Multiparty Private Set Intersection Cardinality and Its Applications

Ni Trieu* Avishay Yanai† Jiahui Gao*

June 8, 2022

Abstract

We describe a new paradigm for multi-party private set intersection cardinality (PSI-CA) that allows n parties to compute the intersection size of their datasets without revealing any additional information. We explore a variety of instantiations of this paradigm. Our protocols avoid computationally expensive public-key operations and are secure in the presence of a malicious adversary.

We demonstrate the practicality of our PSI-CA protocol with an implementation. For $n = 16$ parties with data-sets of 2^{20} items each, our server-aided variant takes 71 seconds. Interestingly, in the server-less setting, the same task takes only 7 seconds. To the best of our knowledge, this is the first 'special purpose' implementation of a multi-party PSI-CA (i.e., an implementation that does not rely on a generic underlying MPC protocol).

Our PSI-CA protocols can be used to securely compute the dot-product function. The dot-product function takes n binary vectors $v_1, \ldots, v_n$, each of m elements, and outputs the sum of m entries, where the i-th entry is equal the product of the i-th entries in all n input vectors. Importantly, the complexity of our protocol for secure dot-product (where party P_i has a secret vector v_i) is linear only in the Hamming weight of the vectors, which is potentially sub-linear in the input size.

We demonstrate that two interesting applications, namely, 'COVID-19 heatmap' and 'associated rule learning (ARL)', can be computed securely using a dot-product as a building block. We analyse the performance of securely computing Covid-19 heatmap and ARL using our protocol and compare that to the state-of-the-art.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to jointly invoke a distributed computation while ensuring correctness, privacy, and more. Garbled circuit [Yao86, GMW87, BMR90] is a popular generic technique for secure computation, which has been enjoyed notable optimizations in recent years (e.g. [RR21]). However, for concrete applications, special-purpose protocols significantly improve performance compared to circuit-based approaches.

In this work, we study Private Set Intersection Cardinality (PSI-CA), a special case of MPC, that allows multiple parties to compute the intersection size of their private sets without revealing additional information. PSI itself has been motivated by many real-world applications such as contact discovery [KRS+19]. Over the last several years PSI has become truly practical with extremely fast cryptographically secure implementations [CM20, RS21, PRTY20, GPR+21]. In the setting of two parties, PSI with post-processing (a.k.a circuit-based PSI), especially PSI-CA, has

*Arizona State University, {nitrieu,jhgao}@asu.edu

†VMware Research, ay.yanay@gmail.com

recently drawn more attention with several applications, such as measuring the effectiveness of online advertising [IKN+20], limiting the spread of Child Sexual Abuse Material (CSAM) [BBM+21], and private contact tracing related to COVID-19 [BBV+20, DPT20, DIL+20]. However, the state-of-the-art PSI-CA is only efficient in the two-party setting [IKN+20, DPT20, DIL+20]. This work considers a natural generalization to the multi-party setting, which opens the opportunity for richer applications, like the two we showcase below. The state-of-the-art protocol for PSI-CA in the multi-party setting [CDG+21] relies on secret-shared computation [DN07], which might be not scale well for a large number of parties. In this work we present, for the first time, a scalable protocol for PSI-CA in the multi-party setting.

Moreover, we present a new protocol, called DotProd, where n parties may compute a sum of element-wise products of their binary vectors without revealing any additional information. Mathematically, suppose party P_i holds the m-element vector x_i, then the parties obtain $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j]$, where $x_i[j]$ is the j^{th} element of the vector x_i. Note that in the two-party case, the computation is exactly of the dot product $x_1 \cdot x_2$. We demonstrate the efficiency of our protocols through two real-world applications: a COVID-19 heatmap computation based on PSI-CA and an associated rule learning (ARL) based on DotProd.

1.1 State-of-the-Art for PSI Cardinality

Private Set Intersection Cardinality (PSI-CA) is a variant of PSI in which the parties learn the intersection size and nothing else. In this work, we also focus on *server-aided* PSI-CA constructions. By "server-aided", we refer to cases where the parties perform PSI-CA computation with the help of distrusted cloud server(s). To the best of our knowledge, this work proposes the first special-purpose PSI-CA protocols that work in the multi-party setting.

We start with discussing PSI-CA works in the two-party setting. Clearly, one can use circuit-based PSI [PSTY19] to implement PSI-CA. However, this generic solution is expensive due to the secure computation inside the circuit. For the special-purpose two-party PSI-CA constructions, the work [IKN+20] extends the classic DH-based PSI protocol [Mea95] to support two-party PSI-CA by having a sender shuffle the PRFs of their items before returning to the receiver. Epione [TSS+20] also proposed a protocol that is suitable to the unbalanced, client-server setting, in which the server has a large database of m_1 items and the client has a small database of m_2 items. The protocol, however, requires $O(m_1 + m_2)$ expensive public-key operations (group exponentiation). Delegated PSI-CA [DPT20] improves the efficiency of the two-party PSI-CA protocol on the client's device, Catalic [DPT20] proposes a delegated system in which the client (i.e. PSI-CA receiver) can shift most of its PSI-CA computation to multiple untrusted servers while preserving privacy. However, Catalic system requires a least two non-colluding cloud servers with a heavy computation/communication cost. Dittmer et al. [DIL+20] introduces a variant of two-party PSI-CA (so-called weighted PSI-CA) in which each token of the client has an associated secret weight. The weighted PSI-CA is based on cheap Function Secret Sharing (FSS) constructions [BGI15, BGI16], thus it is efficient on both client's and server's sides. However, their construction assumes that there exist two non-colluding servers, each holding an identical input set.

A multi-party PSI-CA protocol was first proposed by Kissner and Song [KS05]. The protocol of [KS05] is based on oblivious polynomial evaluation which is implemented using additively homomorphic encryption. The basic idea is to represent a dataset as a polynomial whose roots are its elements, and send the homomorphic encryptions of the coefficients to other parties so that they can evaluate the encrypted polynomial on their inputs. The protocol of [KS05] has a quadratic computation and communication complexity in both the size of dataset and the number of parties.

[MRR20] proposed several protocols for database joins and PSI, but on secret shared data in the honest-majority three-party setting, which is different than the setting in this paper, as we consider a setting with any number of parties, in which the input does not have to be in a secret-sharing form.

Chandran et al. [CDG+21] proposed an efficient protocol for PSI (not PSI-CA), which can be extended to circuit-based PSI. Hence, one could combine their extended protocol with a circuit that computes the size of the intersection to obtain a protocol for PSI-CA. At the technical core, [CDG+21] is built on n-party secret-sharing functionalities introduced by [DN07]. Their use of generic secure computation protocol for a specific problem (of PSI-CA) makes their extended protocol less attractive. Specifically, their protocol involves many (parallel) invocations of secure multiplication over shared secrets, which on its own incurs 5 communication rounds. We compare the performance of our protocols and [CDG+21] in Section 6.2.

1.2 Secure Dot Product and Its Applications

Dot-product plays a key role in machine learning and data analysis tasks. Their implementation in a privacy-preserving setting remains expensive as it requires either generating Beaver triples [Bea91] or using fully homomorphic encryption (FHE). There is a long list of results for secure computation of dot product or linear algebra in general [DGD18, HLL+16, SBS19, ZWH+16, ZWY+17, DSZ15, BBH+20]. For the applications that we consider in this paper, namely, Covid-heatmap and ARL, dot-product of *sparse vectors* would be sufficient. Many algorithms for linear algebra operations, like matrix multiplication, leverage an apriori knowledge of the operands being sparse, and sometimes these algorithms can even be computed securely, without degrading their asymptotic complexity. None of the above works, however, address the problem of dot product in a setting where the vectors are sparse. The most relevant works to ours are [VC02, VC05, DC14, BBH+20, SGRP19], on which we elaborate immediately.

To the best of our knowledge, Vaidya and Clifton [VC02] were the first to study secure computation of scalar product of two m-element vectors in the two-party setting and its application to privacy-preserving association rule learning (ARL). Their dot product protocol heavily relies on public-key operations, and requires four communication rounds, communication complexity of $O(m)$ and computation complexity of $O(m^2)$.

Their follow-up work [VC05] is based on PSI, which makes the complexity dependent only of t, where t is the upper-bound on the Hamming weight of the vectors. They also propose a protocol for the multi-party setting, which requires a commutative one-way hash function so that the input from each party can be encrypted by a common set of keys. The resulting ciphertexts are the same if the original values are the same. Although efficient, their protocol introduces an undesirable leakage; specifically, it leaks the items in the intersection (rather only their sum). Moreover, their protocol is insecure when the input domain is relatively small (e.g. of size 2^{30}) as one party could easily perform a brute force attack [PSZ14]. To handle the latter security issue, [DC14] studied a two-party ARL and proposed a solution via a PSI protocol that is built on the Goldwasser-Micali (GM) Encryption [GM82] and Oblivious Bloom Intersection (OBI) [DCW13]. Their protocol, however, still leaks the items in the intersection, and became much more expensive than the protocol we present in this paper. In addition, they did not consider an extension to the multi-party case.

Recently, Bampoulidis et al. [BBH+20] studies COVID-19 heatmap computation and proposes secure dot product based on homomorphic encryption with several optimizations. However, the number of required HE operations is $O(m)$ (regardless of the Hamming weight of the vectors), which makes their protocol expensive. Schoppmann et al. [SGRP19] presents efficient two-party protocols for several common sparse linear algebra operations including sparse matrix-vector multiplication. The main building block of their protocols is a new functionality – Read-Only Oblivious Map (ROOM). Using ROOM, the cost of the secure matrix-vector multiplication is dependent only on the number of non-zero entries, instead of the operands' size. However, in all three ROOM constructions the parties invoke generic secure computation in order to obtain a secret-shared output. We compare the performance of our protocol to a ROOM-based dot-product in Section 6.1.

1.3 Our Results and Techniques

1.3.1 Our PSI-CA Approach:

We present a new multi-party PSI-CA protocol paradigm with an assumption that a subset of particular parties does not collude . We offer two variants of our protocol. The first protocol relies on a distrusted non-colluding server that has no input. It is optimized for the number of communication rounds between parties; that is, the protocol leverages a star network topology, where parties mostly communicate with the server. The second protocol removes the need of a server by reducing the problem of n-party PSI-CA to the problem of server-aided $(n-1)$-party PSI-CA with use of a distrusted party P_n *who may have an input.* The base case with $n = 2$ can be instantiated efficiently by two-party server-aided PSI protocol of Kamara et al. [KMRS14]. However, [KMRS14] is only for PSI itself (not PSI-CA)[1]. Thus, we simply their PSI protocol and present a new server-aided two-party PSI-CA in Section 4.1.

The main building blocks of our multiparty PSI-CA protocols are oblivious key-value store (OKVS) data structure [GPR+21], and/or Oblivious Programmable PRF (OPPRF) [KMP+17]. To this end, we propose a very simple and efficient protocol for server-aided OPPRF, which we believe to be of independent interest. Our server-aided OPPRF is based on a two-party server-aided shuffled OPRF, a functionality we formally define in Section 3.1.

We provide an implementation of server-aided and server-less variants of our PSI-CA approach for $n > 2$. To the best of our knowledge, this is the first 'special-purpose' implementation of multi-party PSI-CA that does not rely on generic secure computation. We find that multi-party PSI-CA is practical, by evaluating our protocols over settings with million items sets and 16 parties. The main reason for the efficiency of our protocol is its reliance on fast symmetric-key primitives. This is in contrast with prior multi-party PSI-CA protocols, which require expensive public-key operations for each item [KS05] or computation on secret-shared data [CDG+21].

Interestingly, the server-less PSI-CA variant is about 10× faster than the server-aided one. The two variants, however, offer different security guarantees. Specifically, in the former we require that neither of P_1, P_2 nor P_n is colluding, whereas in the latter we only require that P_1 and P_2 do not collude. In some sense, one may look at the server-less variant as a multi-server-aided PSI-CA but the servers have their private input. Hence, we can use our efficient server-aided OPPRF (instead of the two-party OPPRF [KMP+17] in the server-less PSI-CA protocol, which may explain why it is possible to get a better performance in this case. Concretely, in the server-less variant, we assign the non-colluding party P_1 the role of a server in the server-aided OPPRF protocol.

Note that in practice, a server-aided model can be reasonable. Performance is critical and often it makes sense given that the alternative has a weaker security guarantee. For example, in the federated learning setting, there is a server and many clients where the server helps training a machine learning model for the benefit of the clients (imagine an auto-complete enabled keyboard).

[1] Note that [KMRS14] has a protocol for multiparty PSI, but it reveals intersection items of each pair-wise parties sets to the server and is non-trivial to support PSI-CA.

In this work, we motivate our protocols with two real-world applications in which using a non-colluding, but distrusted server, makes complete sense. For example, in the Covid-19 heatmap computation, an established company (e.g. Google or Apple) can play the role of the server.

1.3.2 Our Multi-party Dot-Product (DotProd):

We propose a new protocol for computing the sum of element-wise products of n sparse binary vectors (so-called multiple dot product, DotProd). Let us begin with the simpler case, where $n = 2$, known as secure dot product. One would expect a solution for a dot product of m-elements vectors to incur communication overhead of at least $O(m)$, for the very fact that the parties need to first input those elements (which usually involves some sort of encryption or secret sharing on each element). In this work, we show that the communication and computation complexity is independent of m and can be reduced to $O(t)$, where t is the upper bound on the Hamming weight of the vectors. This improvement is significant when the vectors are sparse (i.e. $t = o(m)$).

For an m-element binary vector x we define $\mathbf{idx}(x) = \{i \in [m] \mid x[i] = 1\}$ to be the set of non-zero indices in x. Suppose the receiver P_0 and the sender P_1 hold an m-element binary sparse vector x_0 and x_1, respectively. The vectors are sparse and have the number of non-empty elements bounded by $t = o(m)$. As a very simple warm-up, we consider a non-secure dot product computation with the communication complexity cost of $O(t)$. Given the input vector x_0, the receiver computes $A_0 = \mathbf{idx}(x_0)$ and the sender computes $A_1 = \mathbf{idx}(x_1)$. The sender then sends A_0 to the receiver, who is able to compute the dot product $x \cdot y$ by computing the intersection $A_0 \cap A_1$ and outputting its cardinality $|A_0 \cap A_1|$.

The main advantage of the above solution is to reduce dependency on the length of the vectors, especially when the input vectors are sparse. To compute $x_0 \cdot x_1$ securely, the parties run a private set intersection cardinality protocol (PSI-CA) where P_0 inputs A_0 and P_1 inputs A_1. This idea, however, has received little attention due to the large overhead required to compute PSI-CA. In this work, we propose a lightweight server-aided PSI-CA construction to improve the performance of the secure dot product. Our two-party protocol relies on only PRF. As a result, our protocols are simple and efficient, with a communication and computation complexity $O(t)$, so is our secure dot product DotProd.

We then extend DotProd to the multi-party case. Given an input vector x_i, party P_i computes $A_i = \mathbf{idx}(x_i)$. It is easy to see that the sum of element-wise products of the vectors is equal to the size of their intersection, namely, $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j] = |\bigcap_{i=1}^{n} A_i|$. We implement the multi-party DotProd using our multi-party PSI-CA.

1.3.3 Application to PSI-CA and DotProd:

We show that our PSI-CA and DotProd techniques can be used to implement and improve the performance of several privacy-preserving applications. More specifically, we consider two running examples: COVID-19 heatmap computation and associated rule learning (ARL).

In the COVID-19 heatmap problem, we consider a scenario where the Department of Health and Human Services (HHS) wants to learn areas with a higher chance of getting infected with the disease without knowing the travel route of infected individuals. The heatmap can be implemented by computing the vector-matrix multiplication as $x^\top Y$, where x and Y are as follows: x is a binary vector of size N, held by the HHS, such that $x[i] = 1$ if the ith user has tested positive to COVID-19 and $x[i] = 0$ otherwise; and $Y = (y_1, \ldots, y_m) \in \mathbb{Z}_2^{N \times m}$ is a user-location matrix, held by a network operator, such that the ith element of the column vector y_j indicates whether the ith user has recently visited the jth location. In that case $y_j[i] = 1$ and otherwise $y_j[i] = 0$. Clearly, $z = x^\top Y$ is an m-element vector where the ith element is equal to the number of users who have tested positive and recently visited the ith location. [BBH+20] proposes different optimizations on HE to implement a secure dot product, which still requires $O(Nm)$ independent multiplications (regardless of the Hamming weight of the vectors). In the heatmap example above, we observe that the vector x is sparse because the proportion of diagnosed individuals per day among all N subscribed individuals is small (e.g, 0.01-1% would be a large percentage [wor21]). Similarly, the matrix Y is also sparse due to people's localized travel habits. In Section 5.2, we apply our DotProd protocol to compute COVID-19 heatmap. In addition, [BBH+20] only supports a two-party computation between the HHS and a network provider. In real-world scenarios, there are many network providers. We modify our two-party PSI-CA protocol to support heatmap computation between the HHS and multiple network providers without revealing additional information.

Second, we study associated rule learning (ARL) as an application of DotProd. ARL is a rule-based machine learning method that is used to discover rules/relations of the type $(X \Rightarrow Y)$ between variables X, Y in databases. As a typical example in the sales database of a supermarket, a rule/relation {onions, potatoes ⇒ burger} indicates that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. In market design, such information can be used as the basis for decisions about product placements, promotional pricing, and more. However, the ARL training process requires a large transaction database, which may be collected from different sources. Thus, it is highly desirable to maintain the privacy of each source. We study a common ARL training algorithm, called Apriori [AIS93, Rud12], and adapt it to the privacy-preserving setting. Most steps in Apriori can be computed locally except a step in which the parties want to compute a confidence score of how many transactions across a joint database that contains all attributes/items in both X and Y. This step can be implemented by computing a sum of bit-wise products of multiple binary vectors. We first apply multi-party DotProd for ARL and make its learning process in a privacy-preserving manner.

2 Preliminaries

In this work, the computational and statistical security parameters are denoted by κ, λ, respectively. We use $[x]$ to denote the set $\{1, 2, \ldots, x\}$ and $[x, y]$ to denote the set $\{x, x+1, \ldots, y\}$. We denote the concatenation of two bit strings x and y by $x||y$. For a pseudorandom function (PRF) F, a key k and a set A, we define $F(k, A) = \{F(k, a) \mid a \in A\}$. For an m-element binary vector x, we define $\mathbf{idx}(x) = \{i \in [m] \mid x[i] = 1\}$ to be the set of non-zero indices in x.

Remark: Throughout the text we keep using the term PRF even though the actual implementation uses a pseudorandom permutation (PRP). Our use of a PRP is due to the fact that we want to allow the simulator, in our proofs, to run F^{-1}. We keep the term PRF mostly because the underlying primitives that we use (like OPRF and OPPRF) are known by that name and we opt to make it easier to the reader who is already familiar with them.

We define the functionality $\mathcal{F}_{\mathsf{Coin}}$ for coin tossing between any number of parties. The result from $\mathcal{F}_{\mathsf{Coin}}$ may be either a PRF key or a random value in any format, depending on the context. The malicious $\mathcal{F}_{\mathsf{Coin}}$ can be achieved by running a simulatable coin tossing protocols of [KOS03, Lin01], which tolerate dishonest majority.

2.1 Security Model

Secure computation allows mutually untrusted parties to jointly compute a function on their private inputs without revealing any additional information. There are two classical security models: colluding model is modeled by considering a single monolithic adversary that captures the possibility of collusion between the dishonest participants; and non-colluding model is modeled by considering independent adversaries, each captures the view of each independent dishonest party. There are also two adversarial models, which are usually considered. In the semi-honest model, the adversary is assumed to follow the protocol, but may try to learn information from the protocol transcript. In the malicious model, the adversary follows an arbitrary polynomial-time strategy to learn additional information.

In this work, we consider various adversarial models and assume that a subset of particular parties does not collude. In that sense, this work slightly deviates from the well-known 'threshold security'; instead, we consider a specific, but general enough access structure, in which we assume that some specific subset of parties does not collude. While acknowledging this deviation, we do believe our approach can be used as a stepping stone toward the achieving security in the 'standard' threshold access structure.

2.2 Oblivious Key-Value Store (OKVS)

A Key Value Store (KVS) consists of two algorithms: i) Encode takes as input a set of (k_i, v_i) key-value pairs from the key-value domain, $\mathcal{K} \times \mathcal{V}$, and outputs an object S (or, with negligible probability, an error indicator $\perp$); ii) Decode takes as input an object S, a key x and outputs a value y.

A KVS is correct if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys: i) $Pr[\mathsf{Encode}(A) = \perp]$ is negligible, and ii) if $\mathsf{Encode}(A) = S \neq \perp$ and $(k, v) \in A$ then $\mathsf{Decode}(S, k) = v$.

We say that a KVS is oblivious if for all $\mathcal{K}_1, \mathcal{K}_2$ of size m and all PPT adversaries $\mathcal{A}$: $\big|\Pr[\mathsf{Exp}^{\mathcal{A}}(\mathcal{K}_1)] - \Pr[\mathsf{Exp}^{\mathcal{A}}(\mathcal{K}_2)]\big| \leq \mathsf{negl}(\kappa)$. In other words, if the values v_i are chosen uniformly then the output of Encode hides the choice of the keys k_i. Oblivious Key-Value Store (OKVS) [GPR+21] is given in Experiment 2.2.1.

EXPERIMENT 2.2.1. ($\mathsf{Exp}^{\mathcal{A}}(\mathcal{K} = (k_1, \dots, k_m))$)

1. for $i \in [m]$: choose uniform $v_i \leftarrow \mathcal{V}$
2. return $\mathcal{A}\big(\mathsf{Encode}(\{(k_1, v_1), \dots (k_m, v_m)\})\big)$

2.3 Oblivious PRF (OPRF) and Programmable PRF (OPPRF)

An oblivious PRF (OPRF) [FIPR05] is a 2-party protocol in which the sender learns a PRF key k and the receiver learns $F(k, q_1), \dots, F(k, q_m)$. Here, F is a PRF and $(q_1, \dots, q_m)$ are inputs chosen by the receiver. Functionality 1 presents a variant of OPRF where the receiver obtains outputs of multiple statically chosen queries.

FUNCTIONALITY 1. (*Oblivious PRF -* $\mathcal{F}^m_{\mathsf{oprf}}$)
Parameters: A PRF F, and a bound m on the number of queries.
Behavior: Wait for input $(q_1, \dots, q_m)$ from the receiver where $q_i \in \{0,1\}^\kappa$. Sample a random PRF key k and give it to the sender. Give $\{F(k, q_1), \dots, F(k, q_m)\}$ to the receiver.

An oblivious programmable PRF (OPPRF) [KMP+17] functionality is given in Functionality 2. It is similar to the plain OPRF functionality except that (1) it allows the sender to initially provide a set of points $\mathcal{P}$ which will be programmed into the PRF; (2) it additionally gives the "hint" value to the receiver. OPPRF constructions for both the semi-honest and malicious setting were proposed by Kolesnikov et. al. [KMP+17] and by Pinkas et. al. [PRTY20].

FUNCTIONALITY 2. (*Oblivious Programmable PRF* - $\mathcal{F}_{\mathsf{opprf}}^{m_1,m_2}$)
PARAMETERS: A PRF F, an upper bound m_1 on the number of points to be programmed, and a bound m_2 on the number of queries.

BEHAVIOR: Wait for input $\mathcal{P} = \{(a_1, t_1), \dots, (a_{m_1}, t_{m_1})\}$ from the sender $\mathcal{S}$ and input $(q_1, \dots, q_{m_2})$ from the receiver $\mathcal{R}$. Run $(k, \mathsf{hint}) \leftarrow \mathsf{KeyGen}(\kappa, \mathcal{P})$. Give (k, hint) to $\mathcal{S}$ and $(\mathsf{hint}, F(k, \mathsf{hint}, q_1), \dots, F(k, \mathsf{hint}, q_{m_2}))$ to $\mathcal{R}$.

2.4 Unconditional Zero Sharing

The unconditional zero sharing gives the parties with a sharing function $S : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\kappa$ and a key K_i for party P_i, such that for every $x \in \{0,1\}^\ell$, we have that $s_i = S(K_i, x)$ is P_i's random share, and $\bigoplus_{i=1}^n s_i = 0$. The functionality and its construction from [KMP+17] are given in Functionality 3 and Protocol 7.

FUNCTIONALITY 3. (*Zero-Sharing* - $\mathcal{F}_{\mathsf{ZS}}$)
PARAMETERS: n parties. The dictionary store is initialized to $\emptyset$.

BEHAVIOR: Upon an input x from P_i, if $\mathsf{store}[x]$ does not exist, generate random values $s_1, \dots, s_n$ s.t. $\bigoplus_{i=1}^n s_i = 0$ and store $\mathsf{store}[x][i] = s_i$ for $i \in [n]$. Output $\mathsf{store}[x][i]$ to P_i.

2.5 Private Set Intersection Cardinality

Private set intersection cardinality (PSI-CA) allows n parties, each holding a set of m items, to learn the intersection size of their private sets without revealing anything else. In the server-aided PSI-CA, we assume there is an untrusted server that has no input and does not collude with either of the parties. The server involves in the PSI-CA computation while learning nothing. PSI-CA and server-aid PSI-CA are formally presented in Functionality 4. The highlighted text is required for the server-aided case.

FUNCTIONALITY 4. (*PSI Cardinality* - $\mathcal{F}_{\mathsf{PSI-CA}}$)
PARAMETERS: n parties $P_1, \dots, P_n$; an untrusted server $\mathcal{C}$; the set size m.
FUNCTIONALITY:

- Wait for input set X_i of size m from the party P_i
- Give the server $\mathcal{C}$ nothing
- Give P_1 an intersection set size $|\bigcap_{i=1}^n X_i|$

2.6 Secure Dot Product

Secure dot product functionality allows n parties, each holding an m-element binary vector, to learn the dot product of their private vectors without revealing any additional information. In this work, we consider the problem of the secure dot product of n binary vectors, in a server-aided setting, in which we make use of a non-colluding distrusted server. Our protocols are extremely efficient when the upper bound on the Hamming weight of the vectors, denoted t, is in $o(m)$. The dot product of n vectors $x_1, \dots, x_n$, each with m elements, is defined by $\sum_{j=1}^m \prod_{i=1}^n x_i[j]$ and is called DotProd. DotProd is presented in Functionality 5. The highlighted text is required for the server-aided case.

FUNCTIONALITY 5. (*Secure Dot Product* - $\mathcal{F}_{\mathsf{DotProduct}}$)

PARAMETERS: n parties: $P_{i \in [n]}$; an untrusted server $\mathcal{C}$; an upper-bound t.

FUNCTIONALITY:

- Wait for input m-element binary vector x_i from P_i
- Give the server $\mathcal{C}$ nothing
- Give to $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j]$ the party P_1

3 Server-Aided OPRF and OPPRF

In this section, we introduce new constructions which make use of a semi-honest non-colluding cloud server. The functionalities of OPRF and OPPRF are as described in Sections 2.3, except that now the functionality involves a third party (the server) who has neither input nor output (except the size of the other parties' inputs).

3.1 Server-Aided Shuffled OPRF

The functionality ($\mathcal{F}_{\mathsf{oprf}}^{(m)}$). The server-aided OPRF functionality involves $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{C}$ and is defined as follows: the sender $\mathcal{S}$ has a key K, the receiver $\mathcal{R}$ has a set of queries $\{y_i\}_{i \in [m]}$ and the cloud server $\mathcal{C}$ has no input. $\mathcal{S}$ does not have an output whereas $\mathcal{R}$ obtains $\{y'_{\pi(1)}, \ldots, y'_{\pi(m)}\}$ where $y'_i = F(K, y_i)$ and π is a uniformly random permutation chosen by $\mathcal{C}$. The output of $\mathcal{C}$ is that permutation π. Clearly, $\mathcal{R}$ cannot associate the response y'_i with the query y_i as all responses are pseudorandom.

The protocol. Define $F'((k_1, k_2), x) = F(k_2, F(k_1, x))$ where F is a PRF. It is easy to see that F' is a PRF. $\mathcal{S}$ has the key $K = (k_1, k_2)$, $\mathcal{R}$ has an input set $Y = \{y_1, \ldots, y_m\}$ which serves as its set of OPRF queries. Then,

1. $\mathcal{S}$ sends k_1 to $\mathcal{R}$ and k_2 to $\mathcal{C}$.

2. $\mathcal{R}$ computes $Y' = F(k_1, Y)$ and sends Y' to $\mathcal{C}$.

3. $\mathcal{C}$ computes $Y'' = F(k_2, Y')$ and sends a random permutation π of Y'' to $\mathcal{R}$.

Correctness. It is clear that $\mathcal{R}$ obtains $y' = F'(K, y) = F(k_2, F(k_1, y))$ for every query $y \in Y$. These values are given to $\mathcal{R}$ in a random order, chosen by $\mathcal{C}$.

Security. Obviously $\mathcal{S}$ learns nothing because it does not receive any message during the protocol. $\mathcal{C}$ learns nothing because it only receives n pseudorandom values as queries. $\mathcal{R}$ learns nothing but a random permutation on the PRF results on its queries. $\mathcal{R}$ also learns one of two random PRF keys $K = (k_1, k_2)$. A simulator Sim for $\mathcal{R}$ picks $K = (k_1, k_2)$, gives k_1 to $\mathcal{R}$ and answers each of $\mathcal{R}$'s queries, y', by $F(k_2, y')$. The distributions of the simulated and real views of the $\mathcal{R}$ are indistinguishable.

3.2 Server-Aided OPPRF

The functionality $\mathcal{F}_{\mathsf{opprf}}^{(m_1, m_2)}$. The server-aided OPRF functionality involves $\mathcal{S}$, $\mathcal{R}$ and $\mathcal{C}$ and is defined as follows: The sender $\mathcal{S}$ has a set of m_1 points $\mathcal{P} = \{(x_i, v_i)\}_{i \in [m_1]}$ with (pseudo)random v_i's, $\mathcal{R}$ has a set $Y = \{y_i\}_{i \in [m_2]}$, and the cloud server $\mathcal{C}$ has no input. Denote the set of first (resp.

second) entries of the pairs in $\mathcal{P}$ by X (resp. V). $\mathcal{S}$ and $\mathcal{C}$ do not have an output whereas $\mathcal{R}$ obtains v_j iff $y_j \in V$, and other pseudo-random value otherwise.

The protocol. The server-aided OPPRF is similar as above, except that the server does not do the permutation and there is one extra message from $\mathcal{S}$ to $\mathcal{R}$ in which $\mathcal{S}$ gives a set of "corrections" to $\mathcal{R}$ so it can adjust its pseudorandom results into results from V. Formally, the protocol is:

1. $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ jointly invoke $\mathcal{F}_{\mathsf{oprf}}^{(m_2)}$ where:
   - $\mathcal{S}$ inputs a random key $k = (k_1, k_2) \leftarrow \{0,1\}^{2\kappa}$ for PRF F' (see section above).
   - $\mathcal{C}$ inputs $\perp$
   - $\mathcal{R}$ inputs Y, and obtains a set $Y' = \{y'_1, \ldots, y'_{m_2}\}$ as output, where $y'_i = F'(k, y_i)$. Note that we use a non-shuffled version of OPRF.
2. $\mathcal{S}$ constructs an OKVS over $T \leftarrow \mathsf{Encode}(\{(x_i, F'(k, x_i) \oplus v_i\}_{i \in [m_1]})$ and sends T to $\mathcal{R}$.
3. For every $i \in [m_2]$, $\mathcal{R}$ outputs $y'_i \oplus \mathsf{Decode}(T, y_i)$

Correctness. For every $y \in X \cap Y$ note that $\mathcal{R}$ obtains $v = F'(k, y)) \oplus \mathsf{Decode}(T, y)$ where $v \in V$ is associated with y. For every other value it obtains $F'(k, y) \oplus r$ where r is a pseudo-random value decoded from T.

Security. Obviously $\mathcal{S}$ learns nothing because it does not receive any message during the protocol. $\mathcal{R}$ learns nothing except the value associated with the items in $X \cap Y$ since it queries the OPRF only on X and the decoding 'makes sense' only for those items in $X \cap Y$.

4 PSI Cardinality Protocol

In this section we present three protocols:

- In Section 4.1, we simplify the server-aided PSI protocol of [KMRS14] and formally present a new server-aided two-party PSI-CA protocol. Unlike previous "server-less" protocols (see Section 1.1) that rely on OT-based oblivious [DPT20] or DH-based [TSS+20, IKN+20] which, in turn, are based on public-key operations, our two-party PSI-CA protocol uses only symmetric-key operations. This is possible, among other improvements, due to the replacement of their OPRF constructions with a server-aided version, which is much simpler and more efficient.
- In Section 4.2, we show an extension of the protocol to the multiparty case, where the adversary may actively corrupt (almost) any strict subset of the parties or passively corrupt the server. To the best of our knowledge, this is the first 'special-purpose' protocol for privately computing the intersection cardinality of more than two parties, for which we present interesting applications (see Section 5).
- In Section 4.3, we show that a server is not necessary when some parties are assumed to be semi-honest and non-colluding.

PROTOCOL 1. (*Server-Aided Two-party PSI-CA*)

Parameters:

- Set size m_1, m_2.
- A sender $\mathcal{S}$, a receiver $\mathcal{R}$, a non-colluding semi-honest server $\mathcal{C}$
- A PRF F that is used by the $\mathcal{F}_{\mathsf{oprf}}$ functionality (see Section 3.1). Let $\mathcal{K}$ be its key-space.

Inputs:

- Sender $\mathcal{S}$ has input $X = \{x_1, \ldots, x_{m_1}\}$
- Receiver $\mathcal{R}$ has input $Y = \{y_1, \ldots, y_{m_2}\}$
- Cloud server $\mathcal{C}$ has no input.

Protocol:

1. $\mathcal{S}$, $\mathcal{R}$, and $\mathcal{C}$ jointly invoke $\mathcal{F}_{\mathsf{oprf}}^{(m_2)}$ where:
    - $\mathcal{S}$ inputs a random key $k \leftarrow \mathcal{K}$.
    - $\mathcal{C}$ inputs $\perp$
    - $\mathcal{R}$ inputs Y, and obtains a set $Y' = \{y'_{\pi(1)}, \ldots, y'_{\pi(m_2)}\}$ as output, where $y'_{\pi(i)} = F(k, y_i)$ and π is a random permutation.
2. $\mathcal{S}$ sends a random permutation of $X' = F(k, X)$ to $\mathcal{R}$.
3. $\mathcal{R}$ outputs $|X' \cap Y'|$.

4.1 Server-Aided Two-Party PSI-CA

We consider sender $\mathcal{S}$ and receiver $\mathcal{R}$ who want to compute the intersection size of their private sets $X = \{x_1, \ldots, x_{m_1}\}$ and $Y = \{y_1, \ldots, y_{m_2}\}$, respectively. To do so, they use a non-colluding, semi-honest cloud server $\mathcal{C}$. The formal description is given in Protocol 1. The protocol is inspired by the size-hiding server-aided PSI of [KMRS14]. For completeness, a description of their PSI protocol is given in Appendix D.

For correctness, notice that for a value $z \in X \cap Y$, the value $F(k, z)$ appears in both X' and Y'. On the other hand, if $z \notin X$ then $F(k, z) \notin X'$; and if $z \notin Y$ then $F(k, z) \notin Y'$.

Security follows since (see formal proof below):

- $\mathcal{R}$ cannot associate $y'_{\pi(i)}$ with y_i since π is a secret random permutation.
- For a value $x_j \in X$, the value $F(k, x_j)$ looks random to $\mathcal{R}$ if $\mathcal{R}$ did not input it as a query to the shuffled-OPRF functionality.

The protocol is extremely efficient because of the efficiency of the shuffled $\mathcal{F}_{\mathsf{oprf}}$. In terms of communication cost, it only requires $\mathcal{S}$ to send m_1 values to $\mathcal{R}$. The construction for $\mathcal{F}_{\mathsf{oprf}}$, in turn, requires only m_2 messages from $\mathcal{R}$ to $\mathcal{C}$ and m_2 messages back from $\mathcal{C}$ to $\mathcal{R}$.

Theorem 1. *Protocol 1 securely implements Functionality 4 for the case of two parties (i.e. $n = 2$) in the $\mathcal{F}_{\mathsf{oprf}}$-hybrid model, in the presence of an adversary who may actively corrupt either $\mathcal{S}$ or $\mathcal{R}$, or passively corrupt $\mathcal{C}$.*

Proof. We separate the proof to three cases, for each party that the adversary may corrupt. We denote the simulator by Sim. Note that the set size m_1 and m_2 are known to Sim.

Corrupted $\mathcal{S}$. The simulator Sim runs $\mathcal{S}$ internally, in which it plays the role of the shuffled-OPRF functionality. It receives k from $\mathcal{S}$ and later it observes the set X' sent from $\mathcal{S}$ to $\mathcal{R}$. The simulator Sim computes $L' = F^{-1}(k, X')$ and sends to the PSI-CA functionality the input set $L = \{i \in L' \mid i \in [m_1]\}$.

Corrupted $\mathcal{R}$. The simulator Sim runs $\mathcal{R}$ internally and plays the role of the shuffled-OPRF functionality. It obtains the set Y from $\mathcal{R}$, picks a random PRF key k and sends $Y' = F(k, Y)$ to $\mathcal{R}$. Sim forwards Y to the PSI-CA functionality, from which it receives $c = |X \cap Y|$. Then Sim constructs $X' = \{x'_1, \ldots, x'_{m_1}\}$ as follows: for $i \in [c]$ set $x'_i = y'_i$ and for $i \in [c+1, m_1]$ set x'_i as a uniformly random value chosen from the range of F. Finally, Sim sends a random permutation of X' to $\mathcal{R}$. The view of $\mathcal{R}$ in the simulated view and in the real view are computationally indistinguishable, since in both views it receives two messages with two sets of pseudo-random values, where the intersection cardinality of the two sets is equal c and the indices of items in the intersection are uniformly random in $[m_1]$.

Corrupted $\mathcal{C}$. The cloud server $\mathcal{C}$ has neither input nor output and therefore its simulated view contains only the random permutation that is output from the OPRF functionality. This can be simulated by picking a random permutation $\pi : [m_2] \to [m_2]$. □

4.2 Server-Aided Multi-Party PSI-CA

In this section, we assume that all parties have the same set size m. Protocol 2 may be seen as if we have one receiver, who is P_1, and multiple senders, who are $P_2, \ldots, P_n$. The role of the server is to shuffle PRF results from the senders before delivering them to the receiver. As a simplification to Protocol 2, suppose that we want the receiver to obtain $n-1$ shares of zero for every each of its items that is in the intersection. This can be done by querying the senders on each of its items and collecting the results. Each sender programs the responses such that if the query is on one of its items, then it responds with its (pseudorandom) share of zero, otherwise, it responds with some other pseudorandom value. When given the senders responses on a query, if the responses are valid shares of the value zero then the receiver knows that its query is in the intersection. Since the server shuffles the responses to the queries, the receiver does not know, for a given set of responses which are shares of zero, to which query it is associated, thus, the output leaks nothing but the intersection size. Formally,

1. $P_2, \ldots, P_n$ (the senders) generate keys for a zero sharing function S, so P_i obtains K_i such that for every x it holds that $\bigoplus_{i=2}^{n} S(K_i, x) = 0$.
2. P_1 (the receiver) sends to the server its OPPRF queries X_1.
3. The server runs an OPPRF instance with every sender, using the queries X_1. A sender P_i ($i \in [2, n]$) programs the responses such that on query $x \in X_i$ the response is $S(K_i, x)$ whereas on any other query the response is another pseudorandom value.
4. The server obtains the set $Y'_{i \in [2,n]}$, of $n-1$ OPPRF responses, on every query $x_i \in X_1$. It chooses a random permutation $\pi : [m] \to [m]$ and sends to P_1 the set $\{Y'_{\pi(1)}, \ldots, Y'_{\pi(m)}\}$.
5. P_1 checks for every response set Y_i whether its values are valid shares of zero. If so, it adds 1 to the cardinality.

In the above simplification, there are several security issues: first, the server learns P_1's queries in the clear; second, the server mediates all PRF responses and therefore it learns whenever there is a set of responses that are valid shares of zero, thus it can learn the intersection size as well; third, if the receiver colludes with one of the senders, together they can reverse the server's permutation

PROTOCOL 2. (*Server-Aided Multi-Party PSI-CA*)

PARAMETERS:

- Parties $P_1, \ldots, P_n$ for $n > 2$, and a cloud server $\mathcal{C}$.
- A PRF $F : (\{0,1\}^\star, \{0,1\}^\kappa) \to \{0,1\}^\kappa$.
- An OPPRF $\mathcal{F}_{\mathsf{opprf}}$ described in Section 2.3.

INPUTS:

- P_i has $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$.
- Cloud server $\mathcal{C}$ has no input .

PROTOCOL:

1. Parties $P_2, \ldots, P_n$ invoke $\mathcal{F}_{\mathsf{ZS}}$ (Functionality 3) and each party P_i obtains the key K_i for a sharing function S.
2. Parties P_1 and P_2 agree on m random values $\Gamma = (\gamma_1, \ldots, \gamma_m)$ using $\mathcal{F}_{\mathsf{Coin}}$.
3. Parties $P_1, \ldots, P_n$ agree on a random PRF key k using $\mathcal{F}_{\mathsf{Coin}}$.
4. Party P_i for $i \in [2, n]$ computes the set of points $\mathcal{P}_i$ where:
   - $\mathcal{P}_2 = \big\{\big(F(k, x_{2,j}), S(K_2, x_{2,j}) \oplus \gamma_{\pi(j)}\big)\big\}_{j \in [m]}$ where $\pi : [m] \to [m]$ is a random permutation chosen by P_2.
   - For $i \in [3, n]$, $\mathcal{P}_i = \big\{\big(F(k, x_{i,j}), S(K_i, x_{i,j})\big)\big\}_{j \in [m]}$.
5. P_1 sends $X'_1 = F(k, X_1) = \{F(k, x_{1,j})\}_{j \in [m]}$ to $\mathcal{C}$.
6. $\mathcal{C}$ and P_i (for every $i \in [2, n]$) invoke $\mathcal{F}_{\mathsf{opprf}}$, where P_i acts as a sender with input $\mathcal{P}_i$ and $\mathcal{C}$ acts as a receiver with input X'_1. $\mathcal{C}$ obtains the result $y_{i,j}$ on the query $x_{1,j}$.
7. For every $j \in [m]$, $\mathcal{C}$ computes $w_j = \bigoplus_{i=2}^n y_{i,j}$ and sets W to be a random permutation of $\{w_1, \ldots, w_m\}$. $\mathcal{C}$ sends W to P_1.
8. P_1 outputs $|W \cap \Gamma|$.

on items that are in the intersection and by that leak the intersection itself (rather than only its size).

The first issue is easily solved by having all parties $P_1, \ldots, P_n$ agree on a PRF key k, so instead of computing $|\bigcap_{i=1}^n X_i|$ their objective is to compute $|\bigcap_{i=1}^n F(k, X_i)|$. This way, the server does not know P_1's set. Hiding the intersection size from the server (the second issue above) is trickier. We solve it by having P_1 and P_2 agree on a set of random values $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ so that instead of programming the responses with the 'zero shares', on a value $x \in X_2$, P_2 programs the response $S(K_2, x) \oplus \gamma$ for some $\gamma \in \Gamma$. Now, for items that are in the intersection, the server $\mathcal{C}$ sees a set of responses that constitutes a valid share of some $\gamma \in \Gamma$, but since the $\mathcal{C}$ does not know Γ, this looks random indistinguishable from the responses on values that are not in the intersection. Finally, we propose a protocol under a relaxed setting, that solves the third issue above. Concretely, the protocol is secure as long as P_1 and P_2 do not collude. This is done by adding one step to the above description: before the server forwards the responses set W to P_1, it sums its items and forwards only the sum to P_1. This means that now P_i ($i \geq 3$) could not trace back and learn the intersection itself. On the other hand, if P_1 and P_2 collude then they can still learn the intersection. This is formally presented in Protocol 2.

Correctness. We consider three following cases based on whether x is in the intersection of all sets $X_{i \in [n]}$:

- Case 1: Suppose $x \in \bigcap X_{i \in [n]}$. In other words, $\forall i \in [n], \exists x_{i,j_i} \in X_i$, such that $x_{i,j_i} = x$. Thus, we have (i) all PRF values $x'_{i,j_i} = F(k, x_{i,j_i}) = F(k, x)$ are equal, (ii) XORing all zero shares $S(K_i, x_{j_i}), i \in [2, n]$ is equal to zero. When querying the OPPRF programmed $\mathcal{P}_{i \in [3,n]}$ using the common PRF value $x'_1 = F(k, x)$, the cloud server obtains $y_{i,j}$. Based on the correctness of

OPPRF, we have $y_{i,j_i} = S(K_i, x_{j_i})$. In addition, the cloud server obtains $y_{2,j_2} = S(K_2, x_{j_2}) \oplus \gamma_{j_2}$ when querying on x'_1. Therefore, the value $w_j = \bigoplus_{i=2}^{n} y_{i,j_i}$ is equal to γ_{j_2} which belongs to the set Γ known by P_1. Thus, P_1 can count how many w_j in Γ to output the intersection size.

- Case 2: Suppose x is in X_1 and is not an element in some sets $X_{i\in[2,n]}$. Some OPPRF output $y_{i,j}$ is a random value since $F(k, x)$ was never used in the OPPRF programming process. Therefore, $w_j = \bigoplus_{i=2}^{n} y_{i,j}$ is random and does not belong to the set Γ.
- Case 3: Suppose x is an element in some sets $X_{i\in[2,n]}$, but not in X_1. Some OPPRF output $y_{i,j}$ is a random value. Therefore, $w_j = \bigoplus_{i=2}^{n} y_{i,j}$ is random and does not belong to the set Γ.

We note that, in our protocol, parties use zero shares to mask their actual input. This step is similar to the one in malicious schemes [GPR+21]. Thus, it is easy to prove that the use of $\mathcal{F}_{\mathsf{ZS}}$ in our protocol is secure in the malicious model.

Theorem 2. *Protocol 2 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{opprf}}, \mathcal{F}_{\mathsf{ZS}}, \mathcal{F}_{\mathsf{Coin}}$)-hybrid model, in the presence of an adversary who may actively corrupt any strict subset of $\{P_1, P_3, \ldots, P_n\}$ or $\{P_2, P_3, \ldots, P_n\}$ or passively corrupt the cloud server $\mathcal{C}$.*

The security proof of the above theorem is given in Appendix A.

4.3 Multi-party PSI-CA

In this section, we describe our "server-less" multi-party PSI-CA protocol. The main idea is to convert the problem of n-party server-aided PSI-CA to the problem of $(n-1)$-party with the use of an untrusted party P_n who, however, has a private input set X_n. Recall that in the server-aid PSI-CA protocol, the cloud server $\mathcal{C}$ has no input, but obtains from P_1 the PRF values $F(k, X_1)$ which are used to invoke an OPPRF with parties $P_{i\in[2,n]}$. In the problem of $(n-1)$-parties, however, the simulated cloud server/party P_n does have input X_n. Thus, P_n can compute its PRF values $F(k, X_n)$ on its own since it knows k. Similar to the server-aided version, P_n computes the exclusive-or of the OPPRF results and its zero share $S(K_n, x_{n,j})$, with the j-th result denoted by w_j. Note that w_j is equal to γ_j if all parties $P_{i\in[2,n]}$ has $x_{n,j}$, otherwise, w_j is random. At this point, if P_n sends all values w_j to P_1 as before, P_1 can only compute the intersection size of $n-1$ sets $\bigcap_{i=2}^{n} X_i$ since there was nothing to do with the input set X_1.

To have P_1 output $|\bigcap_{i=1}^{n} X_i|$, we propose the following steps. Instead of using a random set Γ in Step (2) of Protocol 2, P_2 uses PRF to compute $\gamma_j \leftarrow F(s, x_{2,j})$ where s is known by only P_1 and P_2. We observe that if $x_{1,j}$ is an intersection item, the corresponding PRF value $F(s, x_{1,j})$ should be equal to a value w_k hold by P_n because of $w_k = \gamma_k = F(s, x_{2,k}) = F(s, x_{1,j})$. Therefore, the intersection size $|\bigcap_{i=1}^{n} X_i|$ can be computed by counting how many PRF values $F(s, x_{1,j})$ are in the set $W = \{w_1, \ldots, w_n\}$. P_1 and P_n can do this by invoking a two-party PSI-CA, where P_1 acts as a receiver with an input set $\{F(s, X_1)\}$ and P_n acts as a sender with an input set W.

We implement the two-party PSI-CA using our server-aid protocol described in Protocol 1 in which any party $P_{i\in[2,n-1]}$ (say P_2) can play the role of the cloud server. The two party PSI-CA Protocol 1 requires that both sender and receiver do not collude with the semi-honest server. Thus, in our multi-party protocol, we assume that P_2 is semi-honest and non-colluding with both P_1 and P_n. In addition, given this assumption, we can improve the performance of our multi-party OPPRF. Particularly, unlike Protocol 2 in the above section, we use our server-aided OPPRF construction described in Section 3.2 to execute an OPPRF instance between P_n and each $P_{i\in[2,n-1]}$, where P_1 plays the role of the OPPRF server (thus, P_1 is non-colluding). We formally present our server-less multi-party PSI-CA in Protocol 3.

PROTOCOL 3. (*Multi-Party PSI-CA*)

PARAMETERS:

- Parties $P_1, \ldots, P_n$ for $n > 2$.
- A PRF $F : (\{0,1\}^\star, \{0,1\}^\kappa) \rightarrow \{0,1\}^\kappa$.
- An server-aided $\mathcal{F}_{\mathsf{opprf}}$ described in Section 3.2

INPUTS: P_i has $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$.

PROTOCOL:

1. Parties $P_2, \ldots, P_n$ invoke $\mathcal{F}_{\mathsf{ZS}}$ (Functionality 3) and each party P_i obtains the key K_i for a sharing function S.
2. Parties P_1 and P_2 agree on a random PRF key s using $\mathcal{F}_{\mathsf{Coin}}$.
3. Parties $P_2, \ldots, P_n$ agree on a random PRF key k using $\mathcal{F}_{\mathsf{Coin}}$.
4. Party P_i for $i \in [2, n-1]$ computes the set of points $\mathcal{P}_i$ where:
   - $\mathcal{P}_2 = \big\{\big(F(k, x_{2,j}), S(K_2, x_{2,j}) \oplus F(s, x_{2,j})\big)\big\}_{j\in[m]}$.
   - For $i \in [3, n-1]$, $\mathcal{P}_i = \big\{\big(F(k, x_{i,j}), S(K_i, x_{i,j})\big)\big\}_{j\in[m]}$.
5. P_n and P_i (for every $i \in [2, n-1]$) invoke an instance of the server-aided OPPRF $\mathcal{F}_{\mathsf{opprf}}^{(m,m)}$ where:
   - P_i acts as a sender with input $\mathcal{P}_i$,
   - P_1 acts as a cloud server with no input
   - P_n acts as a receiver with input $X'_n = F(k, X_n)$. P_n obtains the result $y_{i,j}$ on the query $x_{n,j}$.
6. For every $j \in [m]$, P_n computes $w_j = \bigoplus_{i=2}^{n-1} y_{i,j} \oplus S(K_n, x_{n,j})$. Then, P_n sets W to be $\{w_1, \ldots, w_m\}$.
7. P_1 and P_n invoke the server-aided $\mathcal{F}_{\mathsf{PSI-CA}}$ functionality with P_2 as a server, where
   - P_n acts as a sender with input W
   - P_2 acts as a cloud server with no input
   - P_1 acts as a receiver with input $V = F(s, X)$, and obtains $|W \cap V|$.

Correctness. We consider three following cases based on whether x is in the intersection of all sets $X_{i\in[n]}$:

- Case 1: Suppose $x \in \bigcap X_{i\in[n]}$. In other words, $\forall i \in [n], \exists x_{i,j_i} \in X_i$, such that $x_{i,j_i} = x$. Thus, we have (i) all PRF values $F(k, x_{i,j_i}) = F(k, x)$ are equal, (ii) XORing all zero shares $S(K_i, x_{j_i}), i \in [2, n]$ is equal to zero. When querying the OPPRF points $\mathcal{P}_{i\in[2,n]}$ using the common PRF value $F(k, x)$, the party P_n obtains y_{i,j_i}. Based on the correctness of OPPRF, we have $y_{i,j_i} = S(K_i, x)$ for $i \in [3, n-1]$ and $y_{2,j_2} = S(K_i, x) \oplus PRF(s, x)$. Therefore, the value $w = \big(\bigoplus_{i=2}^{n-1} y_{i,j_i}\big) \oplus S(K_n, x)$ is equal to $PRF(s, x)$ as $\bigoplus_{i=2}^{n} S(K_i, x) = 0$. Step (7) allows P_1 to count x to output the intersection set by checking whether $w \in F(s, X)$.
- Case 2: Suppose x is in X_1 and is not an element in some sets $X_{i\in[2,n]}$. Clearly, $w \not\in F(s, X_1)$ with the high probability.
- Case 3: Suppose x is an element in some sets $X_{i\in[2,n]}$, but not in X_1. The value w_j might equal to $F(s, x_2)$ for $x_2 \in X$ or random. However, $x_2 \not\in X_1$, thus $w \not\in F(s, X_1)$ with the high probability.

Theorem 3. *Protocol 3 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the $(\mathcal{F}_{\mathsf{opprf}}, \mathcal{F}_{\mathsf{Coin}})$-hybrid model, in the presence of an adversary who may actively corrupt any subset from $\{P_3, \ldots, P_n\}$ or passively corrupt one of P_1,P_2 or P_n (i.e. P_1, P_2 and P_n are non-colluding).*

The security proof of the above theorem is given in Appendix A. Note that we can remove the assumption of non-colluding corrupt $P1$ or P_2 in our server-less multi-party PSI-CA if we use a standard two-party PSI-CA and/or standard two-party OPPRF to implement Step (5) and Step (7), respectively.

5 Applications

We demonstrate that our PSI-CA can be used for several privacy-preserving applications by implementing two running example applications which are built on our two-party and multi-party PSI-CA protocols, respectively.

5.1 Secure Dot Product Construction

Given a secure protocol for computing the cardinality of the intersection of the parties' sets, the protocol for dot product is simple. Let x_i be an m-element binary vector of party P_i, and let $A_i = \mathbf{idx}(x_i)$. It is easy to see that the dot product of the x_i's is exactly the cardinality of the intersection of the A_i's, that is, $\sum_{j=1}^{m} \prod_{i=1}^{n} x_i[j] = |\bigcap_{i=1}^{n} A_i|$. Thus, to securely compute the dot product, we can use the PSI-CA functionality described in the previous section. Note that even though the input size is $O(m)$, the communication complexity of the protocol is only $O(t)$, which makes it extremely efficient when $t = o(m)$, where t is the upper bound on the Hamming weight of the vectors.

One subtle issue is that in the PSI-CA protocols the parties know the number of elements in each other's set, which leaks more information than required. Here, we assume that there is a known upper bound, t, on the Hamming weight of the vectors X_i's, and require that the parties' input to the PSI-CA contains exactly t items. That is, if the Hamming weight of X_i is $t' < t$ then P_i adds random "dummy" items to its input to the PSI-CA protocol. Formally, for a given upper bound t, P_i inputs A_i to the PSI-CA protocol where $A_i \leftarrow \mathbf{idx}'(X_i, t)$ and $\mathbf{idx}'(X, t)$ is defined as follows: let t' be the Hamming weight of X, set $A = \mathbf{idx}(X)$, pick $t - t'$ random values $D = \{d_1, \ldots, d_{t-t'}\}$ from the domain $\mathcal{D} = \{m + 1, \ldots, 2^{\lambda+\log(t)} + m\}$ and output $A = A \cup D$. The choice of the domain $\mathcal{D}$ allows the collision probability of dummy items to be negligible and equals to $2^{-\lambda}$)

The formal description is given in Protocol 6 in Appendix B. Note that it is possible to compute dot product DotProd with or without the help of a cloud server $\mathcal{C}$, so both variants are presented. The protocol's correctness, complexity and security follow directly from the underlying PSI-CA protocol presented in Section 4 with different corruption structures.

Theorem 4. *Protocol 6 securely computes Functionality 5 ($\mathcal{F}_{\mathsf{DotProduct}}$) in the ($\mathcal{F}_{\mathsf{PSI-CA}}$)-hybrid model. In particular, if π is a protocol that securely computes $\mathcal{F}_{\mathsf{PSI-CA}}$ in the presence of an adversary $\mathcal{A}$ then, when instantiated with π, Protocol 6 is secure in the presence of adversary $\mathcal{A}$ as well.*

5.2 Heatmap Computation

As formally stated in [BBH+20], the heatmap can be considered as a two-party computation between HHS and a mobile network operator (MNO). HHS has a list of individuals who have reported positive for the disease. MNO knows an approximated location data of their subscribers as the subscriber connects to a certain cell tower when traveling (unless the user does not have a phone or disconnects to their network provider). Mathematically, HHS generates a binary vector $x \in \mathbb{Z}_2^N$

which indicates whether the user $i \in [1, N]$ amongst N subscribed individuals has tested positive ($x[i] = 1$) or not ($x[i] = 0$). For each cell tower $j \in [1, m]$, the MNO initializes a vector y_j of n elements, where $y_j[i]$ corresponds to the i-th subscriber (say that HHS and MNO agree on the subscribers' identifier and on their positions in the vectors). If the i-th subscriber connects to a cell tower j within some period of time, then $y_j[i] = 1$, and $y_j[i] = 0$ otherwise. To learn how many individuals visit a certain area (e.g. the area covered by the j-th cell tower, HHS and MNO run a secure dot product protocol to obtain $x \cdot y_j$.

The solution proposed in [BBH+20] relies on HE to implement the secure dot product for the heatmap problem. Even with the HE optimizations, [BBH+20] requires $O(N)$ independent secure multiplications to compute $x \cdot y_j$ for each cell tower. Therefore, their protocol costs $O(mN)$ HE multiplications to compute secure vector-matrix multiplications $x \cdot Y$, where Y consists of m columns $y_1, \ldots, y_m$. Each element of $x \cdot Y$ corresponds to how many diagnosed subscribers visited a cell town.

In this work, we observe that the proportion of diagnosed individuals among all N subscribed individuals is usually small (e.g. $0.01 - 0.1\%$ new positive cases per day [wor21]), thus, the vector x is sparse. In addition, the vector y_j is also sparse due to people's localized travel habits. Therefore, the heatmap computation is a perfect application for our DotProd where the input vectors are sparse. By applying DotProd, we show that the computational complexity of the dot product in the heatmap example can be reduced from $O(N)$ to $O(t)$, where t is the maximum between the upper bound on the number of new positive test cases and the upper bound on the number of individuals visiting a geographical area covered by a cell tower.

Multiple MNOs. We support a heatmap computation between one HHS, P_0, and multiple MNOs, $P_1, \ldots, P_n$. For a cell tower $j \in [1, m]$, the MNO P_k ($k \in [n]$) has the vector y_j^k of N elements. $y_j^k[i]$ indicates whether a subscriber i connects to a cell tower j of the MNO P_k (we assume that the j-th cell tower of all MNOs covers the same geographical area, this should be adjusted in practice). The sum of the dot products $\sum_{k=1}^{n}(x \cdot y_j^k)$ indicates how many individuals, across different MNOs, visit a certain area. In our multi-party heatmap, if P_0 invokes DotProd with each MNO P_k where P_0's input is x and P_k's input is y_j^k, P_0 learns extra information – each term of the sum $\sum_{k=1}^{n}(x \cdot y_j^k)$. To address the issue, we modify the underlying shuffled-opprf protocol of DotProd. At the high-level idea, $\mathcal{C}$ computes PRF values of all MNOs $P_{k \in [n]}$, permutes them before returning to the P_0. The formal description of our multi-party heatmap computation is presented in Protocol 4.

In real-world scenarios, HHS prefers to minimize bandwidth cost and computation workload on their side. Our protocol makes this happen by making use of the untrusted server. For the heatmap computation, HHS only needs to compute nmt and $2nmt$ symmetric-key operations in the two-party and multi-party settings, respectively. In terms of communication cost, HHS sends and receives $3nmt$ elements. Finally, our heatmap protocol requires only 1-round communication.

5.3 Association Rule Learning

Association rules learning (ARL) aims to discover regularities/rules between variables in transaction data. In this work, we use our DotProd protocol to mitigate information leakage in ARL when training the model on a vertical partitioning of the private database between multiple parties. We study the ARL definition in [AIS93] and adapt it to the privacy-preserving context (see Definition 1). We consider only a vertically-partitioned database since if the data is horizontally-partitioned, each party can locally compute ARL.

PROTOCOL 4. (*Server-aided Heatmap Construction*)

Parameters:

- Parameters k, N, t.
- A HHS and n MNO $P_1, \dots, P_n$, and a cloud server $\mathcal{C}$
- A PRF $F : \{0,1\}^\kappa \times \{0,1\}^\star \to \{0,1\}^\kappa$

Inputs:

- A HHS P_0 has input a binary vector x of length N
- Each MNO $P_{k \in [n]}$ has input a binary matrix Y_k of size $N \times m$
- Cloud server $\mathcal{C}$ has no input.

Protocol $n = 1$: For each $j \in [m]$, the HHS and the MNO P_1 invoke $\mathsf{DotProd}$ where P_0 input is x and $P1$ input is y_j^1. The HHS outputs $x \cdot y_j^1$

Protocol $n > 1$:

1. HHS computes a set $A \leftarrow \mathbf{idx}(x)$ and pads A with dummy items to the upper-bound set size t.
2. $P_0, P_1, \dots, P_n$ agree on a random PRF key s using $\mathcal{F}_{\mathsf{Coin}}$.
3. For each $j \in [m]$:
   (a) $P_{k \in [n]}$ computes a set $B_k \leftarrow \mathbf{idx}(y_j^k)$, and pads B_k with dummy items to the upper-bound set size t.
   (b) Each MNO $P_{k \in [n]}$, the HHS, and the cloud server $\mathcal{C}$ jointly invoke a modified shuffled-OPRF:
   - P_k chooses two PRF keys $s_{k,1}, s_{k,2} \leftarrow \{0,1\}^\kappa$
   - P_k sends $s_{k,1}$ to HHS and sends $s_{k,1}$ to $\mathcal{C}$
   - HHS computes and sends $A'_k = F(s_{k,1}, A)$ to $\mathcal{C}$.
   - $\mathcal{C}$ computes $A''_k = F(s_{k,2}, A'_k)$.

   $\mathcal{C}$ sends a *permutation* of $A'' \leftarrow \{A''_1, \dots, A''_n\}$ to HHS
   (c) Each $P_{k \in [n]}$ sends $B'''_k = F\big(s, F(s_{k,2}, F(s_{k,1}, B_k))\big)$ to $\mathcal{C}$ who sends a *permutation* of $B^\star \leftarrow \{B'''_1, \dots, B'''_n\}$ to HHS.
   (d) HHS computes $A^\star = F(s, A'')$ and outputs $|A^\star \cap B^\star|$.

Definition 1. *In the privacy-preserving ARL (PPARL) problem, there are n parties $P_1, \dots, P_n$, each holding a private vertically-partitioned database of transactions $T_1, \dots, T_n$, respectively. Let $T = T_1 || \dots || T_n$ be a jointed vertically database of n parties. Let $I = \{i_1, i_2, ..., i_m\}$ be a public set of binary attributes, called items. Each transaction (row) $t \in T$ is represented as a binary vector, with $t[k] = 1$ if the transaction contains item $i_k \in I$, and $t[k] = 0$ otherwise. We say that the transaction t satisfies $\mathbf{idx}(t)$. Denote an association rule by $\Rightarrow$. Let $X, Y \subseteq [m]$, we consider the following association rules:*

1. *The rule $X \Rightarrow Y$ holds in T with support factor of $0 \le s \le 1$ iff at least $s\%$ of transactions in T satisfy $X \cup Y$*
2. *The rule $X \Rightarrow Y$ holds in T with confidence factor of $0 \le c \le 1$ iff at least $c\%$ of transactions in T that satisfy X also satisfy Y.*
3. *The rule $X \Rightarrow Y$ is global if every transaction in T has at least one item in $X \cup Y$.*

*The goal of PPARL is to allow all parties $P_1, \dots, P_n$ to find all **global** rules having high support and confidence on their jointed database T while maintaining the privacy of each individual database.*

Generally speaking, the support factor indicates how frequently the itemset appears in the dataset. The support of X with respect to T is defined as the proportion of transactions in the

PROTOCOL 5. (*Privacy-Preserving ARL*)

PARAMETERS:

- A ARL threshold τ, α attributes, empty lists $L_n, \ldots, L_\alpha$.
- n parties: $P_1, \ldots, P_n$.
- An DotProd functionality described in Functionality 5.
- An `apriori-gen` algorithm described in Figure 1.

INPUTS: $P_{i\in[n]}$ has input a vertically-partitioned database $T_{i\in[n]}$.

PROTOCOL:

1. $P_{i\in[n]}$ locally computes a list L_1^i of frequent itemsets that has only 1 attribute.
2. Parties $P_{i\in[n]}$ invoke a DotProd execution with each attribute input $j_i \in L_1^i$, and add j_i into a published list L_n if the output of the DotProd is great than τ (e.g. a sum of element-wise products of multiple sparse binary vectors $T[j_i]$ as $\sum_{v=1}^{m} \prod_{i=1}^{n} T[j_i][v] > \tau$)
3. For $k = n+1$ to α, if L_k is empty, the parties do the following:
   (a) $P_{i\in[n]}$ locally computes $C_k = \texttt{apriori-gen}(L_{k-1})$.
   (b) For each candidate $c \in C_k$, let $J = \{j_1, \ldots, j_m\}$ be a set of attributes in c.
      - Assume that each $P_{i\in[n]}$ have h_i attributes $J_i = \{j_{i_1}, \ldots, j_{i_{h_i}}\}$. P_i locally computes an element-wise product of multiple binary vectors $T[j_{i_v}]$ as $X_i \leftarrow \prod_{v=1}^{h_i} T[j_{i_v}]$
      - Parties invoke a DotProd execution:
        – P_i inputs X_i.
        – P_1 obtains the output s, and adds c to L_{k+1} if $s > \tau$

dataset which contains the itemset X. That is, $\mathrm{supp}(X) = \frac{|\{X \subseteq T\}|}{|T|}$.

The confidence factor indicates how often the rule $X \Rightarrow Y$ is true. The confidence value of a rule, $X \Rightarrow Y$, in a set of transactions T, is the proportion of the transactions that contain X which also contain Y. $\mathrm{conf}(X \Rightarrow Y) = \frac{\mathrm{supp}(X \cup Y)}{\mathrm{supp}(X)}$. Thus confidence can be interpreted as an estimate of the conditional probability.

Given the definitions of support and confidence factors, the method for finding an association rule [AIS93] can be decomposed into two subproblems.

1. Find the frequent itemset: The frequent itemset is defined as the itemset that appears in the transaction set T at least τ times, where τ is predefined minimum support (also called a threshold).
2. Use the frequent itemsets to generate the association rules: For every large itemset X, find all non-empty subsets A of X. For every such subset A, output a rule of the form $A \Rightarrow (X \setminus A)$ if the ratio of $\mathrm{supp}(X)$ to $\mathrm{supp}(A)$ is at least τ.

In PPARL, the second subproblem can be publicly solved since the frequent itemsets are a part of the ARL result. According to [VC02], one can reduce the first subproblem of PPARL to securely computing the dot products of the binary vectors with minor leakage information. For simplicity, consider the candidate itemset has only two attributes. Let x and y represent columns in the database. i.e., $x[i] = 1$ iff row i has value 1 for attribute X (similar for y and Y). Each party P_1 and P_2 holds a vertically-partitioned database of the transaction x and y respectively. The dot product of two m-element vectors x and y as $x \cdot y = \sum_{i=0}^{m} x[i]y[i]$ is the support count which indicates how many times the itemset XY appears in the joint transaction set. The dot product computation requires the joint database from both parties, thus, it should be computed in a privacy-preserving manner. Given $s \leftarrow x \cdot y$, the parties can check whether the obtained support count is greater or equal to the threshold τ. If yes, the candidate itemset is a frequent itemset. In the ideal world, if $s < \tau$, the exact value of s is not revealed to the parties. Thus, the information is considered as leakage information in our PPARL scheme as well as previous work [VC02, DC14]. Note that [VC02, DC14] reveal more information than ours - they leak indexes that $x[i] = y[i] = 1$ (i.e. intersection items).

In this work, we consider *global* rules where every vertically-partitioned transaction database $T_{i\in[n]}$ has at least one item in the frequent itemset. Protocol 5 presents our PPARL construction which closely follows the Apriori algorithm [AIS93, VC02]. The first two steps aim to find a list of itemsets that (1) appear in the transaction set T at least τ times; and (2) every party has at least one attribute in the itemset. We denote the obtained list to be L_n. Given L_n, the party locally computes a list of candidates C_{n+1} for itemsets of size $n+1$ using the `apriori-gen` algorithm [AIS93]. At the high-level idea, the function `apriori-gen` is done by generating a superset of possible candidate itemsets and pruning this set. We present the `apriori-gen` algorithm in Figure 1, and refer the reader to [AIS93] for more detail. Note that `apriori-gen` is computed on the public list L_n, thus it leaks no additional information. The parties jointly execute Step (3) to compute $L_{t>n}$ until it is empty. For whom are not familiar with ARL, we provide a detailed explanation of the algorithm in Appendix E.

6 Implementation and Performance

We evaluate the performance of our PSI-CA (or DotProd) protocols and estimate the performance of our applications: heatmap computation and ARL. Protocols are evaluated under different network settings, number of parties, and input set sizes to demonstrate the performance and scalability.

Choice of Parameters. We run experiments on a single machine 2× 36-core Intel Xeon 2.30GHz CPU and 256GB of RAM and simulated network using the Linux *tc* command. We consider two network settings: the LAN setting has 0.02ms round-trip latency and 10 Gbps network bandwidth; the WAN setting has 96ms round-trip latency and 200 Mbps network bandwidth. In our implementation, each party uses a separate thread to communicate with other parties. The computational security parameter $\kappa = 128$ and the statistical security parameter $\sigma = 40$. The number of parties is in a range of $\{2, 4, 8, 16\}$. The set size m of PSI-CA or the upper-bound Hamming weight t of DotProd is in $\{2^{12}, 2^{16}, 2^{20}, 2^{24}\}$.

Choice of PRF, OPPRF, and OKVS. We instantiate the PRF F using AES-NI. We use OKVS and OPPRF as a black box in the implementation. Our implementation uses the table-based OPPRF[2] code from [KMP+17]. While there are different OKVS constructions [GPR+21], we choose the most efficient Encode and Decode of 3-cuckoo PaXoS data structure. The number of bins in the cuckoo table is $1.3m$ with 3 hash functions.

PSI-CA and DotProd protocols. Recall that the steps of PSI-CA and DotProd protocols are similar, except for a small cost overhead in Step (1) of DotProd where each party locally computes a function **idx**(). In the DotProd protocol, we assume that there is a known upper bound, t, on the Hamming weight of the party's input vector X. To implement DotProd using PSI-CA, we require that the parties' input to the PSI-CA contains exactly t items. Thus, we only report the detailed computational and communication performance results of our PSI-CA protocols for the set size m. It indicates that the DotProd protocols are evaluated with the upper bound $t = m$.

[2] Note that the table-based OPPRF which is secure in the semi-honest setting but is about 3× slower than the state-of-the-art malicious PaXoS-based or vOLE-based OPPRF [RS21, GPR+21].

	DH-PSICA [IKN^{+}20]				ROOM [SGRP19]				Catalic [DPT20]				Ours			
m_1	2^{16}		2^{20}		2^{16}		2^{20}		2^{16}		2^{20}		2^{16}		2^{20}	
m_2	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}	2^{12}	2^{16}	2^{16}	2^{20}
LAN	8.31	10.21	112.51	191.87	14.3	144.17	-	-	6.41	8.92	85.1	166.12	0.1	0.13	1.01	1.5
WAN	11.26	11.5	150.14	248.32	-	-	-	-	-	-	-	-	1.54	2.37	4.85	8.24
Comm.	2.82	4.78	46.14	77.59	863	13788	878	13837	6.29	6.29	100.66	100.66	1.18	3.15	18.87	50.33
System	server-less								two non-colluding servers				one non-colluding server			
Req.	semi-honest parties								semi-honest parties/servers				malicious parties, semi-honest server			

Table 1: Run time (in second), communication cost (in MB), and system requirement of the two-party PSI-CA (or DotProd) protocols: DH-based PSICA [IKN^{+}20, Mea95], Catalic [DPT20], ROOM as a building block in DotProd [SGRP19], and ours (a simpler variant of the [KMRS14] PSI protocol) for the sender set size m_1 and receiver set size m_2. Cells with $-$ denote trials that are not supported by the protocol.

6.1 Performance of Two-party Protocols

PSI-CA Protocol. Table 5 (in Appendix) presents the performance of our two-party PSI-CA protocol in both LAN and WAN settings. We consider both balanced and unbalanced set sizes as our heatmap computation is built on the asymmetric two-party PSI-CA. In our protocol, the parties do not need to involve in the entire protocol's computation. The sender can send $F((k_1, k_2), X)$ and k_1 to the receiver, send k_2 to the server at the same time and complete its computation. Similarly, the server does not need to be online during the whole process. Instead, the server can start his/her computation when receiving the sender's key PRF k_2 and the set of receiver's queries. Therefore, we report the performance of each participant separately. We find that our protocol scales well in the experiments as it contains only AES calls. For instance, the total run time of our PSI-CA with the input set size $m_1 = m_2 = 2^{20}$ is only 1.5 seconds.

Comparison with Prior Work. Both DH-based and delegated PSI-CA [DPT20] protocols are secure against a semi-honest adversary, but the latter requires two non-colluding servers. Note that one can use the protocol proposed in [MPR^{+}20] to implement PSI-CA, however, the protocol is much expensive compared to DH-based PSI-CA. The PSI-CA implementation of [TSS^{+}20, DIL^{+}20] is not available[3], thus we omit to compare theirs with ours. In addition, we compare our protocol with ROOM-based protocol [SGRP19]. The two-party DotProd of [SGRP19] consists of two expensive steps: ROOM and a generic dense matrix multiplication. In Table 1, we only report the performance of ROOM in settings where [SGRP19] performs best.

We use DH-based PSI code implemented by [RT21] with the fastest Curve25519 implementation from `libsodium`. For a fair comparison, we run the implementation of delegated PSI-CA [DPT20] and DH-based PSI on the same benchmark machine and network settings. Note that [DPT20] only provides the implementation of their protocol building blocks, thus, there are no performance results on the WAN setting. The times[4] for ROOM are taken from [SGRP19, Figure 17] and [LPR^{+}20, Table 2], initially provided for a database $50,000$ and a number of queries $5,000$ and $50,000$. Table 1 presents the performance of each PSI-CA protocol. When comparing the protocols, we find that the running time of our protocol is $10 - 100\times$ faster than that of the prior works. In addition, our protocol requires $2 - 5\times$ less bandwidth cost compared to them. The results show the benefit of using our protocols in a reasonable server-aided model.

[3] [TSS^{+}20] requires a non-colluding server that is similar to ours, but their protocol heavily replies on DH based PSI. [DIL^{+}20] requires two non-colluding senders, each holds an identical input set.

[4] Unknown benchmark machine

	m	$n=4$			$n=8$			$n=16$		
		P_1	$P_{(2:n)}$	Server	P_1	$P_{(2:n)}$	Server	P_1	$P_{(2:n)}$	Server
Runtime LAN (second)	2^{12}	0.19	0.18	0.19	0.35	0.28	0.35	0.54	0.39	0.54
	2^{16}	1.38	1.02	1.37	2.31	1.22	2.3	4.56	1.97	4.55
	2^{20}	19.65	15.6	19.39	33.86	16.8	33.61	71.17	32.36	70.89
Runtime WAN (second)	2^{12}	1.89	1.15	1.84	2.47	1.16	2.08	3.04	1.25	2.66
	2^{16}	6.9	3.18	6.01	14.07	4.1	13.28	26.09	6.01	25.3
	2^{20}	106.08	23.98	97.49	197.35	39.43	196.87	409.13	71.26	408.65
Comm. Cost (MB)	2^{12}	0.13	1.64	5.05	0.13	1.64	11.61	0.13	1.64	24.73
	2^{16}	2	25.93	79.79	2	25.93	183.51	2	25.93	390.95
	2^{20}	32	467.66	1434.98	32	467.66	3305.62	32	467.66	7046.9

Table 2: Run time (in second) and communication cost (in MB) of our server-aided multiparty PSI-CA protocols for n parties on sets of size m.

	m	$n=4$				$n=8$				$n=16$			
		P_1	P_2	$P_{(3:n-1)}$	P_n	P_1	P_2	$P_{(3:n-1)}$	P_n	P_1	P_2	$P_{(3:n-1)}$	P_n
Runtime LAN (second)	2^{12}	0.07	0.07	0.06	0.07	0.07	0.07	0.06	0.07	0.08	0.08	0.06	0.08
	2^{16}	0.40	0.37	0.21	0.33	0.40	0.38	0.22	0.34	0.430	0.39	0.23	0.36
	2^{20}	6.01	5.69	3.99	6.38	6.32	5.77	4.26	7.02	6.75	6.43	4.60	7.16
Runtime WAN (second)	2^{12}	1.52	1.33	0.06	0.75	1.73	1.54	0.06	0.96	1.74	1.55	0.06	0.97
	2^{16}	4.21	3.61	0.98	2.37	4.59	4.00	1.17	2.76	6.09	5.49	1.46	4.26
	2^{20}	21.60	21.24	11.32	20.20	33.75	33.34	19.86	32.34	59.63	59.27	36.72	58.26
Comm. Cost (MB)	2^{12}	0.52	0.28	0.16	0.71	1.02	0.28	0.16	1.85	2.02	0.29	0.16	4.12
	2^{16}	8.27	4.54	2.54	11.35	16.27	4.54	2.54	29.51	32.27	4.54	2.54	65.83
	2^{20}	132.32	72.64	40.64	181.60	260.32	72.64	40.64	472.16	516.32	72.64	40.64	1053.28

Table 3: Run time (in second) and communication cost (in MB) of our "server-less" multiparty PSI-CA protocols for n parties on sets of size m.

	PSI [CDG+21] (server-less, semi-honest)			PSI-CA Protocol 2 (server-aided, malicious)			PSI-CA Protocol 3 (server-less, semi-honest)		
m_1	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
LAN	0.23	1.6	23.8	0.19	1.38	19.65	0.07	0.4	6.38
WAN	1.9	7	108.2	1.89	6.9	106.08	1.52	4.21	21.6
Comm.	3.2	49.4	790.2	3.41	53.86	967.32	0.84	13.35	213.6

Table 4: Run time (in second) and communication cost (in MB) of [CDG+21] and our protocols for 4 parties and no collusion. Each party has a set size m. The numbers of [CDG+21] are for PSI itself (not, PSI-CA).

Performance of Heatmap Computation. In the two-party setting, executing the heatmap computation essentially involves multiple DotProd or PSI-CA executions. Similar to [BBH+20], we want to evaluate our protocol for smaller nation-states such as New York City or Singapore which has a population around $N = 2^{23}$. Concretely, we consider a case in which the MNO has a matrix Y of size $N \times m$ and the HHS has a vector x of N, where $N = 2^{23}$ and $m = 2^{15}$. The parties need to perform m DotProd instances as $x \cdot y_{j \in [m]}$, where y_j is the j^{th} column of Y. Recall the x and y_j are binary vectors that indicate whether an individual tested positive to COVID-19, and whether this individual visited a place nearby the network town y_j, respectively. Among $N = 2^{23}$, we assume that there are $t_2 = 2^{12}$ new positive cases per day [wor21], and each patient visits 4 places per day on average. We run $m = 2^{15}$ instances of our two-party PSI-CA protocol with the MNO's set size

$t_1 = 2^{14}$ and the HHS's set size $t_2 = 2^{12}$, and find that our protocol costs about 10 minutes using a *single* thread. On the other hand, [BBH+20] reports about 90 minutes but using 96 threads and stronger benchmark machine [5]. Therefore, we estimate that our protocol is at least 50× faster than [BBH+20]. It dues to the fact that our protocol is based on symmetric-key operations while [BBH+20] heavily relies on public-key operations. In addition, [BBH+20] requires that the participants agree on database indices (i.e. data alignment before running heatmap computation). Using PSI-CA, we can remove this requirement. The party's input can be a set of patient/visitor ids (instead of the vector/matrix).

Performance of ARL Based on the DotProd performance, we *estimate* the performance of our ARL. In two-party setting, each party $P_{i\in[2]}$ locally computes a list L_1^i of frequent itemsets that has only one attribute. The parties sequentially invoke DotProd to compute lists L_k of frequent itemsets that has exactly k attributes where L_{k+1} is empty (say L_{m+1} is empty). Assume that each attribute/vector in $L_{j\in[2,m]}$ has a Hamming weight t_j. Also, assume that each C_j has $|C_j|$ candidates. The performance of our ARL is $\sum_{i=2}^{m} |C_j|[\Pi^{(t_j,2)}_{\mathsf{DotProduct}}]$, where $[\Pi^{(t_j,2)}_{\mathsf{DotProduct}}]$ is the cost of two-party DotProd with Hamming weight t_j. According to Table 5, we estimate that our ARL would take under hours to compute ARL of the database with million records.

6.2 Performance of Multi-party Protocols

PSI-CA Protocol. The running times and communication overhead of our server-aided multi-party PSI-CA are shown in Table 2. The protocol is asymmetric with respect to the server, the receiver P_1 and other parties $P_{i\in[2,n]}$, thus, we report the performance results of these parties separately. In our protocol, the workload of the receiver is light as it only requires to call m AES instances. The majority of the receiver's running time is to wait for other parties to finish their work. For example, P_1 takes 33.86 seconds to compute PSI-CA (or DotProd) with $n = 8$ and $m = 2^{20}$ (or $t = 2^{20}$) in the LAN setting. Also, the server plays the role of the receiver in most OPPRFs, his communication cost is highest amongst other participants. For $n = 8$ and $m = 2^{20}$ (or $t = 2^{20}$), the protocol PSI-CA (or DotProd) requires 3305 MB on the server's side.

Table 3 presents the performance of our "server-less" multiparty PSI-CA protocol in both LAN and WAN settings. Similar to the server-aided protocol, we separately report the performance results of P_1, P_2, P_n and other parties $P_{i\in[3,n-1]}$. Unlike server-aided protocol, this protocol only relies on OKVS (i.e. makes use of symmetric-key operations only). We find that our protocol scales to large input sets (e.g. $m = 2^{20}$) with a large number of participants (e.g. $n = 16$). For $n = 16$ and $m = 2^{20}$ (or $t = 2^{20}$), our protocol requires only 6 seconds with the total communication cost 1GB.

Comparison with Prior Work. As far as we know, [CDG+21]'s implementation is not publicly available. Thus, we take their reported run times from [CDG+21, Table 2-5]. For the most direct comparison, we used the same configured machine (2x 36-core Intel Xeon 2.30GHz 256GB of RAM) and network settings to evaluate their and our protocols. We compare our "server-less" protocol with [CDG+21] for the case of $n = 4$, one dishonestly colluding (no collusion), each with $m \in \{2^{12}, 2^{16}, 2^{20}\}$. We show an improvement of $1.6-5\times$ in the run time, and $3.5-4\times$ in the bandwidth cost. We report the performance numbers in Table 4.

From Table 3, our server-less protocol with $n = 16$ requires only 6.38s in the LAN setting and $m = 2^{20}$. From Table 4, the [CDG+21] with $n = 4$ requires 23.8s in the same setting. Our

[5] an c5.24xlarge AWS EC2 instance (96 vCPU @ 3.6 GHz, 192 GiB RAM)

protocol with $n = 16$ is already 3.74× faster than [CDG+21] with $n = 4$, thus, we do not present the comparison of the two protocols for larger n.

Performance of Heatmap Computation. The complexity of our heatmap protocol is linear in the number of MNOs. Using the suitable parameters of the two-party heatmap where each MNO has a matrix of size $2^{23} \times 2^{15}$, and HHS has a vector of size 2^{23}, we *estimate* that our protocol takes about one hour if there are 6 MNOs involved in the protocol execution. Note that our protocol does not reveal additional information other than the output – how many patients visit a certain area. In contrast, [BBH+20] only works in the two-party setting. In real-world scenarios, there are many MNOs. If using only their protocol where the HHS executes vector-matrix multiplication with each MNO and then computes the "global" heatmap, this solution leaks extra information – the individual result of each vector-matrix multiplication.

Performance of ARL Similar to the two-party ARL, the performance of our multi-party ARL is $\sum_{i=n}^{m} |C_j|[\Pi^{(t_j,n)}_{\mathsf{DotProduct}}]$, where $[\Pi^{(t_j,n)}_{\mathsf{DotProduct}}]$ is the cost of n-party DotProd with Hamming weight t_j. Here, we assume that each attribute/vector in $L_{j\in[n,m]}$ has a Hamming weight t_j. According to the performance of our multi-party DotProd (or multi-party PSI-CA) shown in Table 2&3, we estimate that our ARL would take under a day to compute ARL of the database with million records.

References

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[BBH+20] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. Privately connecting mobility to infectious diseases via applied cryptography. Cryptology ePrint Archive, Report 2020/522, 2020. https://ia.cr/2020/522.

[BBM+21] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The apple psi system, 2021. [Online; accessed 18-Sept-2021].

[BBV+20] Alex Berke, Michiel Bakker, Praneeth Vepakomma, Ramesh Raskar, Kent Larson, and AlexSandy' Pentland. Assessing disease exposure risk with location histories and protecting privacy: A cryptographic approach in response to a global pandemic. *arXiv preprint arXiv:2003.14412*, 2020.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO*, volume 576 of *LNCS*, pages 420–432. Springer, 1991.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. STOC, 1990.

[CDG$^+$21] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty psi and extensions to circuit/quorum psi. CCS, 2021.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 34–63. Springer, Heidelberg, August 2020.

[DC14] Changyu Dong and Liqun Chen. A fast secure dot product protocol with application to privacy preserving association rule mining. In *PAKDD*, 2014.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, November 2013.

[DGD18] Outsourcing scalar products and matrix products on privacy-protected unencrypted data stored in untrusted clouds. *Information Sciences*, 2018.

[DIL$^+$20] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Function secret sharing for psi-ca: With applications to private contact tracing. Cryptology ePrint Archive, Report 2020/1599, 2020.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Heidelberg, August 2007.

[DPT20] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 870–899. Springer, Heidelberg, December 2020.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS 2015*. The Internet Society, February 2015.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC*, 2005.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption how to play mental poker keeping secret all partial information. STOC '82, 1982.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GPR$^+$21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 395–425, Virtual Event, August 2021. Springer, Heidelberg.

[HLL$^+$16] Chunqiang Hu, Ruinian Li, Wei Li, Jiguo Yu, Zhi Tian, and Rongfang Bie. Efficient privacy-preserving schemes for dot-product computation in mobile computing. PAMCO '16, 2016.

[IKN$^+$20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *EuroS&P*, pages 370–389. IEEE, 2020.

[KMP$^+$17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1257–1272. ACM Press, October / November 2017.

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, volume 8437 of *Lecture Notes in Computer Science*, pages 195–215. Springer, 2014.

[KOS03] Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 578–595. Springer, Heidelberg, May 2003.

[KRS$^+$19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *USENIX*, August 14-16, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, August 2005.

[Lin01] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 171–189. Springer, Heidelberg, August 2001.

[LPR$^+$20] Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. Private join and compute from pir with default. Cryptology ePrint Archive, Report 2020/1011, 2020. https://ia.cr/2020/1011.

[Mea95] Catherine Meadows. Formal verification of cryptographic protocols: A survey (invited lecture). In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT'94*, volume 917 of *LNCS*, pages 135–150. Springer, Heidelberg, November / December 1995.

[MPR^{+}20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2020.

[MRR20] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast database joins and PSI for secret shared data. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1271–1287. ACM Press, November 2020.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 94–124, Virtual Event, August 2021. Springer, Heidelberg.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 901–930. Springer, Heidelberg, October 2021.

[RT21] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. CCS, 2021. https://ia.cr/2021/1159.

[Rud12] Cynthia Rudin. Mit lecture notes: Machine learning and statistics, 2012. https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec01.pdf.

[SBS19] Babak Siabi, Mehdi Berenjkoub, and Willy Susilo. Optimally efficient secure scalar product with applications in cloud computing. *IEEE Access*, 2019.

[SGRP19] Phillipp Schoppmann, Adrià Gascón, Mariana Raykova, and Benny Pinkas. Make some ROOM for the zeros: Data sparsity in secure distributed machine learning. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1335–1350. ACM Press, November 2019.

[TSS^{+}20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *arXiv*, 2020.

[VC02] Jaideep Vaidya and Chris Clifton. Privacy preserving association rule mining in vertically partitioned data. KDD, 2002.

[VC05] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13:593–622, 10 2005.

[wor21] Covid-19 coronavirus pandemic, 2021. =https://www.worldometers.info/coronavirus/.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[ZWH$^+$16] Youwen Zhu, Zhikuan Wang, Bilal Hassan, Yue Zhang, Jian Wang, and Cheng Qian. Fast secure scalar product protocol with (almost) optimal efficiency. In Song Guo, Xiaofei Liao, Fangming Liu, and Yanmin Zhu, editors, *Collaborative Computing: Networking, Applications, and Worksharing*, pages 234–242, Cham, 2016. Springer International Publishing.

[ZWY$^+$17] Jun Zhang, Xin Wang, Siu-Ming Yiu, Zoe Jiang, and Jin Li. Secure dot product of outsourced encrypted vectors and its application to svm. 2017.

		Comm.			LAN			WAN		
m_2	m_1	Receiver	Sender	Server	Receiver	Sender	Server	Receiver	Sender	Server
2^8	2^8	0.012	0.004	0.008	0.002	0.001	0.002	0.481	0.001	0.289
	2^{10}	0.025	0.016	0.008	0.002	0.002	0.002	0.482	0.002	0.29
	2^{12}	0.074	0.066	0.008	0.008	0.006	0.006	0.679	0.005	0.486
2^{12}	2^{12}	0.197	0.066	0.131	0.01	0.005	0.008	1.066	0.005	0.681
	2^{14}	0.393	0.262	0.131	0.029	0.019	0.02	1.274	0.02	0.888
	2^{16}	1.18	1.049	0.131	0.099	0.065	0.065	1.537	0.066	1.144
2^{16}	2^{16}	3.146	1.049	2.097	0.132	0.058	0.102	2.374	0.065	1.579
	2^{18}	6.291	4.194	2.097	0.315	0.209	0.203	2.924	1.42	2.097
	2^{20}	18.874	16.777	2.097	1.007	0.583	0.553	4.853	3.732	3.597
2^{20}	2^{20}	50.332	16.777	33.554	1.501	0.964	1.206	8.235	5.745	7.681
	2^{22}	100.663	67.109	33.554	4.814	2.637	4.247	19.535	15.373	18.772
	2^{24}	301.99	268.435	33.554	19.123	9.625	17.305	66.244	54.594	64.089

Table 5: Running time (in second) and communication cost (in MB) of our two-party PSI-CA protocols for the sender set size m_1 and receiver set size m_2.

A Security Proof

Theorem 2. *Protocol 2 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{opprf}}, \mathcal{F}_{\mathsf{ZS}}, \mathcal{F}_{\mathsf{Coin}}$)-hybrid model, in the presence of an adversary who may actively corrupt any strict subset of $\{P_1, P_3, \ldots, P_n\}$ or $\{P_2, P_3, \ldots, P_n\}$ or passively corrupt the cloud server $\mathcal{C}$.*

Proof. We separate the proof to n cases, depending on what parties the adversary corrupts. We will consider a maximal corruption (of $n-1$ parties), from which security against a non-maximal corruption can be easily derived. Thus, the following proofs address cases where the adversary corrupts $\{P_i\}_{[n]\setminus\{i\}}$. Finally, we consider a passive corruption of the cloud server.

Corrupted $\{P_i\}_{i\in[n]\setminus\{1\}}$. The simulator Sim runs $\{P_i\}_{[n]\setminus\{1\}}$ internally and plays the role of the $\mathcal{F}_{\mathsf{opprf}}$ and $\mathcal{F}_{\mathsf{Coin}}$ functionalities. First Sim generates the keys $\{K_i\}_{i\in\{2,\ldots,n\}}$ for the sharing function S and hands them to the corresponding parties. Sim also generates the random values $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ and k (from steps 2 and 3 of the protocol) and hands them to the corresponding parties. Now, Sim observes the input $\mathcal{P}_i$ from P_i to the $\mathcal{F}_{\mathsf{opprf}}$ functionality, from which it can extract the corrupted parties' inputs as follows: Sim adds to the input sets of the parties all items a for which $\mathcal{P}_i$ contains the key-value point $(F(k,a), b_i)$ where $\bigoplus_{i=2}^n b_i = \gamma_j \bigoplus_{i=2}^n S(K_i, a)$ for some $j \in [m]$. Then, Sim pads the corrupted parties' sets with distinct items, so that their size is exactly m. The simulator Sim sends the generated input sets, X_i's as the input of P_i, to the ideal $\mathcal{F}_{\mathsf{PSI-CA}}$ functionality, from which only P_1 obtains the output. Since all incoming messages to parties $\{P_i\}_{i\in[2,n]}$ are from ideal functionalities, their views generated by Sim are distributed identically as in the ideal world. In addition, P_1's output in the real and ideal worlds is identically distributed, since the input sets generated by Sim contain all and only items that will end up in the intersection between all parties.

Corrupted $\{P_i\}_{i\in[n]\setminus\{2\}}$. Similar to the above case, Sim runs the corrupted parties internally, playing the role of $\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}_{\mathsf{opprf}}$. Sim generates the keys $\{K_i\}$ of the sharing function S and hand them to the parties, as well as the set of values $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ that are handed to P_1

and P_2. The simulator Sim extracts the input sets of $P_3, \ldots, P_n$ as follows. Sim adds to the input sets of the parties all items a for which $\mathcal{P}_i$ contains the key-value point $(F(k,a), b_i)$ where $\bigoplus_{i=3}^{n} b_i = \bigoplus_{i=3}^{n} S(K_i, a)$. Then, Sim pads the corrupted parties' sets with distinct items, so that their size is exactly m. To extract P_1's input set, Sim observes the set X_1' sent from P_1 to $\mathcal{C}$. Then, it computes $X_1 = F^{-1}(k, X_1')$. The simulator Sim sends the generated input sets, X_i's as the input of P_i, to the ideal $\mathcal{F}_{\mathsf{PSI-CA}}$ functionality, from which only P_1 obtains the output. Since all incoming messages to parties $P_1, P_3, \ldots, P_n$ are from ideal functionalities, their views generated by Sim are distributed identically as in the ideal world. Finally, given the output size, c, Sim randomly picks c items from V and sends them to P_1. P_1's view is indistinguishable from its view in the ideal world since in both cases it receives c random items from V. In addition, the output is of exactly the same size as in the real world, as the adversary could not 'artificially' increase the intersection size. Specifically, the adversary manages to incorrectly add an item to the intersection, say the item a, if (1) a was not input by P_1, and (2) computing the XOR of the OPRF results on a in the invocations with $P_2, \ldots, P_n$ results with exactly $\gamma_j \oplus S(K_2, a) \bigoplus_{i=3}^{n} S(K_i, a)$. Now, since the OPRF result, y_2', results from the invocation with P_2 is pseudorandom (and w.h.p. different than $\gamma_j \oplus S(K_2, a)$ for all $j \in [m]$) the probability that the adversary guesses $y_2' =$ is negligible.

Corrupted $\{P_i\}_{i \in [n] \setminus \{j\}}$ for $j \in \{3, n\}$. Note that P_i's part of the protocol is the same for all $i \in \{3, n\}$, therefore, we treat all these cases at once. The simulator Sim runs the corrupted parties internally, playing the role of the $\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}_{\mathsf{opprf}}$ functionalities. As before Sim generates the keys $\{K_i\}$ of the sharing function S, and the values $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ and hand them to the corresponding parties. Sim constructs the sets of the corrupted parties as follows: Sim adds to the input sets of the parties all items a for which $\mathcal{P}_i$ contains the key-value point $(F(k,a), b_i)$ where $\bigoplus_{i \in [2,n] \setminus \{j\}} b_i = \bigoplus_{i \in [2,n] \setminus \{j\}} S(K_i, a)$. Then, Sim pads the corrupted parties' sets with distinct items, so that their size is exactly m. To extract P_1's input set, Sim observes the set X_1' sent from P_1 to $\mathcal{C}$. Then, it computes $X_1 = F^{-1}(k, X_1')$. The simulator Sim sends the generated input sets, X_i's as the input of P_i, to the ideal $\mathcal{F}_{\mathsf{PSI-CA}}$ functionality, from which only P_1 obtains the output. Since incoming messages to all corrupted parties are from ideal functionalities, their views generated by Sim are distributed identically as in the ideal world. Finally, given the output size, c, Sim randomly picks c items from Γ and sends them to P_1. P_1's view is indistinguishable from its view in the ideal world since in both cases it receives c random items from Γ. The adversary manages to "artificially" add an item to the intersection with negligible probability, using exactly the same analysis as in the above case.

Corrupted $\mathcal{C}$. Here there is no input to be extracted as $\mathcal{C}$ does not have an input. Note that all the messages received by $\mathcal{C}$ in the real world are pseudorandom, so the simulator can simply simulate them.

□

Theorem 3. *Protocol 3 securely computes Functionality 4 ($\mathcal{F}_{\mathsf{PSI-CA}}$) for arbitrary n, in the ($\mathcal{F}_{\mathsf{opprf}}, \mathcal{F}_{\mathsf{Coin}}$)-hybrid model, in the presence of an adversary who may actively corrupt any subset from $\{P_3, \ldots, P_n\}$ or passively corrupt one of P_1,P_2 or P_n (i.e. P_1, P_2 and P_n are non-colluding).*

Proof. We separate the proof to multiple cases, depending on the adversary's corruption. As before, we assume maximal corruption and stress that the security in the case of a non-maximal corruption can be easily derived.

Algorithm 1 `apriori-gen`(L_t)

```
Find all pairs of itemsets in $L_t$ where the first $t-1$ items are identical.
        e.g., $t = 5$ and two pairs $\{a, b, c\}, \{a, b, d\}$
Union them (lexicographically) to get a list of candidates $C'_{t+1}$
        e.g., $\{a, b, c\}, \{a, b, d\} \to \{a, b, c, d\}$
Prune $C_{t+1} = \{c \in C'_{t+1} \mid \forall s_c \notin L_t\}$, where $s_c$ is a $t$-subsets of $c$.
Return $C_{t+1}$
```

Figure 1: A Simplest `apriori-gen` Algorithm [AIS93, Rud12]

Corrupted $\{P_i\}_{i\in[2,n-1]}$. The simulator Sim runs the corrupted parties internally, playing the role of the $\mathcal{F}_{\mathsf{Coin}}$, $\mathcal{F}_{\mathsf{ZS}}$ and $\mathcal{F}_{\mathsf{opprf}}$ functionalities. Sim generates the keys $\{K_i\}_{i\in[2,n]}$ and the PRF keys s and k, and send them to the corresponding parties. Sim extracts the input sets of the corrupted parties as follows: The corrupted parties use s and k to generate their set of points to the $\mathcal{F}_{\mathsf{opprf}}$ invocations, from which they do not receive an output. Finally, P_2 is a server in the invocation of the $\mathcal{F}_{\mathsf{PSI-CA}}$ functionality between P_1 and P_n, from which the semi-honest P_2 learns nothing (except the set size, which is already known). This completes the simulation. The parties' views are identically distributed in the ideal and real worlds.

Corrupted P_1. The simulator Sim generates the PRF key s and hands it to P_1. It obtains the intersection cardinality, c, from the ideal execution, and forwards c to P_1 as its output. It is easy to see that P_1's view in the ideal and real worlds are identically distributed.

Corrupted P_n. The simulator Sim generates the keys $\{K_i\}$ for the sharing function S and the PRF key k, and hands them to the corresponding parties. On every invocation of $\mathcal{F}_{\mathsf{opprf}}$ between P_i and P_n, the simulator Sim sends P_n a pseudorandom result. The distributions of the view of P_n in the real and ideal worlds are indistinguishable, as in both worlds all received messages are pseudorandom.

□

B Our Secure Dot Product Protocol

See Protocol 6.

C Zero Sharing Protocol

See Protocol 7.

D Server-Aided 2-Party PSI Protocol

See Protocol 8.

PROTOCOL 6. (*Secure Dot Product* - $\Pi^{(t,n)}_{\mathsf{DotProduct}}$)

Parameters:

- An upper-bound t.
- n parties: $P_1, \dots, P_n$; an untrusted server $\mathcal{C}$;
- A PSI-CA functionality $\mathcal{F}_{\mathsf{PSI-CA}}$ in Functionality 4.
- A function $\mathbf{idx}' : \mathbb{Z}_2^\star \times \{0,1\}^\star \rightarrow (\{0,1\}^\star)^\star$ in Section 5.1

Inputs:

- $P_{i\in[n]}$ has $X_i = \{x_{i,1}, \dots, x_{i,m}\}$.
- Cloud server $\mathcal{C}$ has no input.

Protocol:

1. Each party $P_{i\in[n]}$ computes $A_i \leftarrow \mathbf{idx}'(X_i, t)$.
2. All parties invoke $\mathcal{F}_{\mathsf{PSI-CA}}$ where P_i inputs A_i, $\mathcal{C}$ inputs nothing, and P_1 obtains the output $|\bigcap_{i=1}^n A_i|$.

PROTOCOL 7. (*Zero-Sharing* - Π_{ZS} *[KMP⁺17]*)

Parameters: There are n parties $P_1, \dots, P_n$ and an adversary $\mathcal{A}$. There is a PRF $F : \{0,1\}^\kappa \times \{0,1\}^\ell \rightarrow \{0,1\}^\kappa$.

Protocol:

1. Each party P_i picks a random seed $r_{i,j}$ for $j \in [i+1, n]$ and sends $r_{i,j}$ to P_j. The key K_i of party P_i is $(k_{1,i}, \dots, k_{i-1,i}, k_{i,i+1}, \dots, k_{i,n})$.
2. To obtain its share for value x, party P_i computes

$$S(K_i, x) = \left(\bigoplus_{j<i} F_{k_{j,i}}(x)\right) \oplus \left(\bigoplus_{j>i} F_{k_{i,j}}(x)\right)$$

E Example of the ARL algorithm

For simplicity, we consider two parties P_1 and P_2, each holding a vertical-partitioned database T_1 and T_2, respectively. Assume that T_1 has 3 attributes/columns $\{a_1, a_2, a_3\}$, and T_2 has 2 attributes/columns $\{b_1, b_2\}$.

One important step of the ARL algorithm is to find all "global" frequent itemsets. For example, we want to compute how many transactions that contain 2 attributes (a_1, b_1). If the number of these transactions is greater than a threshold t, we say that (a_1, b_1) is a frequent itemset.

For a better protocol explanation. We define "global" vs "local" frequent itemset. A frequent itemset is global if each party has at least one item in the frequent itemset (this aligns with the global rule mentioned in Definition 1). A frequent itemset is local if the frequent itemset contains only items belonging to one party.

If (a_1, b_1) is a "global" frequent itemset, the attribute a_1 itself should be a "local" frequent itemset. Thus, before any interaction between parties, each party P_i needs to locally compute a list L_1^i that has only 1 attribute. For example, if the attribute a_1 appears more than or equal t times in T_1, then a_1 is a local frequent itemset, and thus a_1 is added to L_1^1. In contrast, if the attribute a_2 appears less than t times in the T_1, then a_2 is not a local frequent itemset, and thus $a_2 \notin L_1^1$. Assume that from Step 1, we have $L_1^1 = \{a_1, a_3\}$, and $L_1^2 = \{b_1, b_2\}$.

Step 2 of Protocol 5 aims to find a list L_n of "global" frequent itemsets, where each itemset has n items ($n = 2$ in the two-party setting). To do so, the parties run DotProd where

PROTOCOL 8. (*Server-Aided 2-Party PSI [KMRS14]*)
PARAMETERS: There are 2 parties P_1, P_2 and a third-party server S. P_1 and P_2 have sets X_1 and X_2 as input, respectively. The server S does not have input. Let F be a PRF, and parameter $d > 0$.

PROTOCOL:

1. P_1 chooses sets D_0, D_1, D_2 and a key k_1 such that $|D_0| = |D_1| = |D_2| = d$, sends them to P_2 and set $Y_1 \leftarrow X_1 \cup D_0 \cup D_1$.
2. P_2 sets $Y_2 \leftarrow X_2 \cup D_0 \cup D_2$.
3. P_2 chooses a random key k_2 and sends it to the server S.
4. Party P_1 sends a shuffled version of $Y_1' = \{F(k_1, x)\}_{x \in Y_i}$ to S.
5. The server returns a shuffled version π of $Y_1'' = \{F(k_2, y)\}_{y \in Y_1'}$ to P_1
6. Party P_2 sends a shuffled version of $Y_2'' = \{F(k_2, F(k_1, x))\}_{x \in Y_2}$ to P_1.
7. P_1 computes $I = Y_1'' \cap Y_2''$ and sends the result to P_2
8. P_2 computes $I^{-1} = \{F^{-1}(k_1, F^{-1}(k_2, x)) | \forall x \in I\}$
9. P_2 check that I has the right form and aborts if:
   (a) Either $D_0 \not\subset I^{-1}$ or $D_2 \cap I^{-1} \neq \emptyset$
   (b) There exists $x \in X_2$ and $\alpha, \beta \in [\lambda]$ such that $x||\alpha \in I^{-1}$ and $x||\beta \notin I^{-1}$
10. If P_2 does not abort, it notifies S who sends the shuffled function π to P_1. P_1 uses π learns the values in the set I^{-1}
11. P_1 checks that I has the right form as in Step (9) and aborts if the check fails.
12. The parties output distinct items in $I^{-1} \setminus D_0$.

the party's input is each itemset in L_1^1 and L_1^2. For example, the parties check whether each of pairs $(a_1, b_1), (a_1, b_2), (a_3, b_1), (a_3, b_2)$ are "global" frequent items. Assume that the column a_1 is $(1, 1, 1, 0, 0)$ and the column b1 is $(1, 1, 1, 1, 0)$. The dot product $a_1 \cdot b_1$ is 3. E.g. a pair (a_1, b_1) appears 3 times in the database. If the threshold $t = 2$, the (a_1, b_1) is a "global" frequent itemset.

Step 3 of the protocol aims to find a list L_k of "global" frequent itemsets, where each itemset has k items (here, $k > 2$). For example, the parties want to check whether (a_1, a_3, b_1) is a "global" frequent itemset (in this case, $k = 3$). They first need to compute the dot product $a_1 \cdot a_3 \cdot b_1$. To do so, P_1 locally computes a dot product of a_1 and a_3 before running a secure DotProd with P_2 (see Step 3b). The function apriori-gen is for improving the computation – it helps to generate the set of *candidate* itemsets for L_k.