On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality

Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel,
Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, Moti Yung
{mion, benkreuter, anergiz, sarvar,
shobhitsaxena, karn, marianar, dshanahan, moti}@google.com
Google LLC

***Abstract*—In this work, we discuss our successful efforts for industry deployment of a cryptographic secure computation protocol. The problem we consider is privately computing aggregate conversion rate of advertising campaigns. This underlying functionality can be abstracted as Private Intersection-Sum (PI-Sum) with Cardinality. 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 the number of identifiers they have in common and the sum of the integer values associated with these users without revealing any more information about their private inputs.**

We identify the major properties and enabling factors which make the deployment of a cryptographic protocol possible, practical, and uniquely positioned as a solution for the task at hand. We describe our deployment setting and the most relevant efficiency measure, which in our setting is communication overhead rather than computation. We also present a monetary cost model that can be used as a unifying cost measure and the computation model which reflect out use-case: a low-priority batch computing.

We present three PI-Sum with cardinality protocols: our currently deployed protocol, which relies on a Diffie-Hellman style double masking, and two new protocols which leverage more recent techniques for private set intersection (PSI) that use Random Oblivious Transfer and encrypted Bloom filters. We compare the later two protocol with our original solution when instantiated with different additively homomorphic encryption schemes. We implement our constructions and compare their costs. We also compare with recent generic approaches for computing on the intersection of two datasets and show that our best protocol has monetary cost that is $20\times$ less than the best known generic approach.

1. Introduction

Secure multiparty computation (MPC) has long held the promise of enabling joint analysis on data coming from multiple sources, while maintaining the privacy of each input source. Recent developments in the field over the past decade have demonstrated impressive efficiency improvements, showing that that this promise stretches beyond feasibility results and into real implementations. Yet, the adoption of this technology in real industry settings has been very limited.

In this paper we present our work on using cryptographic MPC techniques for a particular business application: *attributing aggregate ad conversions*. Our solution has been used in practice for several years and in this paper we discuss the full spectrum of constraints we encountered for the deployment and use of an MPC solution for this problem and how these influenced the cryptographic protocol that we chose to implement. Our application problem has similarities to the problem of Private Set Intersection (PSI) but the exact functionality that we need is an extension of the PSI problem, which we call *private set intersection sum with cardinality*. However, since there has been a tremendous progress in the state of the art in PSI [11], [31], [37], [47], [49], [51], [55], often motivated with our business scenario as a practical application example, in this work we consider recent efficient PSI construction approaches and how they can be extended to solutions for our setting. We compare the efficiency cost of these constructions to that of our original solution in the context of the *monetary cost* of resources when executing in a low-priority "batch-computing" setting, where the results are not needed in real-time. We model monetary costs by using the prices charged by cloud providers for compute and network resources, which we found to be a good approximation of costs in our deployment environment. This constraint is different from that considered by most works in secure-computing space, which are generally focused on minimizing end-to-end runtime. [1]

Ad Conversion. An ad conversion occurs when a user sees an online ad for a particular company on some website, and later makes a purchase in the company's store. The company placing the ads would like to know how much of its revenue it can attribute to its online ad campaign. However, the data needed to compute these attribution statistics is split across two parties: the ad supplier, which knows the users who have seen a particular ad, and the company, which knows who made a purchase and what they spent. The two parties may be unwilling or unable to expose the underlying data, but both parties would still like to compute an aggregate population-level measurement: how many users both saw an ad and made a corresponding purchase, and how much those users spent in total. They wish to do this while making sure that nothing beyond these aggregate values is revealed about individual users in the input data sets.

Private Intersection-Sum with Cardinality. Abstractly, the above problem can be viewed as a variant of the Private Set Intersection problem, which we call the *Private Intersection-Sum with Cardinality* problem. In this setting,

1. A recent exception is the work of [48], which explicitly considers the related problem of PSI, using monetary cost as an efficiency measure.

there are two parties that have private input sets consisting of identifiers and one of the parties has additionally an integer value associated with each identifier. The parties want to learn the *sum* of the associated integer values for all identifiers in the intersection of the two input sets as well as the cardinality (or size) of the intersection, but nothing beyond that. In particular, neither party should learn the actual identifiers in the intersection, nor should they learn any additional information about the other party's data (beyond the size of their input set) such as associated values at finer granularity than the aggregate over the intersection. Additionally the parties could choose to reveal the intersection-sum conditioned on minimal threshold size of the intersection.

For the ad conversions measurement use case, the identifiers held by one party correspond to users who have seen the advertising campaign, while the identifiers and integer values held by the other party correspond to the users who bought the related item and how much they spent, respectively. While the ad conversion problem is one example scenario that can be solved using a Private Intersection Sum protocol, this functionality is applicable much more broadly. It could be easily extended to other statistics such as average and variance. More generally, a PSI-Sum protocol provides a secure solution for any setting where one party holds a private statistic about users, and another party holds private knowledge about the membership of users in a particular population, and both parties want to learn the aggregate statistic for that particular population. For example, one could answer questions like ”What is the average blood pressure of patients taking a particular medication?” or ”What is the home-ownership rate of people living in a particular zip code?”.

Contributions. Our contributions are two-fold. First, we give a detailed description of our deployment, including constraints and limiting factors for our choice of solution. This includes discussion of the features of the *secure aggregate ad conversion* business problem, that made it a good candidate for a cryptographic MPC-based solution. These factors include the privacy needs and the strong business benefit, but also the tractability of the problem using efficient techniques. Key constraints for the solution are simplicity and explainability, along with ease of implementation, ease of maintenance, and extensibility.

We discuss the details of our protocol deployed, which is based on the classic set-intersection protocol of [41], 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 [34]. The aggregation property is achieved through the use of additively homomorphic encryption. We describe typical daily workloads, and the costs for running those workloads, including *monetary costs*. We observe that our application is not real-time and can use cheap low-priority, pre-emptible computation. In this “batch-computing” setting bandwidth remains a scarce shared resource for which multiple applications compete (see Table 1 for resource prices of various cloud providers). Thus, communication complexity becomes the most relevant efficiency measure for our protocols. We believe that our insights will be useful to those trying to bring secure computing techniques to use in practice.

As our second main contribution, we revisit the question of Private Intersection-Sum with Cardinality through the lenses of the large number of recent papers improving the state of the art in efficient PSI protocols [11], [31], [37], [47], [49], [51], [55], in order to compare the efficiency costs of potential new solutions against the DDH-style protocol used in our deployed implementation Our goal is to see whether more modern techniques can lead to significant savings at the cost of increased complexity. To this end we compare costs against a suite of recent works based on garbled circuit-style techniques [11], [31], [32], [51], which explicitly consider privately computing functions on an intersection, and can be directly extended to compute our functionality. We also examine recent approaches for computing Private Set Intersection [17], [20], [22], [37], [47], [49], and try to adapt them to compute private intersection-sum-with-cardinality.

While initially the Private Intersection-Sum-with-Cardinality functionality looks very close to the 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. As such, recent PSI approaches [17], [20], [22], [37], [47], [49] require significantly different techniques in order to be adapted to the setting of PSI-Sum with cardinality. We focus our adapting efforts on two major approaches from these works. The first one is based on Random Oblivious Transfer, and builds on techniques developed by [49] and [37]. 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, as in our deployed protocol. The second protocol is based on Encrypted Bloom Filters and is inspired by several recent PSI solutions [17], [20], [22]. 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 these two new constructions, we give a *recipe* for transforming a common form of PSI protocol into one for computing PSI-Sum-with-Cardinality. We call this recipe “Tag, Shuffle, Aggregate”, and believe that it applies to a broad class of newer approaches.

Furthermore, in the two new Private Intersection-Sum with Cardinality protocols that we summarize above, as well as our deployed protocol, the output recipient is the party who has input set of pairs of identifiers and values. We also construct “reverse” variants of the protocols, which change the output recipient to be the party that has input set only with identifiers. All our constructions implement secure computation functionalities with security against semi-honest adversaries.

All three protocols use additively homomorphic encryption (HE) as a building block. While our deployed implementation used Paillier [46] encryption with optimizations due to Damgard-Jurik [16], which offered best efficiency guarantees at the time, there has been significant improvement in the efficiency of lattice-based constructions [8], [9], [24], [29] recently. We consider these two schemes together with the third existing additively homomorphic encryption scheme, ElGamal [23], to conduct a comprehensive comparison and to analyze the trade-offs for the efficiency of our protocols when instantiated with each of the additive HE options.

We implement and benchmark each of the protocols we describe, and compare their performance in computation, communication and monetary cost with each other, and also with the recent protocols using garbled-circuit-style techniques for computing generically on the intersection of two datasets [11], [31], [50]. We find that the deployed DDH-style protocol with Paillier as homomorphic encryption achieves lowest monetary cost among the protocols, requiring 0.084 cents(USD) on input sets of 100,000 elements. We also found that our variant of DDH-style protocol for PI-Sum with cardinality using RLWE homomorphic encryption for the associated values takes about 40% less time (47.4 seconds versus 74.4 seconds) on sets of size 100,000, with a 11% smaller monetary cost of 0.075 cents. Additionally, all of our protocols (DDH, ROT and Bloom Filter) outperform the generic approaches of [11], [31] in terms of *monetary cost*. The approach of [50] outperforms our solution based on Bloom Filters, but is more expensive monetarily than the solutions based on ROT and DDH. Overall, the deployed DDH protocol remains the best option for our setting under the constraints that we have identified. Specifically, the monetary cost of the deployed DDH-based protocol is $20\times$ cheaper than any of the novel generic approaches based on garbled circuits.

Roadmap. In Section 2 we discuss in detail the practical setting for our problem. In Section 3, we give details of the protocol we deployed.In Section 4 we revisit the problem of Private Intersection-Sum-with-Cardinality in light of recent progress in PSI. We give an overview of recent PSI approaches and propose a recipe for transforming PSI protocols into Private Intersection-Sum-With-Cardinality in Section 4.2. In Section 4.3 we present two new constructions using novel techniques. All our constructions use several cryptographic primitives defined in Appendix A. In Section 5 we present measurements from our protocol implementations, including comparisons based on monetary costs with works based on garbled-circuit techniques. In Appendix A we give various useful notation and definitions. In Appendix B we discuss the tradeoffs of using different encryption schemes to instantiate our protocols.

2. Problem Setting

We now describe details of the problem and of our deployment environment including the formal problem statement and its characteristics, our execution environment and estimates of daily workloads, as well as the monetary cost model we use for our protocols.

2.1. Formal Problem Statement

In Figure 1 we give a formal description of the functionality we are aiming to compute securely. It captures that we want Party P_2 to learn the intersection-sum S, and that we allow both parties to learn the intersection size C. In Appendix G, we also describe a Reverse variant, where P_1 learns the intersection-sum S instead of P_2.

Our deployment also assumes that the parties already have data with the same identifier space. Finding an appropriate common identifier space is an important problem that is orthogonal to this current work.

Private Intersection Sum with Cardinality

Inputs:

P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$ $\qquad$ P_2 : Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$

Outputs:

$\mathsf{P}_1 : C = |\{i : w_i \in V\}|$ $\quad$ $\mathsf{P}_2 : C = |\{i : w_i \in V\}|, S = \sum_{i : w_i \in V} t_i$

Figure 1: F_{PIS-C} : The Private Intersection-Sum with Cardinality functionality.

Protection of Individual Values. The PSI-Sum functionality reveals aggregates of values for the records in the intersection. If there are no outlier values, then this aggregates provides a high level of privacy protection of the individual values that were added. One approach to protect outliers that may be revealed by the sum is to remove outliers from the input dataset. A similar issue occurs with lower-order digits for which we can use rounding. For many application scenarios, these protections may provide sufficient privacy guarantees. For setting where these do not suffice, the parties can add differential privacy noise [21] to the output of the computation.

2.2. Problem Characteristics and Constraints

In this section, we discuss various features of the business problem our deployment addresses, namely computing ad-conversion metrics. We first consider the characteristics that make the problem a good fit for a secure-computing solution.

Strong business benefit: Given that cryptographic solutions add significant complexity and efficiency costs, a clear business benefit is a prerequisite to exploring such solutions. The measurement of ad campaign conversions is of central importance to advertisers, enabling them to deploy their marketing budget more efficiently.

Strong privacy need: For our application, the data needed to compute 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.

Tractable: The underlying problem must be amenable to efficient solutions. Existing efficient private set intersection solutions were a good sign that the private intersection-sum problem underlying our application will be tractable.

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.

Design Constraints: Next we turn to the various design constraints for our problem. We note that our context is secure computation across multiple businesses (over Wide Area Network–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: Perhaps 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 (WAN solutions). Network cost is an order of magnitude more expensive than computation, and this is reflected in the costs charged by cloud providers (see Table 1).

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.

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.

Flexibility: Any successful deployment may drive the demand for solutions to related problems. A modular design relying on APIs that can be reused in solutions to a whole class of problems, is always desirable.

2.3. Adversarial Model

Another important design axis is the class of misbehavior the secure computation should protect against.

There are various different models in which we can design our protocol, where the two extremes are the *fully-trusted* and the *malicious* adversary models. Full-trust implies that parties trust each other to compute correctly on data in the clear, and to delete the data after the end of the protocol. On the other hand, malicious adversaries are allowed to deviate arbitrarily from any prescribed protocol. Intermediate notions exist between these two extremes, an important one being security against honest-but-curious adversaries. Protocols that provide "honest-but-curious" (or semi-honest) security, (modeled in [58]) assume that the participants will follow the protocol steps honestly, but may try to learn as much as possible from transcript and logs of the various protocol messages.

While protection against stronger adversaries is naturally more appealing, it does come with substantial efficiency cost (especially communication). Furthermore, the security model for a cryptographic protocol is one piece of a comprehensive risk analysis which would include incentives of parties to deviate, the nature of the underlying data, the size and level of trust among interacting parties, and the availability of external enforcement mechanisms such as contracts or code audits. Compared to protocols in the malicious security model, the semi-honest model has relatively efficient solutions available, especially in terms of communication cost and monetary cost. It is also a significant improvement over the fully-trusted model which relies on external non-cryptographic mechanisms to ensure privacy. In particular, the semi-honest model gives strong privacy protections against data breaches in either party, since, since semi-honest protocol logs leak nothing beyond the prescribed protocol output.

In our deployment, with all factors considered, we chose to target "semi-honest" security.

2.4. Workloads and Execution environment

A typical daily workload for our deployment involved around 1000 protocol executions between each pair of parties, with each execution involving on the order of $100,000$ items in each party's input.

Our application is not real-time, and therefore our deployment environment is representative of a latency-insensitive batch-computation. Each party controls its own set of computational and storage resources, and in addition, both parties have access to shared storage resources that are used transferring data from the intermediate steps of the protocol execution. Each of the 1000 protocol executions is assigned a separate directory in the shared storage. Whenever a party finishes a round for a particular execution, it writes the round result into the shared directory for that execution. Each party continuously polls these directories in the shared storage with a relatively long delay (on the order of minutes) to check if a new file has been written, and if so, performs the next round of the corresponding protocol execution. The executions continue in this way until all executions have completed.

The parties execute their portions of the protocol in their own datacenters. The intermediate protocol data is transferred between these datacenters using a standard SSL connection over the internet.

2.5. Monetary Cost Model

We use the total monetary cost of a protocol execution as our primary cost measure.

Why monetary cost: Our business use-case is not latency sensitive: the output statistics are only needed for generating periodic reports. Therefore, the traditionally used end-to-end running time of executions is not the most important measure of cost (as long as all executions can finish within a day). Rather, the cost of resources (CPU, RAM and especially network) are most relevant. Monetary cost gives a convenient measure to unify the cost of these various resources in a practically relevant way.

	CPU ($/hr)	Network ($/GB)
Google Cloud Platform (GCP)	$0.01	$0.08
Amazon Web Services (AWS)	$0.005	$0.08
Microsoft Azure	$0.005	$0.083

TABLE 1: Computation and Network Costs, as charged by different cloud providers. Computational costs are for a single pre-ordered pre-emptible CPU and 2-4 GB of RAM. Network costs are the cost for the 10-50TB tier of Cloud-to-Internet egress traffic for each provider. This table is representative of pre-planned, low-priority resource costs.

Specific costs used: For our concrete monetary costs of resources, we used the costs charged by Google Platform for pre-emptible virtual machines (including CPU and RAM), and the cheapest cost for "Internet Egress" network usage (representing data flowing to an external datacenter or cloud provider).

These costs are in a similar range to the bulk costs charged by the other cloud providers, as shown in Table 1 [2]. The prices shown are for single-CPU machines with 2GB RAM in AWS and Azure, and 3.75 GB RAM in Google Cloud. The network bandwidth costs are for internet egress in the 10-50TB data transfer tier.

A key feature of monetary cost as a metric compared to end-to-end running time is that it gives a large weight to communication cost. A gigabyte of communication would add only minutes to end-to-end running time, but its monetary cost is equivalent to 8 hours of computation.

Justification: Even though our execution environment involves parties running the protocol in their own datacenters, the cloud pricing model closely reflects the relative costs of resources for our environment as well. In fact, communication was even more lopsidedly constrained in our deployment scenario. This reflects the fact that network bandwidth is a highly constrained resource, and that it is much cheaper to add more processors to a datacenter than to increase the total bandwidth leaving a datacenter.

Alternative cost models: In our experience, the cost model we use closely represents the costs of executing protocols datacenter-to-datacenter, and also between parties that are hosted in different cloud providers. While we are not aware of business-to-business secure computation deployments beyond our own, we expect the datacenter-to-datacenter and cloud-to-different-cloud to be the most common deployment scenarios, and therefore using cloud costs would be a good fit there as well.

Network costs become cheaper if the two participants use the same cloud provider and are colocated in the same cloud region, which is not the case for our deployment. For example, network transfer in the same region costs 0.01 per GB within GCP, which is $8\times$ cheaper than the internet egress rates. In this setting protocols with different efficiency tradeoffs may have better monetary cost.

2. Google Cloud Platform Costs are from https://cloud.google.com/compute/all-pricing and https://cloud.google.com/compute/network-pricing respectively. AWS costs are for "spot" instance CPUs from https://aws.amazon.com/ec2/spot/pricing/ and network costs are from https://aws.amazon.com/cloudfront/pricing/. Azure CPU costs are from https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/ and network costs are from https://azure.microsoft.com/en-us/pricing/details/bandwidth/.

2.6. Related Works

Deploying secure computation in practice is challenging, but there are some real-world applications leveraging MPC. These include the Danish Sugar Beet Auction [6], a financial application in Estonia [5], a secure survey of faculty salaries at universities in Massachusetts [39], and a platform for reporting harassment [54]. More recently, Lindell et al. [2], [28] deploy secure computation for cryptographic operations running between machines owned by a single organization. Google also reports using a secure computation protocol to aggregate locally trained machine learning models from mobile devices [7]. However, routine use by organizations of secure computing between multiple entities or businesses remains rare.

3. Deployed Protocol

In this section, we describe our deployed protocol, namely the protocol based on DDH.

3.1. Protocol details

Here we present our DDH-based intersection-sum protocol which we designed in response to the above constraints. Our protocol was, in our understanding, the *simplest* protocol that solves the intersection-sum problem. It uses well-understood cryptographic primitives that are based on classical assumptions, and can be built from widely-deployed libraries (e.g. OpenSSL). It targets semi-honest security, and aims to minimize communication cost. Furthermore, it is flexible enough to allow several important variants.

Our protocol extends an existing PSI protocol [41]. It uses the following primitives defined in Appendix A: a group $\mathcal{G}$ in which the DDH assumption holds, an additively homomorphic encryption scheme $(\mathsf{AGen}, \mathsf{AEnc}, \mathsf{ADec})$ and a hash function H modeled as a random oracle for the security proofs. We present our construction in Figure 2.

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 $H(x)^k$ [34] where the key is multiplicatively shared, or evaluating deterministic Pohlig–Hellman cipher [30] 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 2 that $\mathcal{G}$ has prime order.

Our protocol extends ideas from the work of of Meadows [41] on private matching and Huberman [33] 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

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:
  $$\mathsf{AEnc}(pk, S_J) = \mathsf{ASum}\left(\{\mathsf{AEnc}(t_j)\}_{j \in J}\right) = \mathsf{AEnc}\left(\sum_{j \in J} t_j\right)$$
  P_1 then randomizes the ciphertext using $\mathsf{ARefresh}$ and sends it to P_2.
- **Output** (P_2): P_2 decrypts the ciphertext received in Round 3 using the secret key sk to recover the intersection-sum S_J.

Figure 2: Π_{DDH}: Our deployed DDH-based Private Intersection-Sum protocol.

sent back to the first party, who then decrypts to recover the intersection-sum.

Correctness of the protocol follows immediately from inspection, namely that P_2 learns the intersection-sum and cardinality.

We state the theorems for security below. The formal security proof in the honest-but-curious model appears in Appendix D. They show that each party learns nothing more than the intersection size and intersection-sum. The proofs use a standard hybrid argument to show that the view of each party can be simulated by replacing "real" protocol values with dummy values while maintaining the same intersection size and sum, and that the dummy values are indistinguishable from the real values by the assumed hardness of DDH and the security of the homomorphic encryption scheme.

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

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

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

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

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.*

3.2. Parameter Choices

For our deployment, we use OpenSSL's implementation "prime256v1", a NIST elliptic curve with 256-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 [3]. In order to simulate a new random oracle for each execution, the parties choose a common random seed and prepend it to each input before hashing.

For the additive homomorphic encryption scheme, we use Paillier encryption with 768 bit primes, with the Damgard-Jurik optimization [16] with $s = 3$. Thus each ciphertext is 6144 bits, with 4608 bits of plaintext space. We assume that each of the associated integer values is at most 32 bits, including after summation, and therefore we can pack many values into a single ciphertext. Details of slotting can be seen in Appendix B.1. Allowing 40 bits between slots for masking, and leaving half the plaintext space empty to allow slots to be shifted-and-added, we can pack 65 integer values into a single ciphertext. Our Paillier implementation was built on top of OpenSSL's BigNum libraries.

3.3. Deployment Costs

In Table 2, we give the communication and computation costs for both a single session, with 100,000 entries in each party's database, and also for 1000 such sessions, which corresponds roughly to a days workload. In addition we give the dollar cost using resource valuations for GCP given in Table 1. The runtimes are the total for both parties, and are measured on a single core of a desktop workstation with an Intel Xeon CPU E5-1650 v3 (3.50 GHz), which is representative of a typical machine in our deployment, and also representative of a virtual CPU in Google Cloud Platform.

We see that even though the raw computation time appears to be quite large, and the communication relatively modest, communication is a factor of $3\times$ more expensive in dollar cost, and constitutes 75% of the monetary cost of the protocol.

3.4. Variants

We considered variants of this protocol in different contexts. These variants give modifications of the functionality that offer important flexibility.

Threshold variant: One variant is a simple threshold-variant, which allowed P_1 to abort the protocol in Round 3 if the intersection size was too small, before the intersection sum was computed.

Reverse variants: The protocol in Figure 2 has P_1 perform the intersection and homomorphic intersection sum in Round 3. The protocol can be reversed so that P_2 instead performs the intersection and intersection sum, and P_1 receives the output. To achieve this, P_1 in Round 3 additionally additively masks each of the encrypted associated values and shuffles and sends them to P_2 together with the shuffled set $\{\mathsf{H}(w_j)^{k_1 k_2}\}$. P_2 then decrypts and adds together the values corresponding to the masked shuffled-intersection. P_1 can then unmask the intersection using it's knowledge of the masks. Note that this variant is not compatible with packed additive homomorphic encryption.

Segmented input: In some settings, it is desirable to split P_1's input into several disjoint segments, and for P_1 to learn the intersection sum for each of these segments. For example, in the advertising conversion measurement use-case, the segment could correspond to different age groups, or different geographical regions. One way to achieve this is to run a separate protocol execution for each segment held by P_1. However, this adds significantly to the cost. A simple alternative is for P_1 to send its values in Round 1 separated by segment, and for P_2 to shuffle values only within a segment, but not across different segments. This enables P_1 to compute the intersection size and encrypted sum for each segment, with roughly the same cost as a single unsegmented execution.

Multiple integer values: The protocol generalizes easily to cases where each item in P_2's data has multiple integer values associated, and parties wish to learn the sum of all such values. This is handled by simply packing all such values into different slots of the homomorphic encryption, and creating one sum for each type of value.

Multiple values also enable computing the variance of the intersection-sum, by additionally sending the square of the original associated value. The parties will learn the intersection size, intersection sum, and intersection-sum-of-squares, from which they can compute the variance.

4. Revisiting the Problem: New Protocols

In this section, we move to the second main contribution of this work, namely revisiting the problem of securely computing Private Intersection-Sum-with-Cardinality, in light of the vast recent progress in PSI protocols. We give two new protocols based on new approaches. In addition, we describe a simple recipe for turning a particularly common flavor of PSI protocols into an intersection-sum-with-cardinality protocol.

4.1. Private Set Intersection: an Overview

Private set intersection has been studied extensively in a long sequence of works [14], [15], [17], [20], [22], [26], [32], [33], [37], [38], [49], [52], [55], [56]. Several works limit the parties to learning only the *cardinality* of the intersection [1], [13], [26], [33], [36], [44], [57].

The most prevalent technique among recent specialized protocols for set intersection, which also underlies the most efficient PSI constructions currently [37], [48], is based on the concept of oblivious pseudorandom functions (OPRF) [25]. 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

3. We note that this method of hashing to the curve is susceptible to timing attacks, however these attacks don't apply in our scenario since items are hashed to the curve before protocol execution. Methods of hashing to a curve that are free from timing attacks can be found at https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve

	Comp. time (hrs).	Total Comm. (MB)	Comp. cost (cents)	Comm. cost(cents)	Total Cost (cents)
Single execution (100,000 entries)	0.2068	8.105	0.0206	0.06332	0.08400
Daily total (1,000 executions)	20.68	8105	20.68	63.32	84.00

TABLE 2: Concrete deployments costs, for both a single protocol execution over 100,000 entries for each party, and for a day which has 1,000 such executions. Monetary costs are in cents (USD) use the values for GCP from Table 1 to value communication and computation.

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 [52], 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 [18] 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. [20], and was later strengthened by Rindal et al. [55] in the context of malicious set intersection leverages the notion of Bloom filters (BF) [4]. 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.

Similar to our deployed approach, there are also works based on the Decisional Diffie–Hellman (DDH) computational assumption, which comes from some older works [14], [33], [34], [38], [56]. The general ideas are as follows: 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 [34].

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 [32] 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. [49] hashes items to bins and uses garbled circuits to perform intersection bin-wise. More recently, several partly concurrent works [11], [31], [51] 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. [35] explore offline precomputation to maximize online efficiency for PSI. Several works also rely on additional parties or multiple non-colluding servers in order to compute aggregate statistics, for example [12], [27], [40].

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

As 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 (this is the case for both our deployed protocol and our new constructions). 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 [33], 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 items were 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, each 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 Appendix B. 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 take 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 new private intersection-sum protocol described in Section 4.3.1, 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 4.3.2 is inspired by the techniques using encrypted Bloom Filters.

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.

4.3. New Protocols for PSI-Sum with Cardinality

In this section we present our new cryptographic protocols for secure private set intersection-sum-with-cardinality which leverage recent ROT and BF techniques for set intersection.

4.3.1. 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 [20], [49], [52], [55]. We believe this is the first construction of the Private Intersection-Sum-with-Cardinality 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 A:

- Random OT with the following functionality: P_1 has no input and obtains a PRF key k that can be evaluated on some known domain. P_2 has input x and receives $f_k(x)$, the evaluation of the PRF on the point x. Additionally, any evaluations of f_k on a polynomial subset of inputs (including or excluding on x) appears pseudorandom to P_2.
- An additively homomorphic encryption scheme $(\mathsf{AGen}, \mathsf{AEnc}, \mathsf{ADec})$.

We present our protocol Π_{ROT} in Fig. 3. At a high level, the protocol Π_{ROT} is similar to the Batched-OPRF based PSI of [37]. 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 $f_{k_i}(x_i)$ to $f_{k_i}(x_i) + r_i$, 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 [31] and [11] have some high-level similarities with our protocol, in that they adapt [37]-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, [31] 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. Security is proved using a standard hybrid argument, where the encryptions are replaced with encryptions of zero, and the outputs of the Random OT are replaced with unrelated uniformly random values, in such a way that the intersection size and intersection-sum remain the same. We state the theorems here. These theorems prove that the view of an honest party interacting in a real execution can be simulated by a simulator who only knows the intersection sum and intersection size.

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

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

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 4 (HBC-Security against Party 2 in Π_{ROT})**.** *There exists a PPT simulator* SIM_2 *such that for all*

Randomized-OT based Private Intersection-Sum protocol

- **Inputs**:

$$\mathsf{P}_1 : \text{Set } V = \{v_i\}_{i=1}^{m_1} \qquad\qquad \mathsf{P}_2 : \text{Set of pairs } W = \{(w_i, t_i)\}_{i=1}^{m_2}$$

- **Setup**: P_1 and P_2 each generate an additively homomorphic encryption key-pair (pk_1, sk_1) and (pk_2, sk_2) and exchange public keys pk_1 and pk_2.
- **Protocol Steps**:

  1) P_1 and P_2 choose Cuckoo hash table size n, stash size s, and l hash functions: $h_1, ..., h_l$ mapping inputs to bins $[1, ..., n]$.
  2) P_1 hashes its items using Cuckoo hashing with $h_1, ..., h_l$ 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 hashing fails.
  3) For each $1 \le i \le n+s$ the parties run a random OT where P_2 obtains a PRF key k_i and P_1 obtains the evaluation $f_{k_i}(v_i)$.
  4) P_1 encrypts and sends to P_2: $\{\mathsf{ct}_i = \mathsf{AEnc}(pk_1, f_{k_i}(v_i))\}_{i=1}^{n+s}$
  5) P_2 chooses random values $\{r_i\}_{i=1}^{n+s}$, and, for each $i \in [1, n+s]$, computes: $\mathsf{ct}'_i = \mathsf{ARefresh}(\mathsf{ct}_i + \mathsf{AEnc}(pk_1, r_i))$. 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:

  $$\mathcal{I}_j = \{h_i(w_j)\}_{i=1}^{l} \cup \{n+1, ..., n+s\} \qquad W_j = \{f_{k_i}(w_j) + r_i\}_{i \in \mathcal{I}_j} \qquad \mathsf{ct}^t_j = \mathsf{AEnc}(pk_2, t_j)$$

     If any W_j has size $< l+s$, P_2 pads W_j with dummy random elements until it has size $l+s$. P_2 sends P_1 the set $\{(W_j, \mathsf{ct}^t_j)\}_{j=1}^{m_2}$, 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

  $$\mathsf{CT} = \sum_{j \,:\, |W_j \cap V^*| > 0} \mathsf{ct}^t_j$$

     P_1 rerandomizes CT into $\mathsf{CT}' := \mathsf{ARefresh}(\mathsf{CT})$ and sends it to P_2.

- **Output**: P_2 decrypts CT' to recover d, and outputs d, equal to $\sum_{i:w_i \in V} t_i$, the intersection-sum.

Figure 3: Π_{ROT} : Randomized-OT based Private Intersection-Sum protocol.

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.

The formal security proof appears in Appendix E.

4.3.2. 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 [17], [20], [22].

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

We present the protocol in Figure 4. 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. The proof proceeds using a standard hybrid argument, which replaces the encrypted associated values with encryptions of 0, while keeping the intersection-sum the same.

The formal security proof appears in Appendix F. It shows that the view of each party can be simulated using only the intersection size and sum.

Encrypted-Bloom-Filter-based Intersection-Sum Protocol

- **Inputs:**

  P_1 : Set $V = \{v_i\}_{i=1}^{m_1}$ $\qquad$ P_2 : Set of pairs $W = \{(w_i, t_i)\}_{i=1}^{m_2}$

- **Setup**: P_1 and P_2 each generate an additively homomorphic encryption key-pair (pk_1, sk_1) and (pk_2, sk_2) and exchange public keys pk_1 and pk_2.
- **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 (where r_j is random):

     $$\mathsf{ct}_i^2 = r_j.(\sum_j \mathsf{ct}^1_{\mathsf{IND}_j} - \mathsf{AEnc}(pk_1, k)) = \mathsf{AEnc}(pk_1, r_j(\sum_j b_{\mathsf{IND}_j} - k))$$

  5) P_2 sends to P_1 the following values in a permuted order: $\{\mathsf{ARefresh}(\mathsf{ct}_i^2), \mathsf{ct}_i^t = \mathsf{AEnc}(pk_2, t_i)\}_i$.
  6) P_1 decrypts all values $\mathsf{ARefresh}(\mathsf{ct}_i^2)$ and computes: $\mathsf{CT} = \sum_{j:\mathsf{ADec}(sk_1, \mathsf{ct}_j^2)=0} \mathsf{ct}_j^t$ P_1 rerandomizes CT into $\mathsf{CT}' := \mathsf{ARefresh}(\mathsf{CT})$, and sends it to P_2.

- **Output**: P_2 decrypts CT' to recover d, and outputs $d = \sum_{i:w_i \in V} t_i$.

Figure 4: Π_{BF}: Encrypted-Bloom-Filter-based Private Intersection-Sum protocol.

5. Measurements

In this section, we present the measurements for our implementations of our two new protocols for private intersection-sum-and-cardinality presented in Sections 4.3.1 and 4.3.2, along with additional measurements for our deployed protocol described in Section 3.1, both as deployed, and a variant using a different homomorphic encryption scheme. We also compare the concrete costs with numbers that have been reported previously for schemes based on Garbled Circuits, specifically the works [31], [49], [50], [52], which are the best known works for computing function on a private intersection. Our comparisons include monetary costs based on a "batch-computing" scenario, using GCP cloud pricing for resource costs from Table 1.

In Tables 3 and 4, we present the asymptotic 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.

5.1. Parameters and Encryption Schemes

All computational cost measurements for our protocols 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 32GB of RAM, which is similar to the machines in our deployment environment. Computational cost excludes the time spent transferring files between parties over the network.

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 also 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.

Random-OT based protocol. We use Paillier encryption to encrypt the $f_{k_i}(v_i)$ 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. [18] 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. [37], 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 [37] protocol.

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 256-bit group elements. As noted in Appendix B 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}.

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

TABLE 3: Computational operations in each protocol. Numbers are totals across both parties, each having M inputs. “Exponentiations” is the number of group exponentiations (or “public-key” operations). Homomorphic Ops. are the number of homomorphic encryptions, decryptions and additions. We assume homomorphic operations have the same cost. 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	–	$3.6M+1$	$\mathrm{ROT}(1.3M)+96M\mathrm{Bytes}$
Bloom-filter-based Protocol	-	$M(61+1.44\log_2(M))+1$	-

TABLE 4: Communication in each Private Intersection-Sum protocol. Numbers presented are totals across both parties, each having M inputs. Group Elements is the number of EC curve elements transferred. HE Ciphertexts is the number of homomorphic encryptions transferred, not including optimizations like slotting. ROT(x) refers to the communication cost of x Random OTs.

For our DDH-protocol, we use we use “prime256v1” as the curve, and SHA-256 as the Random Oracle, as described in Section 3.2.

Finally, to encrypt the associated values, we present measurements using “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^{-27} statistical-hiding of the error (even with our largest input size of 2^{20} entries). Each ciphertext is 20 KB and can hold up to 2048 values of 32 bits each. See Appendix B for more details. We found experimentally that slotted RLWE encryption has roughly the same communication cost as slotted Paillier-encryption with Damgard-Jurik optimizations, while needing significantly less computation. We present additional comparisons of the different homomorphic encryption schemes in Appendix H.

5.2. Discussion of Measurements

We present our measurements in Table 5, and monetary costs in Table 6. We show the costs for the deployed protocol based on DDH, as well as the 2 new protocols based on ROT and BF, using RLWE as the homomorphic encryption scheme for the associated values, and also compare our deployed protocol, which uses the DDH-protocol with Slotted Paillier as the encryption scheme (see Section 3.2).

We see that the monetary costs closely track communication costs. The DDH protocol with Paillier has most succinct communication, and also the most efficient monetary cost, whereas the ROT-protocol has $15\times$ more monetary cost and communication, and the BF-based protocol has over $100\times$ more of each. The DDH protocol, specifically with RLWE as the homomorphic encryption, turns out to be our most computationally efficient protocol, about 25 $\times$ and $40\times$ faster than the ROT and the BF protocols respectively. Finally, the DDH protocol with RLWE has 11% smaller monetary costs as the DDH protocol with Paillier, with about 40% less computation.

5.3. Comparison with Garbled Circuits Works

In this section, we compare concrete costs (computation, communication and monetary) with those presented in existing works based on Garbled Circuits, specifically [31], [49], [50], [52]. For these costs, we rely heavily on the excellent exposition of [50], drawing heavily from Tables 3 and 5 of [50]. [4]

For all Garbled Circuit protocols, we assume that the identifiers in each party’s inputs are hashed to be $40+\log(n)$ bits long where n is the input size: this prevents hash-collisions except with probability 2^{-40}.

The comparisons of communication, computation and monetary costs are presented in Table 7.

We note that the comparisons are imperfect for several reasons. One important point is that the running times presented in [50] include time for communication transfer over a LAN, whereas for our protocols, we ignore time for communication and includes only running time for computation. This means we somewhat over-estimate the computation costs of the garbled-circuit-style solutions. Furthermore the measurements use slightly different execution environments. Another point is that the communication costs for the Garbled Circuit solutions in [50] are only for the PSI functionality, and not for Private Intersection-Sum. We ignore the additional cost associated with computing the intersection-sum, and therefore somewhat underestimate the communication costs of the Garbled-Circuit-style solutions.

Even with these caveats, we believe the comparison is informative, and bears out expectations, namely that Garbled Circuit solutions are faster computationally than the solutions we present, but also incur significantly more communication cost. Because of the dramatic difference in communication costs, these protocols also turn out to be much more expensive in terms of monetary cost in our cost model.

Table 7 shows that the ROT-based protocol and both variants of the DDH-based protocol, including the deployed protocol, are cheaper in monetary terms than any of the protocols based on garbled circuits. In particular, the deployed protocol is $20-30\times$ cheaper than the cheapest protocol based on garbled-circuits [50] for the data sizes considered. Our new protocol based on Bloom FIlters is also cheaper than all of the Garbled Circuit-based solutions except [50].

Within the protocols newly presented in this paper, the DH-based protocols are still the cheapest in terms of

4. One important omission from the comparison is the work of [11], which doesn’t have an existing implementation. We refer readers to Table 8 for an asymptotic comparison.

	DDH + Paillier (Deployed)		DDH + RLWE		Random-OT + RLWE		Bloom Filter + RLWE	
Input Size	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)	Time (s)	Comm. (MB)
1000	0.81	0.08	0.49	0.14	9.89	1.33	18.24	8.25
2000	1.55	0.16	0.96	0.21	19.61	2.52	36.60	16.76
3000	2.29	0.24	1.42	0.30	29.66	3.74	56.22	25.41
4000	3.04	0.33	1.88	0.37	39.33	4.94	74.28	34.11
5000	3.79	0.41	2.34	0.47	48.72	6.16	93.29	42.90
10000	7.47	0.81	4.72	0.87	97.67	12.22	191.34	87.33
20000	14.81	1.62	9.61	1.71	196.86	24.55	383.75	177.82
30000	22.15	2.43	14.16	2.54	294.95	36.73	582.56	269.51
40000	29.63	3.24	18.73	3.37	399.50	49.02	782.62	361.98
50000	37.08	4.05	23.26	4.20	492.17	61.34	987.30	455.04
100000	74.46	8.11	47.40	8.34	989.88	123.21	1988.47	925.98

TABLE 5: Comparison of Computation time and Network costs for Private Intersection-Sum protocol. We include the deployed protocol (which uses Paillier encryption), together with the variants using slotted Ring-LWE-encryption to encrypt the associated values.

Input	DDH + Paillier (Deployed)	DDH + RLWE	Random-OT + RLWE	Bloom Filter + RLWE
1000	0.0008	0.0011	0.0131	0.0695
2000	0.0017	0.0018	0.0251	0.1410
3000	0.0025	0.0026	0.0374	0.2140
4000	0.0033	0.0033	0.0494	0.2871
5000	0.0042	0.0041	0.0616	0.3610
10000	0.0084	0.0078	0.1245	0.7354
20000	0.0167	0.0153	0.2456	1.4958
30000	0.0251	0.0228	0.3689	2.2673
40000	0.0335	0.0303	0.4939	3.0453
50000	0.0419	0.0377	0.6159	3.8293
100000	0.0840	0.0753	1.237	7.7865

TABLE 6: Comparison of Monetary costs in cents (USD) for our Private-Intersection-Sum protocol variants. We include the deployed protocol (which uses Paillier encryption), together with the variants using slotted Ring-LWE-encryption to encrypt the associated values. Resource valuations use the costs for GCP from Table 1.

	Input size 2^{12}			Input size 2^{16}			Input size 2^{20}		
Garbled-Circuit Protocols	Comm.	Time	Cost	Comm.	Time	Cost	Comm.	Time	Cost
Phasing + OPPRF + GC [50]	9	1199	0.070	149	8486	1.16	2540	120731	19.8
2D-Cuckoo-Hashing [49]	115	5031	0.89	1751	25960	13.68	25532	336134	199.5
Hashing + SCS [31]	-	-	-	3998	-	-	72140	-	-
Circuit-Phasing [52]	320	7825	2.50	552	67292	43.3	99708	1126848	763.6
Our Protocols	Comm.	Time	Cost	Comm.	Time	Cost	Comm.	Time	Cost
DH-Protocol + RLWE	0.38	1949	0.003	5.5	30977	0.05	87	499468	0.81
ROT-Protocol + RLWE	5	40052	0.05	84	64413	0.83	1380	10601582	13.7
BF-Protocol + ElGamal + RLWE	37	76133	0.307	629	1318153	5.28	-	-	-
DH-Protocol + Paillier (Deployed)	0.33	3165	0.003	5.1	48930	0.06	84	776461	0.88

TABLE 7: Comparison of Communication, Computation and Monetary Costs between Garbled Circuit Protocols and our solutions. Garbled Circuit Protocol costs are drawn from Tables 3 and 5 of [50], and are only costs for PSI, and exclude the cost of Intersection-sum. Garbled Circuit protocols also include time due to network communication over a LAN, while our solutions exclude network communication time. All communication costs are in MB, all times are in milliseconds, and all "Costs" are the monetary cost in cents (USD), using the resource valuations for GCP in Table 1. Missing measurements are either due to unavailable measurements, or because the computation ran out of memory.

monetary cost. The DH-protocol using Paillier (that is, our deployed protocol) is more expensive than the variant based on RLWE in monetary cost, but the difference is small (around 10%). Meanwhile, the RLWE-based protocol is 40% cheaper in computation. This means that the RLWE-based protocol may be especially useful in settings where computation cost is higher or communication cost is cheaper than the valuations we consider.

6. Conclusion

We described deploying 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. We described factors that we believe made this problem a good fit for a secure-computing solution, and described various practical constraints on our design.

The protocol we deployed was chosen because of its simplicity, but also for its communication efficiency, which has a huge impact on monetary cost. We also revisited the problem of private intersection-sum with cardinality in light of recent advancements, based on which we constructed two new protocols. We implemented all our constructions in order to evaluate and compare their concrete costs, including monetary costs. From our measurements, all but one of our protocols compares favorably in terms of monetary cost against approaches for securely computing generic functions over an intersection. Our deployed protocol was found to be 20× cheaper in monetary cost than generic approaches. Our other solutions offer better computation in trade off for communication.

References

[1] Agrawal, R., Evfimievski, A.V., Srikant, R.: Information sharing across private databases. In: SIGMOD Conference. pp. 86–97. ACM (2003)

[2] Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: ACM Conference on Computer and Communications Security. pp. 805–817. ACM (2016)

[3] Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM Conference on Computer and Communications Security. pp. 62–73. ACM (1993)

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

[5] Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis - (short paper). In: Financial Cryptography. Lecture Notes in Computer Science, vol. 7397, pp. 57–64. Springer (2012)

[6] Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T.P., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 5628, pp. 325–343. Springer (2009)

[7] Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1175–1191. ACM (2017)

[8] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. TOCT **6**(3), 13:1–13:36 (2014)

[9] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM J. Comput. **43**(2), 831–871 (2014)

[10] Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: ACM 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. In: International Conference on Security and Cryptography for Networks. pp. 464–482. Springer (2018)

[12] Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). pp. 259–282 (2017)

[13] Cristofaro, E.D., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: CANS. vol. 7712, pp. 218–231. Springer (2012)

[14] Cristofaro, E.D., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 213–231. Springer (2010)

[15] 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)

[16] Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 1992, pp. 119–136. Springer (2001)

[17] 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)

[18] Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: Pir-psi: scaling private contact discovery. Proceedings on Privacy Enhancing Technologies **2018**(4), 159–178 (2018)

[19] Diffie, W., Hellman, M.: New directions in cryptography. IEEE transactions on Information Theory **22**(6), 644–654 (1976)

[20] 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)

[21] Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of Cryptography (2006)

[22] 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)

[23] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)

[24] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive **2012**, 144 (2012)

[25] 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)

[26] 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)

[27] Froelicher, D., Egger, P., Sousa, J.S., Raisaro, J.L., Huang, Z., Mouchet, C., Ford, B., Hubaux, J.P.: Unlynx: a decentralized system for privacy-conscious data sharing. Proceedings on Privacy Enhancing Technologies **2017**(4), 232–250 (2017)

[28] Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority (2017)

[29] 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)

[30] Hellman, M.E., Pohlig, S.C.: Exponentiation cryptographic apparatus and method (Jan 3 1984), uS Patent 4,424,414

[31] Hemenway Falk, B., Noble, D., Ostrovsky, R.: Private set intersection with linear communication from general assumptions. In: Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society. pp. 14–25 (2019)

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

[33] 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)

[34] Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: SCN. Lecture Notes in Computer Science, vol. 6280, pp. 418–435. Springer (2010)

[35] 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)

[36] 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

[37] 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)

[38] 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)

[39] 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)

[40] Le, P.H., Ranellucci, S., Gordon, S.D.: Two-party private set intersection with an untrusted third party. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2403–2420. ACM (2019)

[41] 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)

[42] Mitzenmacher, M., Upfal, E.: Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge university press (2005)

[43] Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. J. ACM **51**(2), 231–262 (2004)

[44] 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)

[45] Pagh, R., Rodler, F.F.: Cuckoo hashing. J. Algorithms **51**(2), 122–144 (2004)

[46] 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)

[47] Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: SpOT-light: Lightweight private set intersection from sparse OT extension. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 11694, pp. 401–431. Springer (2019)

[48] Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Spot-light: Lightweight private set intersection from sparse ot extension. In: Annual International Cryptology Conference. pp. 401–431. Springer (2019)

[49] 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)

[50] Pinkas, B., Schneider, T., Tkachenko, O., Yanai, A.: Efficient circuit-based PSI with linear communication. In: EUROCRYPT (3). Lecture Notes in Computer Science, vol. 11478, pp. 122–153. Springer (2019)

[51] 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)

[52] Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on ot extension. In: Usenix Security. vol. 14, pp. 797–812 (2014)

[53] Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Trans. Priv. Secur. **21**(2), 7:1–7:35 (2018)

[54] Rajan, A., Qin, L., Archer, D.W., Boneh, D., Lepoint, T., Varia, M.: Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In: COMPASS. pp. 49:1–49:4. ACM (2018)

[55] Rindal, P., Rosulek, M.: Improved private set intersection against malicious adversaries. Tech. rep. (2016)

[56] Segal, A., Ford, B., Feigenbaum, J.: Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In: FOCI (2014)

[57] Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security **13**(4), 593–622 (2005)

[58] 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)

Appendix A. Preliminaries, Cryptographic Primitives and Hardness Assumptions

A.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.

A.2. Decisional Diffie–Hellman

Definition 1 (Decisional Diffie–Hellman [19]). *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.

A.3. Random Oracle Model

Definition 2 (Random Oracle [3]). *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.

A.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*[5]*, which we denote for ease of exposition as:*

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

Note 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. In addition, we will make use of the property that one can randomize ciphertexts using a randomized procedure denoted as ARefresh. In particular, we will use the fact that the distribution of

$$\mathsf{ARefresh}(\mathsf{ASum}(\{\mathsf{AEnc}(pk, m_i)\}))$$

and

$$\mathsf{ARefresh}(\mathsf{AEnc}(pk, \sum_i m_i))$$

are statistically close, even against an adversary that holds the secret key sk. In popular schemes such as Paillier [46] and Exponential ElGamal [23], this ARefresh can be achieved by adding an additional fresh encryption of 0. In the RLWE-based homomorphic encryption scheme [8], this can be achieved by adding a sufficiently large random value in order to drown the "noise" of the ciphertext.

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.

A.5. Oblivious Pseudorandom Function (OPRF)

Definition 4 (Oblivious Pseudorandom Function [25], [43])**.** *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.*

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 Party 1 to receive the output of a PRF f_k on an input x, using a key k held by Party 2, while hiding the input x from Party 2. Furthermore, the evaluation of the PRF f_k on all other inputs remains pseudorandom in the view of Party 1.

A.6. Random Oblivious Transfer (ROT)

Definition 5 (Random Oblivious Transfer [37])**.** *Let f_k be a Pseudorandom function family. 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 keys k_i; Party 1 outputs $f_{k_i}(x_i)$.*

That is,ROT can be viewed as multiple OPRF protocols executed simultaneously, where the keys for the underlying PRF are dynamically generated during the ROT execution.. Like in an OPRF evaluation, the output of any f_{k_i} on any point $x \neq x_i$ remains pseudorandom in the view of Party 1. (In the implementation of [37], technically the PRF family is a "relaxed" PRF family with related keys. We refer the reader to [37] for details, and omit them here since the distinction does not have a significant impact on our protocol or proofs.)

A.7. Bloom Filter

Definition 6 (Bloom Filter [4])**.** *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 $h_1, ..., h_k \colon \{0,1\}^* \rightarrow [N]$. 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. [42] is a good reference on setting the Bloom Filter parameters.

A.8. Cuckoo Hash

Definition 7 (Cuckoo Hash Table [45])**.** *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.

[18] 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.

Appendix B. Instantiating the Homomorphic Encryption scheme

Each of the three Private Intersection-Sum protocols that we presented, requires an additive-homomorphic encryption scheme in order to encrypt the associated values

5. 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.

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 [46], Exponential ElGamal encryption [23], and schemes based on Ring-LWE [8], [9], [24], [29]. We discuss the various characteristics of each scheme, together with optimizations that could be applied to each of them. These differences are summarized in Figure 5.

B.1. Paillier Encryption

Paillier encryption [46] is a well-known additively homomorphic encryption scheme with security based on the Decisional Composite Residiuosity Assumption.

It requires relatively expensive modular exponentiation (*"public-key"*) operations in order to encrypt and decrypt. Paillier ciphertexts also have 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 $200\times$ ciphertext expansion compared to the plaintext.

However, the Paillier scheme's large plaintext space can be divided into "slots" by packing multiple values together by shifting-and-adding, to create a single large place that better utilizes the large capacity. The lower slots can also be shifted into higher positions by homomorphically multiplying powers of 2.

Slotting helps make Paillier encryption much more efficient for encrypting associated values: 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.

The slots need to be large enough to accomodate the associated values, with some extra bits to allow summing without overflow, and some further bits to allow random masking. Additionally, only half the plaintext space can be used for slots in order to rotate the lowest slot into the highest slot's position without overflowing the plaintext space. Slotting can also be combined with Damgard-Jurik optimizations [16], 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 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.

B.2. Exponential ElGamal

Exponential ElGamal encryption [23] also has public-key operations for encryption, but has relatively smaller ciphertexts than Paillier encryption. ElGamal over elliptic curves has ciphertexts of length 512 bits, which for a 20 bit plaintext value would have an expansion of $25\times$. A 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 (e.g. 20 bits), and the number of them to add together is also small (e.g. 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. Decryption can also be made faster by using lookup tables in memory

Exponential ElGamal is an improvement over Paillier in the Bloom Filter-based protocol (Section 4.3.2), 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.

B.3. Ring-LWE-Based Cryptosystems

Another option is to lattice-based encryption. The most efficient schemes of this type are based on the hardness of Ring-Learning-With-Errors [8], [9], [24], [29].[6] 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).

RLWE-based schemes also have ciphertexts with large plaintext spaces, but these naturally decompose into multiple 'slots', where each slot is individually additively homomorphic. Furthermore, slots can be efficiently 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

6. These schemes are actually "fully"-homomorphic, but here we consider only their additive homomorphism

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)

Figure 5: 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.

	# of sym. key operations	Communication (bits)
Yao SCS [32]	$12\lambda M + 3\lambda M$	$2\lambda Ms(1 + 3\log M)$
GMW SCS [32]	$12\lambda M \log M$	$6\lambda M(s + 2)\log M$
Yao PWC [53]	$4\lambda M + 6393\lambda$	$\lambda(M3s + 3198s + 15)$
GMW PWC [53]	$6\lambda M + 9594\lambda$	$\lambda(M4 + 6396 + 2sM + 6396s)$
Graph Navigation [11]	$4\lambda M + 3\lambda$	$2\lambda Ms + Ms$

TABLE 8: (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.

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.

One subtlety with RLWE-based encryption schemes is in the ARefresh procedure used to randomize the ciphertexts. Indeed, the error in the resulting ciphertexts must be masked to hide the sequence of operations used to create them. To do this, the ARefresh procedure *statistically drowns* the error with a large amount of fresh noise, so as to statistically hide which input ciphertexts were used. This induces some additional communication overhead.[7]

Appendix C. Analytical 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. [8]

To clarify the comparison to our works, compare to Table 3. Table 3 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.

Appendix D. Security Analysis for DDH-Based Protocol

In this section, we will prove security of the DDH-based protocol Π_{DDH}, presented in Figure 2, Section 3.1. 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).

7. 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 $30 + 13 + 40 + 1 = 84$ bits (where the last bit is required to ensure correctness), to hold each 20 bit associated value, an expansion of $4.2\times$.

8. [50] describes hashing or truncating the inputs to the Garbled Circuits to be $40 + \log(M) - 1$ bits long, where M is the number of inputs. However, each of these inputs still need to be encrypted bit-by-bit, which leads to similar communication complexity.

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 3.1, 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 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* P_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. Notice that the main difference between SIM_1 and

Algorithm 1 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 $\mathsf{P}_1$'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 $\mathsf{P}_2$'s message in Step 2 of Round 2.
  4: Create a dummy set $W^* = \{h_j\}_{j=1}^{m_2}$ for $\mathsf{P}_2$ 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 $\mathsf{P}_2$'s message in Step 4 of Round 2, where each $\mathsf{AEnc}(0)$ is freshly generated.
  6: Honestly generate $\mathsf{P}_1$'s message in Round 3 using $\mathsf{P}_2$'s dummy messages from the previous step.
  7: Output $\mathsf{P}_1$'s view in the simulated execution above.
```

a real protocol execution is in Round 2: instead of sending $\{\mathsf{H}(v_i)^{k_1k_2}\}$ 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 $\mathsf{H}(v_{i^*})^{k_1k_2}$ 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 • $\mathsf{H}(v_{k^*})^{k_1k_2}$ 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 2 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 2 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 $\mathsf{H}(v_i)^{k_1k_2}$, 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 2 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 $g_{i^*}^{k_1} = \mathsf{H}(v_i)^{k_1k_2}$. This change is exactly the difference

Algorithm 2 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 $\mathsf{Hyb}_{1,i}$

```
1: for $i \in \{1, ..., m_1\}$ do
2:     if $v_i \neq v_{i^*}$ then
3:         Randomly sample $r_i \leftarrow \{1, , ..., |\mathcal{G}|\}$
4:         Program $\mathsf{H}(v_i) = g^{r_i}$
5:     else if $v_i = v_{i^*}$ then
6:         Program $\mathsf{H}(v_i) = g^a$
7:     end if
8: end for
9: for $j \in \{1, ..., m_2\}$ do
10:    if $w_j \notin \{v_i\}_{i=1}^{m_1}$ then
11:        Randomly sample $s_j \leftarrow \{1, ..., |\mathcal{G}|\}$
12:        Program $\mathsf{H}(w_j) = g^{s_j}$
13:    end if
14: end for
15: Generate key $k_1 \in \mathcal{G}$, and key-pair $(pk, sk)$ for the additively homomorphic encryption scheme.
16: Send $\{\mathsf{H}(v_i)^{k_1}\}_{i=1}^{m_1}$ in shuffled order as Party 1's message in Round 1.
17: for $i \in \{1, ..., m_1\}$ do
18:    if $v_i = v_{i^*}$ then
19:        $g_i \leftarrow g^c$
20:    else if $v_i \notin \{w_j\}_{j=1}^{m_2}, v_i < v_{i^*}$ then
21:        $g_i \leftarrow$ random element of $\mathcal{G}$
22:    else
23:        $g_i \leftarrow (g^b)^{s_i}$
24:    end if
25: end for
26: Send $\{g_i^{k_1}\}_{i\in[m]}$ in shuffled order as $\mathsf{P}_2$'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.
27: Honestly generate $\mathsf{P}_1$'s message in Round 3 using $\mathsf{P}_2$'s dummy messages from the previous step.
28: Output $\mathsf{P}_1$'s view in the simulated execution above.
```

between $\mathsf{Hyb}_{1,i-1}$ and $\mathsf{Hyb}_{1,i}$. Thus, the output of Algorithm 2 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 3.1, 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 2 (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* $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,

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

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.

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 1, and we leave them as an exercise. □

Appendix E. 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 4.3.1 and Figure 3. We prove security in the honest-but-curious model.

Let $\mathsf{REAL}_{\Pi_{\mathsf{ROT}}}^{i,\lambda}(\{v_i\}_{i=1}^{m_1}, \{(w_j, t_j)\}_{j=1}^{m_2})$ 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 4.3.1:

Theorem 3 (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* $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,

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

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 3.

Algorithm 3 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 $\mathsf{P}_1$ and $\mathsf{P}_2$.
2: Honestly simulate steps 1-4 between $\mathsf{P}_1$ and $\mathsf{P}_2$, receiving the set $\{\mathsf{ct}_i = \mathsf{AEnc}(pk_1, f_{k_i}(v_i))\}_{i=1}^{n+s}$ at the end of Step 4.
3: In Step 5, choose a set $A^*$ consisting of $n+s$ random elements $a_i$. Compute $\mathsf{ct}'_i$ to fresh encryptions to each $a_i$ under $\mathsf{P}_1$'s public key $pk_1$, and send these values to $\mathsf{P}_1$.
4: In Step 6, choose the set $\{(W_j, \mathsf{ct}_j^t)\}_{i=1}^{m_2}$ such that each $W_j$ contains exactly $k+s$ randomly chosen elements, and each $\mathsf{ct}_j^t$ is a fresh encryption of 0 under $\mathsf{P}_2$'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 $\{(W_j, \mathsf{ct}_j^t)\}_{i=1}^{m_2}$ to $\mathsf{P}_1$.
5: Simulate Step 7 for $\mathsf{P}_1$ honestly.
6: Output the view of $\mathsf{P}_1$ in this interaction.
```

We argue that

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

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 $f_{k_i}(w_j)$ values for w_j 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

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

□

We now argue that P_2 learns nothing from the protocol except the intersection-sum. We restate the theorem from Section 4.3.1:

Theorem 4 (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 $\{v_i\}_{i=1}^{m_1}$, $\{(w_j, t_j)\}_{j=1}^{m_2}$,*

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

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.

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. □

Appendix F. 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 4.3.2 and Figure 4. 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.

Appendix G. "Reverse" variants

We note that, in each of the protocols described in Section 4.3, 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 G, 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.

G.1. The "Reverse" DDH-based Protocol

The DDH-based Private Intersection-Sum protocol Π_{DDH} we presented in Section 3.1 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.

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 $\sum_{j \in J} t_j$.

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

G.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 1. We omit details of simulators and the hybrid security argument.

Appendix H. Additional Measurements

H.1. Using an idealized homomorphic encryption scheme

In Table 9 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.

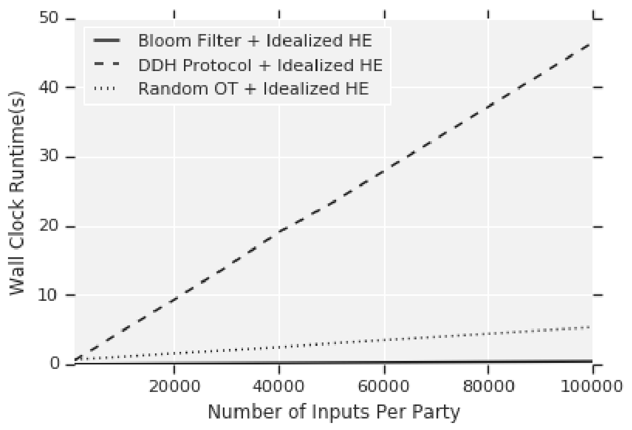
In this section, we present some additional measurements for the Random-OT based protocol (Section 4.3.1), Figure 3) using a different combination of additive homomorphic encryptions.

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

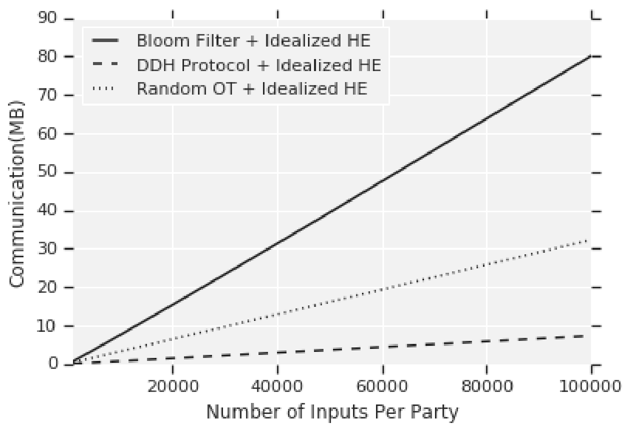
In particular, we measure the computation and communication when we use Ring-LWE for encrypting both $f_{k_i}(v_i)$ values, and encrypting the associated values. We use two different sets of RLWE parameters for the two different use-cases.

For encrypting the associated values, we use the same parameters as discussed in Section 5, namely slotted RLWE encryption with an 80 bit modulus and 2048 coefficients, with 32 bit plaintext space.

For encrypting $f_{k_i}(v_i)$ values, we use RLWE encryption with a 14-bit modulus, 1024 coefficients, and a plaintext modulus of 2, corresponding to 1 bit per coefficient.



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



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

Figure 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.

	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 9: 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.

We encrypt each $f_{k_i}(v_i)$ value in a different ciphertext. To be encrypted, each 256 bit $f_{k_i}(v_i)$ 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 10. For easy comparison, we also present the costs when using unslotted-Paillier to encrypt both $f_{k_i}(v_i)$ and the associated values, and also the costs from Table 5, which shows the cost when using unslotted-Paillier to encrypt the $f_{k_i}(v_i)$ 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 $f_{k_i}(v_i)$ and second part representing the scheme used for the associated values.

We note that using RLWE for both the $f_{k_i}(v_i)$ 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 5).

	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 10: Comparison of variants of the Random-OT based Private Intersection-Sum protocol, using different homomorphic encryption schemes to encrypt the $f_{k_i}(v_i)$ values and the associated values.