of additive-homomorphic encryptionsTherefore, a natural question is the following:

"Would optimizing the homomorphic encryption schemes lead to protocols that are more efficient than Π_{DDH}, the Intersection-Sum Protocol based on DDH?"

To attempt to answer this question, we compare the three protocols assuming we have an "idealized" additive homomorphic encryption scheme, for which encryption, decryption and homomorphic addition have no computation cost, and the size of a ciphertext is equal to the size of the plaintext value being encrypted. The resulting costs are presented in Table 5 and Fig 7 in Appendix E.1, where we present the communication and computation costs of all three intersection-sum protocols, ignoring the computation costs and communication overheads due to the use of additive homomorphic encryption.

With such an "ideal" encryption scheme, we see that the Random-OT and Bloom Filter variants have huge computational advantages over the DDH-based Intersection-Sum Protocol (60x - 1000x faster). However, even with this idealized encryption scheme, both variants have a larger communication cost than the DDH-based protocol, with the Random OT-based protocol requiring 4x more

communication, and the Bloom Filter variant requiring 10x more communication. We believe this shows a clear advantage for the DDH-based Intersection-Sum protocol in terms of communication.

This evaluation shows that more efficient homomorphic encryption or replacement protocols for it may give advantage to the ROT and BF constructions in terms of efficiency.

Further optimizations We believe further optimizing the homomorphic encryption scheme is an interesting question for future work. We present some efforts towards this direction in Appendix E.2. More specifically, we consider using RLWE to encrypt both the $\mathsf{L}_i[v_i]$ values and the associated values in the Random-OT based protocol.

9 Choosing a Protocol to Deploy

From our overall analysis, we found that the DDH-style protocol Π_{DDH} was the one best suited to deployment. In the previous two sections we analyzed the flexibility and performance advantages of Π_{DDH} over the other protocols we considered. We now reflect on the other related factors and desired goals we developed throughout the project and initially (stated in Section 2).

A key factor to being able to deploy Π_{DDH} in practice was its simplicity. The fact that the protocol is a simple combination of blinded sum and blinded set intersection, contributed to the stakeholders being convinced of its value and simplicity of deployment. For security experts and risk analysts, the security was relatively straightforward to see, and was based on well-established assumptions used extensively elsewhere (DDH over curves as in TLS, and Paillier security which is related to factoring as is RSA which is in turn, is used in TLS). These factors raised confidence in the solution and eases its acceptance.

Simplicity and flexibility also made the protocol easier to implement correctly, and also to parallelize. Simplicity was a key factor in our deployment, and we anticipate this to commonly be the case for future deployments of secure computation between multiple businesses.

10 Conclusion

We have successfully developed and deployed a secure computation protocol in an industry setting as a solution for business interactions between different companies. The problem that our work has tackled is privacy preserving attribution of aggregate ad conversions. Throughout this process we encountered multiple challenges at both technical and business incentive level, based on which we have distilled some principles on what are good candidate problems and solution approaches for adoption in practice.

We developed three cryptographic solutions for the problem of Private Set Intersection-Sum, which underlies our application. We evaluated their efficiency under different instantiations of their building blocks and compared them both

asymptotically and in terms of actual measurements from our implementations. Based on this comparison, we selected the solution that best meets our most restrictive communication efficiency requirements as candidate for deployment, namely the DDH-based intersection-sum protocol. This solution has also an added benefit of simplicity, which becomes another critical component in the case of adoption of completely new security techniques in real applications.

References

1. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. pp. 86–97. SIGMOD '03, ACM, New York, NY, USA (2003)
2. Angel, S., Chen, H., Laine, K., Setty, S.: Pir with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 962–979. IEEE (2018)
3. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. Cryptology ePrint Archive, Report 2016/768 (2016), `http://eprint.iacr.org/2016/768`
4. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security. pp. 62–73. ACM (1993)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
6. Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis. Cryptology ePrint Archive, Report 2011/662 (2011), `http://eprint.iacr.org/2011/662`
7. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., et al.: Secure multiparty computation goes live. In: International Conference on Financial Cryptography and Data Security. pp. 325–343. Springer (2009)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 13 (2014)
9. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing **43**(2), 831–871 (2014)
10. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255. ACM (2017)
11. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. IACR ePrint **105**, 2018 (2018)
12. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: International Conference on Applied Cryptography and Network Security. pp. 125–142. Springer (2009)
13. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In: International Workshop on Public Key Cryptography. pp. 119–136. Springer (2001)
14. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: International Conference on Cryptology and Network Security. pp. 218–231. Springer (2012)

15. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 213–231. Springer (2010)
16. Debnath, S.K., Dutta, R.: Secure and efficient private set intersection cardinality using bloom filter. In: International Information Security Conference. pp. 209–226. Springer (2015)
17. Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: Pir-psi: Scaling private contact discovery (2018)
18. Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory **22**(6), 644–654 (1976)
19. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 789–800. ACM (2013)
20. Egert, R., Fischlin, M., Gens, D., Jacob, S., Senker, M., Tillmanns, J.: Privately computing set-union and set-intersection cardinality via bloom filters. In: Australasian Conference on Information Security and Privacy. pp. 413–430. Springer (2015)
21. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
22. Falk, B.H., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions (2018)
23. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive **2012**, 144 (2012)
24. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Theory of Cryptography Conference. pp. 303–324. Springer (2005)
25. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: International conference on the theory and applications of cryptographic techniques. pp. 1–19. Springer (2004)
26. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. Cryptology ePrint Archive, Report 2016/944 (2016), http://eprint.iacr.org/2016/944
27. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 465–482. Springer (2012)
28. Hellman, M.E., Pohlig, S.C.: Exponentiation cryptographic apparatus and method (Jan 3 1984), uS Patent 4,424,414
29. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS (2012)
30. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM conference on Electronic commerce. pp. 78–86. ACM (1999)
31. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: SCN (2010)
32. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. Proceedings on Privacy Enhancing Technologies **2017**(4), 177–197 (2017)
33. Kissner, L., Song, D.: Privacy-preserving set operations. In: Proceedings of the 25th Annual International Conference on Advances in Cryptology. pp. 241–257. CRYPTO'05, Springer-Verlag, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11535218_15

34. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 818–829. ACM (2016)
35. Lambæk, M.: Breaking and fixing private set intersection protocols. Tech. rep., Cryptology ePrint Archive, Report 2016/665, 2016. http://eprint. iacr. org/2016/665 (2016)
36. Lapets, A., Volgushev, N., Bestavros, A., Jansen, F., Varia, M.: Secure multi-party computation for analytics deployed as a lightweight web application. Tech. rep., Computer Science Department, Boston University (2016)
37. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: Security and Privacy, 1986 IEEE Symposium on. pp. 134–134. IEEE (1986)
38. Mitzenmacher, M., Upfal, E.: Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge university press (2005)
39. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. Journal of the ACM (JACM) **51**(2), 231–262 (2004)
40. Narayanan, G.S., Aishwarya, T., Agrawal, A., Patra, A., Choudhary, A., Rangan, C.P.: Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In: International Conference on Cryptology and Network Security. pp. 21–40. Springer (2009)
41. Pagh, R., Rodler, F.F.: Cuckoo hashing. Journal of Algorithms **51**(2), 122–144 (2004)
42. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 223–238. Springer (1999)
43. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: private set intersection using permutation-based hashing. In: Proceedings of the 24th USENIX Conference on Security Symposium. pp. 515–530. USENIX Association (2015)
44. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based psi via cuckoo hashing. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 125–157. Springer (2018)
45. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on ot extension. In: Usenix Security. vol. 14, pp. 797–812 (2014)
46. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Transactions on Privacy and Security (TOPS) **21**(2), 7 (2018)
47. Rindal, P., Rosulek, M.: Improved private set intersection against malicious adversaries. Tech. rep. (2016)
48. Segal, A., Ford, B., Feigenbaum, J.: Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In: FOCI (2014)
49. Shamir, A., Rivest, R.L., Adleman, L.: Mental Poker. Technical Memo LCS/TM-125, Massachusetts Institute of Technology (1979)
50. Shamir, A.: On the power of commutativity in cryptography. In: Proceedings of the 7th Colloquium on Automata, Languages and Programming. pp. 582–595. Springer-Verlag, London, UK, UK (1980), `http://dl.acm.org/citation.cfm?id=646234.682550`
51. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security **13**(4), 593–622 (2005)
52. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167. IEEE (1986)

A Security Analysis for Random OT-Based Protocol

In this section, we prove security for the Random-OT based Private Intersection-Sum protocol Π_{ROT} presented in Section 5.1 and Figure 1. We prove security in the honest-but-curious model.

Let $\mathsf{REAL}^{i,\lambda}_{\Pi_{\mathsf{ROT}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2})$ be a random variable representing the view of P_i in a real execution of the Random-OT based protocol Π_{ROT}, where the random variable ranges over the internal randomness of all parties.

We restate the theorem from Section 5.1:

Theorem 1 (Honest But Curious Security against Party 1 in the ROT-based protocol Π_{ROT}). *There exists a PPT simulator SIM_1 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{ROT}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

Where m_2 is the size of Party 2's input, $J = \{j : w_j \in \{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $|J|$ is its cardinality.

Proof. We describe the simulator algorithm SIM_1 in Algorithm 1.

Algorithm 1 The simulator for Party 1 in the ROT-based protocol.

Input:$(\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$ **Output:**$SimView(P_1)$ $\mathsf{SIM}_1(\lambda, \{v_i\}_{i=1}^{m_1}, |J|)$

1: Honest simulate the Setup phase between P_1 and P_2.
2: Honestly simulate steps 1-4 between P_1 and P_2, receiving the set $\{\mathsf{ct}_i = \mathsf{AEnc}(pk_1, \mathsf{L}_i[v_i])\}_{i=1}^{n+s}$ at the end of Step 4.
3: In Step 5, choose a set A^* consisting of $n+s$ random elements a_i. Compute ct'_i to fresh encryptions to each a_i under P_1's public key pk_1, and send these values to P_1.
4: In Step 6, choose the set $\{(W_j, \mathsf{ct}^t_j)\}_{i=1}^{m_2}$ such that each W_j contains exactl $k+s$ randomly chosen elements, and each ct^t_j is a fresh encryption of 0 under P_2's key pk_2. For exactly $|J|$ indices j, replace a single random element of W_j with an element of A^*, chosen randomly without replacement (i.e. each element in A^* is used at most once). Send the set $\{(W_j, \mathsf{ct}^t_j)\}_{i=1}^{m_2}$ to P_1.
5: Simulate Step 7 for P_1 honestly.
6: Output the view of P_1 in this interaction.

We argue that

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{ROT}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

using a multi-step hybrid argument, where each neighboring pair of hybrid distributions is computationally indistinguishable.

Hyb_0 The transcript corresponding to the view of Party 1 in a real execution of the protocol.

Hyb_1 The same as Hyb_0, except, in Step 6, P_2 replaces the additively-homomorphic ciphertexts ct_j^t with fresh encryptions of 0 under pk_2.

Hyb_2 The same as Hyb_1, except in Step 6 for all j, P_2 replaces the elements in $W_j \setminus V^*$ with uniformly random values.

Hyb_3 The same as Hyb_2, except, in Step 5, each ct'_i is replaced with an encryption of a uniformly random value a_i, and, in Step 6, the corresponding element of W_j (which is an element of $W_j \cap V^*$) is also replaced with a_i.

Hyb_4 The view of Party 1 output by SIM_1.

We now argue that each successive pair of hybrids in the sequence above is indistinguishable. Notice first that Hyb_0 and Hyb_1 are indistinguishable by the hiding property of the additive-homomorphic encryption scheme. Hyb_1 and Hyb_2 differ exactly in that the $\mathsf{L}_i[w_i]$ values for w_i not in the set held by P_1 have been replaced with uniformly random values. Therefore Hyb_1 and Hyb_2 can be shown indistinguishable based on the pseudorandomness of non-retrieved items in the Random OT protocol. Hyb_2 and Hyb_3 can be shown to be indistinguishable using the one-time-pad property of adding the random value r_i, together with the special property of the homomorphic encryption scheme discussed in Definition 3, namely that a fresh encryption is indistinguishable from one produced using homomorphic operations. Finally, Hyb_3 and Hyb_4 are identically distributed.

Since each pair of neighboring hybrids is indistinguishable, we conclude that

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{ROT}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

We now argue that P_2 learns nothing from the protocol except the intersection-sum. We restate the theorem from Section 5.1:

Theorem 2 (Honest But Curious Security against Party 2 in the ROT-based Protocol Π_{ROT}). *There exists a PPT simulator SIM_2 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\begin{aligned}&\mathsf{REAL}^{2,\lambda}_{\Pi_{\mathsf{ROT}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2})\\ &\approx\\ &\mathsf{SIM}_2(1^\lambda, \{(w_j, t_j)\}_{j=1}^{m_2}, m_1, S_J)\end{aligned}$$

Where m_1 is the size of Party 1's input, $J = \{j : v_j \in \{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $S_J = \sum_{j \in J} t_j$ is the intersection-sum.

Proof. We observe that P_2 view in the protocol consists of the following:

1. Whether P_1 aborts due to cuckoo-hashing failure (Step 2).
2. The Sender's view in a Random-OT execution (Step 3).
3. The set $\{ct_i\}_{i=1}^{n+s}$ of ciphertexts encrypted under P_1's key pk_1 (Step 4).
4. A ciphertext CT encrypting the intersection sum d under P_2's key pk_2 (Step 7).

Observe that for the items 2 and 3 (the sender's view in a Random-OT protocol and encryptions under the key of the other party), P_1 learns nothing. Furthermore, for item 4, P_2 sees a ciphertext encrypting the homomorphically-computed sum d, which, by the special property of the homomorphic encryption scheme discussed in Definition 3, is indisitinguishable from a fresh encryption of d. Finally, the probability of cuckoo-hash failure can be made negligible by appropriate choice of parameters.

Therefore, we can simulate the view of P_2 as follows:

- The Simulator never aborts in Step 2.
- In Step 3, the Simulator requests arbitrarily chosen values a_i in the Random OT protocol.
- In Step 4, the Simulator sends P_2 $\{ct_i\}_{i=1}^{n+s}$ to be fresh encryptions of 0.
- In Step 7, the Simulator sends CT to be a fresh encryption of d under pk_2.

It is straightforward to show that the view of P_2 in the above simulated execution is indistinguishable from P_2's view in a real execution. We leave the detailed hybrid proof as an exercise.

B Security Analysis for Bloom-Filter-Based Protocol

We briefly argue security against semi-honest adversaries for the Bloom-Filter-based Intersection-Sum protocol Π_{BF} presented in Section 5.2 and Figure 2. The arguments below assume parameters have been set to have negligible probability of Bloom-Filter collision.

We first argue that P_2 learns nothing more than the intersection-sum d. Observe that P_2's view consists of

- Ciphertexts encrypted with P_1's encryption key pk_1 (Step 3).
- A rerandomized encryption CT of the homomorphically-computed intersection-sum d (Step 6).

We can thus simulate its view by sending it encryptions of 0 in Step 3, and a fresh encryption of d in Step 6. The indistinguishability of this simulated view from its view in a real protocol can be seen based on the hiding of the encryption scheme, as well as (for Step 3) and the special property of the homomorphic encryption scheme discussed in Definition 3, namely that a fresh encryption is indistinguishable from one produced using homomorphic operations (for Step 6). We conclude that P_2 learns only the intersection-sum d from the protocol Π_{BF}.

For P_1, we argue it learns nothing more than the intersection cardinality C. We sketch a simulator SIM as follows.

- In Step 5, SIM replaces the ct_i^2 values with encryptions of 0 at exactly C random positions, and encryptions of uniformly chosen random values at all other positions. SIM further replaces all ct_i^t with encryptions of 0. It sends all such pairs $(\mathsf{ct}_i^2, \mathsf{ct}_i^t)$ to P_1.
- For all other steps, SIM simulates the behavior an honest P_2.

One can see that P_1's view in the simulation described above is indistinguishable from a (correct) real execution by the hiding property of the encryption scheme.

We reiterate that, unlike the real execution, SIM has zero correctness error. Therefore, in order for the real and simulated executions to remain indistinguishable, we require that parameters have been selected to ensure that real executions have negligible correctness error.

C Security Analysis for DDH-Based Protocol

In this section, we will prove security of the DDH-based protocol Π_{DDH}, presented in Figure 3, Section 5.3. The proof is in the honest-but-curious model, where we assume participants follow the steps of the protocol honestly, but try to extract as much information as possible afterwards from the protocol transcript. This model still requires some degree of trust between the two parties not to deviate from the prescribed protocol.

We prove security in the honest-but-curious model; our proof is similar to the proof given by Agrawal et al. [1]. We show security by giving a simulator that can indistinguishably simulate the view of each honest party in the protocol given only that party's input, the cardinality of the intersection, and the intersection-sum (but not the input of the other party). Intuitively, this will show that each party learns nothing more by participating in the protocol than the cardinality of the intersection and the intersection sum.

In such a protocol execution, the *view* of a party consists of its internal state (including its input and randomness) and all messages this party received from the other party (the messages sent by this party do not need to be part of the view because they can be determined using the other elements of its view).

Let $\mathsf{REAL}^{i,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2})$ be a random variable representing the view of P_i in a real protocol execution, where the random variable ranges over the internal randomness of all parties, and the randomness in the setup phase (including that of the Random Oracle).

Our first theorem, which we restate from Section 5.3, shows that P_1's view in the protocol Π_{DDH} can be simulated given only that P_1's input and the size of the intersection (but not the input of P_1).

Theorem 3 (Honest But Curious Security against P_1 in the DDH-based Protocol Π_{DDH}). *There exists a PPT simulator* SIM_1 *such that for all security parameters* λ *and inputs* $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

Where m_2 *is the size of* P_2*'s input,* $J = \{j : w_j \in \{v_i\}_{i=1}^{m_1}\}$ *is the intersection set, and* $|J|$ *is its cardinality.*

Algorithm 2 The simulator for P_1 in the DDH-based Protocol

Input:$(\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$ **Output:**$SimView(P_1)$ $\mathsf{SIM}_1(\lambda, \{v_i\}_{i=1}^{m_1}, |J|)$:

1: Generate key $k_1 \in \mathcal{G}$, and key-pair (pk, sk) for the additively homomorphic encryption scheme.
2: Honestly generate and send $\{\mathsf{H}(v_i)^{k_1}\}_{i\in[m]}$ in shuffled order as P_1's message in Round 1.
3: Create a dummy set $V^* = \{g_i\}_{i=1}^{m_1}$, where each g_i is randomly selected from $\mathcal{G}$. Send $\{g_i^{k_1}\}_{i=1}^{m_1}$ in shuffled order as P_2's message in Step 2 of Round 2.
4: Create a dummy set $W^* = \{h_j\}_{j=1}^{m_2}$ for P_2 by setting $h_j = g_j$ for $j \in \{1, ..., |J|\}$, and each h_j for $j \in \{|J|, ..., m_2\}$ is randomly selected from $\mathcal{G}$.
5: Send $\{(h_j, \mathsf{AEnc}(pk, 0))\}_{j=1}^{m_2}$ in shuffled order as P_2's message in Step 4 of Round 2, where each $\mathsf{AEnc}(0)$ is freshly generated.
6: Honestly generate P_1's message in Round 3 using P_2's dummy messages from the previous step.
7: Output P_1's view in the simulated execution above.

Proof. We describe the simulator algorithm SIM_1 in Algorithm 2. Notice that the main difference between SIM_1 and a real protocol execution is in Round 2: instead of sending $\{\mathsf{H}(v_i)^{k_1 k_2}\}$ and $\{(\mathsf{H}(w_j)^{k_2}\}$ as in a real execution, SIM_1 instead uses random group elements $\{g_i\}$ and $\{h_j\}$ which have an intersection of the same size, and additively homomorphic encryptions of 0. We argue that

$$\mathsf{REAL}^{1,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_1(1^\lambda, \{v_i\}_{i=1}^{m_1}, m_2, |J|)$$

using a multi-step hybrid argument, where each neighboring pair of hybrid distributions is computationally indistinguishable.

Hyb_0 The view of P_1 in a real execution of the protocol.

$\mathsf{Hyb}_{1,0}$ The same as Hyb_0, except, in Round 2, all additively-homomorphic ciphertexts sent by P_2 are replaced with fresh encryptions of 0.

$\mathsf{Hyb}_{1,i}$ for $i \in \{1, ..., m_1 - |J|]\}$: The same as $\mathsf{Hyb}_{1,i-1}$, except with $\mathsf{H}(v_{i^*})^{k_1 k_2}$ replaced by $g_{i^*}^{k_1}$ in Party 2's message in Step 2 of Round 2, where where v_{i^*} is the lexicographically smallest as-yet-unreplaced element of $\{v_i\}_{i=1}^{m_1} \setminus \{w_j\}_{j=1}^{m_2}$, and g_{i^*} is a random element of $\mathcal{G}$.

$\mathsf{Hyb}_{2,0}$ Identical to $\mathsf{Hyb}_{1,m-|J|}$.

$\mathsf{Hyb}_{2,j}$ for $j \in \{1, ..., n - |J|]\}$: The same as $\mathsf{Hyb}_{2,j-1}$, except with $\mathsf{H}(w_{j^*})^{k_2}$ replaced by h_{j^*} in Party 2's message in Step 4 of Round 2, where where w_{j^*} is the lexicographically smallest as-yet-unreplaced element of $\{w_j\}_{j=1}^{m_2} \setminus \{v_i\}_{i=1}^{m_1}$, and h_{j^*} is a random element of $\mathcal{G}$.

$\mathsf{Hyb}_{3,0}$ Identical to $\mathsf{Hyb}_{2,n-|J|}$.

$\mathsf{Hyb}_{3,k}$ for $k \in \{1, ..., |J|\}]$: The same as $\mathsf{Hyb}_{3,k-1}$, except

- $\mathsf{H}(v_{k^*})^{k_1 k_2}$ replaced by $g_{k^*}^{k_1}$ in P_2's message in Step 2 of Round 2 and
- $\mathsf{H}(w_{k^*})^{k_2}$ replaced by g_{k^*} in Party 2's message in Step 4 of Round 2

where $v_{k^*} = w_{k^*}$ is the lexicographically smallest as-yet-unreplaced element of $\{v_j\}_{i=1}^{m_1} \cap \{w_i\}_{i=1}^{m_2}$, and g_{k^*} is a random element of $\mathcal{G}$.

Hyb_4 The view of P_1 output by SIM_1.

We now argue that each successive pair of hybrids in the sequence above is indistinguishable.

We first observe that Hyb_0 and $\mathsf{Hyb}_{1,0}$ are indistinguishable by the CPA security of the additively-homomorphic encryption scheme. We also observe that the pairs of hybrids ($\mathsf{Hyb}_{1,m-|J|}$, $\mathsf{Hyb}_{2,0}$), ($\mathsf{Hyb}_{2,n-|J|}$, $\mathsf{Hyb}_{3,0}$) and ($\mathsf{Hyb}_{3,|J|}$, Hyb_4) are identical.

It remains to show that hybrids of the form $\mathsf{Hyb}_{1,i-1}, \mathsf{Hyb}_{1,i}$, $\mathsf{Hyb}_{2,j-1}, \mathsf{Hyb}_{2,j}$ and $\mathsf{Hyb}_{3,k-1}, \mathsf{Hyb}_{3,k}$ are indistinguishable. We will argue that $\mathsf{Hyb}_{1,i-1}$ and $\mathsf{Hyb}_{1,i}$ are indistinguishable for all $i \in \{1, ..., m - |J|\}$, based on the hardness. We note that hybrids of the form $\mathsf{Hyb}_{2,j-1}, \mathsf{Hyb}_{2,j}$ and $\mathsf{Hyb}_{3,k-1}, \mathsf{Hyb}_{3,k}$ can be proven indistinguishable by a very similar argument.

Consider Algorithm 3 below, that takes as input a DDH tuple (g, g^a, g^b, g^c) and hybrid index i, and simulates $\mathsf{Hyb}_{1,i}$:

We observe that the output distribution produced by Algorithm 3 on input i and a DDH tuple (g, g^a, g^b, g^c) for uniformly random a, b, c is identical to $\mathsf{Hyb}_{1,i}$. To see this, we first observe that the Random Oracle has uniformly random outputs even after reprogramming, since all the reprogrammed values are random powers of a generator. Next, interpreting the hidden exponent b as P_2's key k_2, all the simulated messages sent by P_2 in Round 2 are of the correct form for $\mathsf{Hyb}_{1,i}$: un-replaced messages in Round 2 Step 2 have the form $\mathsf{H}(v_i)^{k_1 k_2}$, and messages sent in Round 2 Step 4 have the form $(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(0))$.

We now replace the DDH tuple given as input to Algorithm 3 to have the form (g, g^a, g^b, g^{ab}). The only effect is that, instead of $g_{i^*} = g^c$, we have $g_{i^*} = g^{ab} = \mathsf{H}(v_{i^*})^b$. From our earlier interpretation of b as k_2, this means $g_{i^*}^{k_1} = \mathsf{H}(v_i)^{k_1 k_2}$. This change is exactly the difference between $\mathsf{Hyb}_{1,i-1}$ and $\mathsf{Hyb}_{1,i}$. Thus, the output of Algorithm 3 on inputs i and (g, g^a, g^b, g^{ab}) is identical to $\mathsf{Hyb}_{1,i-1}$.

From the preceding argument, we can infer that if any adversary can distinguish between $\mathsf{Hyb}_{1,i-1}$ and $\mathsf{Hyb}_{1,i}$, then it can distinguish between (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c). Therefore, by the assumed hardness of DDH, $\mathsf{Hyb}_{1,i-1}$ and $\mathsf{Hyb}_{1,i}$ are indistinguishable.

Our second theorem, which we restate from Section 5.3, shows that P_2's view in the protocol Π_{DDH} can be simulated given only that P_2's input and the intersection-sum (but not the input of P_1).

Theorem 4 (Honest But Curious Security against P_2 in the DDH-based Protocol Π_{DDH}). *There exists a PPT simulator SIM_2 such that for all security parameters λ and inputs $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

$$\mathsf{REAL}^{2,\lambda}_{\Pi_{\mathsf{DDH}}}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2}) \approx \mathsf{SIM}_2(1^\lambda, \{(w_j, t_j)\}_{j=1}^{m_2}, m_1, S_J)$$

Where m_1 is the size of P_1's input, $J = \{j : w_j \in (\{v_i\}_{i=1}^{m_1}\}$ is the intersection set, and $S_J = \sum_{j \in J} t_j$ is the intersection-sum.

Algorithm 3 Simulator for $\mathsf{Hyb}_{1,i}$

Input: $(\lambda, i, (g, g^a, g^b, g^c), \{v_i\}_{i=1}^{m_1}, \{w_j\}_{j=1}^{m_2})$ **Output:** $SimView(P_1)$ in $\mathsf{Hyb}_{1,i}$
$\mathsf{SIM}_{\mathsf{Hyb}_{1,i}}(\lambda, i^*, (g, g^a, g^b, g^c), \{v_i\}_{i=1}^{m_1}, \{w_j\}_{j=1}^{m_2})$
with v_{i^*} being the new element replaced with a random one in $\mathsf{Hyb}_{1,i}$

for $i \in \{1, ..., m_1\}$ **do**
  if $v_i \neq v_{i^*}$ **then**
    Randomly sample $r_i \leftarrow \{1, , ..., |\mathcal{G}|\}$
    Program $\mathsf{H}(v_i) = g^{r_i}$
  else if $v_i = v_{i^*}$ **then**
    Program $\mathsf{H}(v_i) = g^a$
  end if
end for
for $j \in \{1, ..., m_2\}$ **do**
  if $w_j \notin \{v_i\}_{i=1}^{m_1}$ **then**
    Randomly sample $s_j \leftarrow \{1, ..., |\mathcal{G}|\}$
    Program $\mathsf{H}(w_j) = g^{s_j}$
  end if
end for
Generate key $k_1 \in \mathcal{G}$, and key-pair (pk, sk) for the additively homomorphic encryption scheme.
Send $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ in shuffled order as Party 1's message in Round 1.
for $i \in \{1, ..., m_1\}$ **do**
  if $v_i = v_{i^*}$ **then**
    $g_i \leftarrow g^c$
  else if $v_i \notin \{w_j\}_{j=1}^{m_2}, v_i < v_{i^*}$ **then**
    $g_i \leftarrow$ random element of $\mathcal{G}$
  else
    $g_i \leftarrow (g^b)^{s_i}$
  end if
end for
Send $\{g_i^{k_1}\}_{i \in [m]}$ in shuffled order as P_2's message in Step 2 of Round 2. Send $\{((g^b)^{s_j}, \mathsf{AEnc}(0))\}_{j \in [n]}$ in shuffled order as Party 2's message in Step 4 of Round 2, where each $\mathsf{AEnc}(0)$ is freshly generated.
Honestly generate P_1's message in Round 3 using P_2's dummy messages from the previous step.
Output P_1's view in the simulated execution above.

Proof. We define SIM_2 to perform the Setup phase honestly, and honestly performs the operations corresponding to P_2. SIM_2 simulates the messages sent by P_1 as follows:

- In Round 1, instead of sending $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ as P_1's message, SIM_2 instead sends m_1 randomly selected elements of $\mathcal{G}$.
- In Round 3, instead of performing the intersection and computing the intersection-sum, SIM_2 instead sends a fresh additively-homomorphic ciphertext encrypting the value S_J it received as input.

We note that the only difference between the output of SIM_2 and the view of P_2 in a real execution is in the Round 1 messages. However, the Round 1 messages output by SIM_2 can be shown to be indistinguishable from those in a real execution by using a simple hybrid argument: Define $m_!$ hybrids, where, in each successive hybrid, SIM_2 replaces one additional "real" Round 1 message of the form $\mathsf{H}(v_i)^{k_1}$ with a random element of $\mathcal{G}$. Then, each pair of neighboring hybrids can be shown to be indistinguishable based on the fact that k_1 is secret and that DDH is hard in $\mathcal{G}$. The details are very similar to the proof of Theorem 3, and we leave them as an exercise.

D "Reverse" variants

We note that, in each of the protocols described in Section 5, a specific party receives the intersection-sum as output, namely P_2, the party who has the associated values as input. Additionally, each of the protocols has the property that the other party (P_1) performs the homomorphic addition of associated values. Furthermore, in each protocol, P_1 also learns the *size* of the intersection before performing the homomorphic addition, and this enables us to give P_1 the option to abort the protocol if the intersection-size is too small, without P_2 learning the intersection-sum.

In some cases, we may want the reverse configuration, namely that P_1 learns the intersection-sum, and P_2 has the ability to abort before the intersection-sum is learned if the intersection-size is too small. It turns out that there is a straightforward modification that can be made to each of our protocols that allows this alternate configuration, which we call a "reverse" variant. The idea is as follows

- P_2 uses an additive-homomorphic encryption scheme to encrypt each of its associated values t_j using its encryption key pk_2, and sends these encrypted values to P_1.
- P_1 homomorphically masks each associated value with a random additive mask r_i, and sends the resulting ciphertexts (rerandomized) to P_2 in shuffled order.
- P_2 decrypts the ciphertexts to recover the masked associated values. P_2 then adds together the values that were determined to be in the intersection. (Which values are to be added is determined differently in each protocol,

depending on the underlying PSI technique used in that protocol, namely DDH, Random-OT or Encrypted Bloom Filter.) If the intersection-size is too small, P_2 can abort, otherwise P_2 sends the masked sum to P_1, together with the set of indices that were in the intersection.
- P_1 uses the indices received in the previous step to determine which masks must be removed from the masked sum. P_1 subtracts these masks from the masked sums, and outputs the recovered value as the intersection-sum.

We present the concrete instantiation of the reverse variant of the DDH-based protocol in Appendix D, together with a short security sketch drawing on the security argument for the forward variant. We omit the detailed descriptions for the reverse variants for the Random-OT and Bloom-filter based protocols, noting that they can be constructed in a straightforward way using the recipe described above.

D.1 The "Reverse" DDH-based Protocol

The DDH-based Private Intersection-Sum protocol Π_{DDH} we presented in Section 5.3 can be modified in a straightforward way to allow both parties to learn the intersection-sum or intersection-size. It is also possible to ensure that one or the other party performs the actual intersection operation, for example, to allow that party to abort if the intersection is below some threshold, which might be imposed for policy reasons. We present one such variant in Figure 6, which we refer to as the "reverse" protocol. In this protocol, P_2 performs the intersection, and can abort the protocol if the intersection size is too small, without either party learning the intersection-sum. In addition, both parties learn the intersection size, but only P_1 learns the intersection-sum. To implement this, we additionally need P_1 to blind the additively homomorphic ciphertexts with random masks, and for these masks to be removed after the masked associated values have been added. The details can be seen in Figure 6.

D.2 Security Analysis

The security proof for the reverse variant of the DDH-based protocol is similar to the "forward" DDH-based protocol, but with the roles of the parties reversed, with Party 2 learning only the intersection size, and Party 1 learning both the intersection size and the intersection sum. The simulator for Party 1 is almost identical to the simulator for Party 2 in the original protocol; the one difference is that the simulator must also provide indices J (known to the simulator during the course of simulation) in Round 3 to allow Party 1 to compute $S_J - \sum_{j \in J} r_j$. For Party 2, the simulator is very similar to the original Party 1 simulator SIM_1 in Algorithm 2. We omit details of simulators and the hybrid security argument.

Reverse Intersection-Sum Protocol

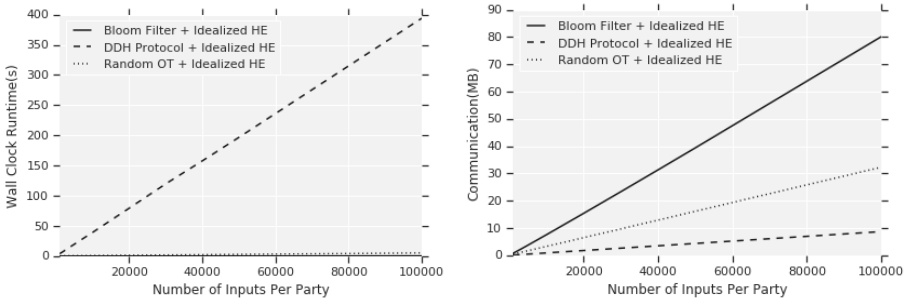
- **Inputs**:
  - Both parties: A group $\mathcal{G}$ of prime order, and an identifier space $\mathcal{U}$. A hash function $\mathsf{H} : \mathcal{U} \to \mathcal{G}$, modeled as a random oracle, that maps identifiers to random elements of $\mathcal{G}$.
  - P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$, where $v_i \in \mathcal{U}$.
  - P_2: Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$, with $w_i \in \mathcal{U}$, $t_i \in \mathbb{Z}^+$.
- **Setup**:
  - Each P_i chooses a random private exponent k_i in the group $\mathcal{G}$.
  - $Party_2$ generates a fresh key-pair $(pk, sk) \leftarrow \mathsf{AGen}(\lambda)$ for the additive homomorphic encryption scheme and sends the public key pk with $Party_1$.
- **Round 1 (P_2)**:
  1. For each element (w_j, t_j) in its set, P_2 applies the Random Oracle and then single-encrypts w_j using its key k_2, thus computing $\mathsf{H}(w_j)^{k_2}$.
  2. P_2 sends $\{(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))\}_{j=1}^{m_2}$ to P_1 in shuffled order.
- **Round 2 (P_1)**:
  1. For each element $(\mathsf{H}(w_j)^{k_2}, \mathsf{AEnc}(t_j))$ received from P_2 in the previous step, P_1 double-encrypts them using its key k_1 and homomorphically computes a one-time pad encryption of t_j under addition in the message space $\mathcal{M}$ of the additively-homomorphic encryption scheme, computing $(\mathsf{H}(w_j)^{k_1 k_2}, \mathsf{AEnc}(t_j + r_j))$.
  2. P_1 sends $\{(\mathsf{H}(w_j)^{k_1 k_2}, \mathsf{AEnc}(t_j + r_j)\}_{j=1}^{m_2}$ to P_2 in shuffled order. The (shuffled $j \to r_j$) map is saved for a future step.
  3. For each item v_i in its input set, P_1 applies the Random Oracle to the first element of the pair and encrypts it using key k_1. It encrypts the second element of the pair using the key pk for the additively homomorphic encryption scheme. It thus computes the pair. $\mathsf{H}(v_i)^{k_1}$.
  4. P_1 sends the set $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ to P_2 in shuffled order.
- **Round 3 (P_2)**:
  1. For each item $\mathsf{H}(v_i)^{k_1}$ received from P_1 in Round 2 Step 4, P_1 double-encrypts the using k_2, thus computing $\mathsf{H}(v_i)^{k_1 k_2}$.
  2. P_2 computes the intersection set J:
  $$J = \{j : \mathsf{H}(w_j)^{k_1 k_2} \in \{\mathsf{H}(v_i)^{k_1 k_2}\}_{i=1}^{m_2}\}$$
  3. For all items in the intersection, P_2 decrypts $\mathsf{AEnc}(t_j + r_j)$ and adds the associated (one-time pad encrypted) ciphertexts, computing a ciphertext encrypting the intersection-sum $S_J = \sum_{j \in J} t_j + r_j$
  4. P_2 sends S_J together with the indexes J corresponding to the additively homomorphic ciphertexts in the intersection, to P_1.
- **Output (P_1)**: P_1 computes $S_J - \sum_{j \in J} r_j$ to recover $\sum_{j \in J} t_j$.

Fig. 6: Detailed description of the "Reverse" Private Intersection-Sum protocol.

E Additional Measurements

E.1 Using an idealized homomorphic encryption scheme

In Table 5 and Figure 7, we show communication and computation costs for each of the 3 protocols we present, assuming an "idealized" homomorphic encryption scheme that incurs no computational cost and no communication overhead as compared to sending plaintexts. We see that in these schemes, the Random OT protocol and Bloom-Filter protocol gain a large computational edge over the DDH-based protocol. We interpret this as an indication that it may be beneficial to explore further optimizations to the additive homomorphic encryption scheme.



(a) Comparison of Computation Costs for Intersection Sum with Idealized Homomorphic Encryption.

(b) Comparison of Communication Costs for Intersection Sum with Idealized Homomorphic Encryption.

Fig. 7: Compares computation and communication costs for "idealized" variants of intersection-sum, assuming and additive homomorphic encryption schemes which have no computational cost, and ciphertexts that are the same size as the plaintext. Wall-clock running times are the totals across both parties, excluding network transfer time. Communication costs are also totals for both parties.

In this section, we present some additional measurements for the Random-OT based protocol (Section 5.1), Figure 1) using a different combination of additive homomorphic encryptions.

E.2 Using different additive encryption schemes for the Random-OT-based Protocol

In particular, we measure the computation and communication when we use Ring-LWE for encrypting both $\mathsf{L}_i[v_i]$ values, and encrypting the associated values. We use two different sets of RLWE parameters for the two different use-cases.

	DDH + Idealized HE		Random-OT + Idealized HE		Bloom Filter + Idealized HE	
Input Size	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]
1000	4.01	0.09	0.61	0.41	0.00	0.71
2000	7.83	0.17	0.66	0.73	0.01	1.45
3000	11.81	0.26	0.70	1.04	0.01	2.19
4000	15.80	0.35	0.75	1.36	0.01	2.95
5000	19.69	0.43	0.79	1.68	0.01	3.71
10000	39.29	0.87	1.02	3.27	0.03	7.56
20000	79.38	1.74	1.53	6.47	0.07	15.39
30000	119.70	2.60	1.98	9.68	0.10	23.32
40000	157.71	3.47	2.43	12.90	0.14	31.32
50000	196.61	4.34	2.97	16.12	0.18	39.37
100000	393.84	8.68	5.31	32.28	0.37	80.12

Table 5: Comparison of the protocols for Intersection Sum, assuming the existence of an ideal additively-homomorphic encryption scheme, with no computation cost or ciphertext expansion. We assume this scheme is used for encrypting not just associated values, but also the $\mathsf{L}_i[v_i]$ values in the Random-OT variant, and the bits of the Bloom filter in the Bloom-filter variant.

For encrypting the associated values, we use the same parameters as discussed in Section 8, namely slotted RLWE encryption with an 80 bit modulus and 2048 coefficients, with 32 bit plaintext space.

For encrypting $\mathsf{L}_i[v_i]$ values, we use RLWE encryption with a 14-bit modulus, 1024 coefficients, and a plaintext modulus of 2, corresponding to 1 bit per coefficient. We encrypt each $\mathsf{L}_i[v_i]$ value in a different ciphertext. To be encrypted, each 256 bit $\mathsf{L}_i[v_i]$ value is decomposed into 256 single-bit values that are inserted into the first 256 coefficients. The masks are homomorphically added to the ciphertexts, along with 12 bits of additional noise, leading to a 2^{-9} statistical hiding of the error. (We note that this is less statistical hiding, but since we are only hiding a single homomorphic addition, this is enough to drown out the original error. We defer a detailed analysis.)

We present the measurements in Table 6. For easy comparison, we also present the costs previously shown in Table 3, namely the cost when using unslotted-Paillier to encrypt both $\mathsf{L}_i[v_i]$ and the associated values, and also the costs from Table 4, which shows the cost when using unslotted-Paillier to encrypt the $\mathsf{L}_i[v_i]$ and slotted-Ring-LWE to encrypt the associated values. We refer to the three variants as Paillier-Paillier, Paillier-RLWE and RLWE-RLWE, with the first part representing the scheme used to encrypt which shows the cost when using unslotted-Paillier to encrypt both $\mathsf{L}_i[v_i]$ and second part representing the scheme used for the associated values.

We note that using RLWE for both the $\mathsf{L}_i[v_i]$ values and the associated values nearly halves the computational cost of the scheme, while significantly increasing communication costs. However, the total computation is still about 20% more than the DDH + RLWE protocol (see Table 4).

	Paillier-Paillier		Paillier-RLWE		RLWE-RLWE	
Input Size	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]	Time(s)	Comm.[MB]
1000	11.31	1.64	9.89	1.33	5.15	7.07
2000	21.82	3.20	19.61	2.52	9.71	14.07
3000	32.31	4.76	29.66	3.74	14.28	21.12
4000	43.28	6.32	39.33	4.94	19.00	28.17
5000	53.83	7.89	48.72	6.16	24.22	35.26
10000	107.91	15.75	97.67	12.22	46.99	70.79
20000	216.57	31.51	196.86	24.55	94.73	142.30
30000	325.42	47.36	294.95	36.73	140.37	214.13
40000	434.27	63.22	399.50	49.02	187.31	286.17
50000	543.68	79.09	492.17	61.34	245.57	358.36
100000	1,115.10	158.74	989.88	123.21	554.75	720.83

Table 6: Comparison of variants of the Random-OT based Private Intersection-Sum protocol, using different homomorphic encryption schemes to encrypt the $\mathsf{L}_i[v_i]$ values and the associated values.

F Comparison of Garbled-Circuit style approaches

In this section, we reproduce the table of [11] that provides an excellent analytical comparison of the communication and computation costs of different Garbled-circuit style protocols for privately computing generic functions over an intersection. This table appears as Table 1 in [11]. The specific private function considered by [11] in creating this table is "Private Set Membership with encrypted output", which is a building block for privately computing any generic function over the intersection.

They key high-level takeaway from this table from the point of view of our work is that the communication cost of each protocol is $O(\lambda Ms)$, that is, it depends on the product of the security parameter, the set size and, crucially *the bit length of each identifier*. This asymptotic communication complexity also appears in garbled-circuit style solutions for functionalities beyond Private Set Membership, and is roughly due to the need to separately encrypt each bit of the identifiers.

	# of sym. key operations	Communication bits]
Yao SCS [29]	$12\lambda M + 3\lambda M$	$2\lambda Ms(1 + 3\log M)$
GMW SCS [29]	$12\lambda M \log M$	$6\lambda M(s + 2)\log M$
Yao PWC [46]	$4\lambda M + 6393\lambda$	$\lambda(M3s + 3198s + 15)$
GMW PWC [46]	$6\lambda M + 9594\lambda$	$\lambda(M4{+}6396{+}2sM{+}6396s)$
Graph Navigation [11]	$4\lambda M + 3\lambda$	$2\lambda Ms + Ms$

Table 7: (Table 1 from [11]) Computation and communication complexity comparison for the PSM case. M represents the size of the set, s is the security parameter and λ is the bit-length of each element.

To clarify the comparison to our works, compare to Table 1. Table 1 shows that our protocols all have communication dominated by the size of group elements and homomorphic ciphertexts, and the size of the sets held by each party. Our protocols do also depend in subtle ways on the bit length of the identifiers, for example, the parameters of homomorphic encryption may have to be increased if the identifiers are very large. However, of the protocols presented in this work, none have communication complexity with a *multiplicative* dependency on the bit length of the identifiers in the way that garbled-circuit style protocols do.