Two-Sided Malicious Security for Private Intersection-Sum with Cardinality

Peihan Miao[‡,*], Sarvar Patel[†], Mariana Raykova[†], Karn Seth[†], and Moti Yung[†]

[†] Google LLC
{sarvar, marianar, karn, moti}@google.com
[‡] Visa Research
pemiao@visa.com

Abstract

Private intersection-sum with cardinality allows two parties, where each party holds a private set and one of the parties additionally holds a private integer value associated with each element in her set, to jointly compute the cardinality of the intersection of the two sets as well as the sum of the associated integer values for all the elements in the intersection, and nothing beyond that.

We present a new construction for private intersection sum with cardinality that provides malicious security with abort and guarantees that both parties receive the output upon successful completion of the protocol. A central building block for our constructions is a primitive called *shuffled distributed oblivious PRF (DOPRF)*, which is a PRF that offers oblivious evaluation using a secret key shared between two parties, and in addition to this allows obliviously permuting the PRF outputs of several parallel oblivious evaluations. We present the first construction for shuffled DOPRF with malicious security. We further present several new sigma proof protocols for relations across Pedersen commitments, ElGamal encryptions, and Camenisch-Shoup encryptions that we use in our main construction, for which we develop new batching techniques to reduce communication.

We implement and evaluate the efficiency of our protocol and show that we can achieve communication cost that is only $4-5\times$ greater than the most efficient semi-honest protocol. When measuring monetary cost of executing the protocol in the cloud, our protocol is $25\times$ more expensive than the semi-honest protocol. Our construction also allows for different parameter regimes that enable trade-offs between communication and computation.

1 Introduction

Private Set Intersection. A private set intersection (PSI) protocol enables two parties, each with a private input set, to compute the intersection of the two sets while revealing nothing more than the intersection itself. Despite the simplicity of the functionality, PSI has found many applications in privacy-preserving location sharing [NTL+11], testing of fully sequenced human genomes [BBDC+11], collaborative botnet detection [NMH+10], data mining [ARF+10], social networks [LCYL11, NDCD+13], online gaming [BHLB11], measuring ads conversion rates [IKN+17],

[*] Part of work done while interning at Google LLC.

and so on. Due to its importance and wide applications, PSI has been extensively studied in a long sequence of works [HFH99, FNP04, DSMRY09, DCKT10, HEK12, DCW13, SFF14, PSZ14, PSSZ15, EFG+15, DD15, Lam16, KKRT16, RR17a, RR17b, FNO18, PSWW18, PRTY19].

Enhanced Functionality. While the PSI functionality models successfully the confidentiality requirements in several application scenarios, there are information-sharing settings where revealing the whole intersection is unacceptable and instead a more fine-grained privacy preserving computation is needed. In particular different aggregated computations over the intersection set model a wide range of applications with restricted privacy leakage. PSI-cardinality is one example of such an aggregated functionality that limits the two parties to learning only the *cardinality* (or size) of the intersection [HFH99, AES03, FNP04, KS05, VC05, NAA+09, DCGT12].

The private intersection-sum functionality introduced by Ion et al. [IKN+17] is another example of an aggregate functionality where one of the input sets has integer values associated with the elements in the set and the two parties compute the cardinality of the intersection as well as the aggregate of the integer values associated with the intersection set. This primitive models many applications in practice. These include settings where one party holds private statistics about a set of people and another party has information about the membership of the people in a particular group, and the two parties want to compute an aggregate of the statistics over the members of the set. A particular instantiation of this scenario was consider by Nagu et al. [NDCD+13] in the context of social networks where a user has knowledge of weights associated with each of her friends and wants to compute the total (or average) weight of the friends that she has in common with another user. In measuring ads conversion rates [IKN+17], an advertiser may know the purchase amount for every customer, and the advertiser and an ads publisher can jointly compute the total number and total purchase amount of the customers who have seen the ads from the publisher and end up buying the product.

Existing solutions for private intersection-sum [IKN+17] provide security only in the semi-honest case where each party is assumed to follow the protocol honestly. While this level of security might be sufficient in settings where the interacting parties have external incentives (e.g. legal agreements) to follow the protocol, this level of security is not sufficient for a broad set of scenarios where the adversary could deviate arbitrarily from the protocol. In the setting of malicious security we have protocols that achieve only the PSI functionality, however, constructions with competitive efficiency [FHNP16, RR17a, RR17b] have a major shortcoming that they support only one-sided output, where in many settings both parties need to obtains the output of the computation. Upgrading these protocols to achieve two-sided output in a non-trivial task. For example, as explained by Rindal et al. [RR17b], the output recipient from the one-sided protocol will need to prove that it executed the last step of the protocol honestly. We do not have tailored constructions for this task and applying generic approaches comes with a high price.

In this work we consider the problem of private intersection sum with cardinality in the malicious setting which provides protection against such adversaries. We require that either both parties receive the output of the computation or they abort. Our focus is on optimizing the communication efficiency of the protocol since as discussed in the work of Ion et al. [IKN+17] this is the most significant cost in practice.

Our Contributions. We present a new protocol for private intersection-sum with cardinality which achieves malicious security with abort, which guarantees that both parties receive the in-

tersection sum if the protocol does not abort. Our protocol provides two-sided output, which is already an improvement even if we restrict our attention only to the PSI functionality since existing malicious PSI protocols [FHNP16, RR17a, RR17b] are restricted to a single output recipient.

Our construction is the first construction for private intersection-sum with cardinality with malicious security to achieve linear communication and computation overhead in the size n of the sets. This improves significantly over the only other existing approach [HEK12] that can be used to solve this problem, which uses existing generic MPC techniques with malicious security, and as we discuss in the related work, incurs at least a factor of $\lambda \log n$ multiplicative overhead assuming a security parameter λ. As can be seen in Table 6, these generic techniques incur 250× higher communication and 65× higher monetary cost than our protocol on inputs of size 2^{20}.

Our construction can also be instantiated such that the overhead required to achieve malicious security over the semi-honest version requires sublinear communication $O(\sqrt{n})$ with computation $O(n \log n)$, which would be advantageous in setting where communication is much more expensive that computation.

Our construction adopts the general approach from the work of Ion et al. [IKN+17], which leverages an oblivious pseudorandom function (PRF) with a shared key, which can be evaluated in a distributed way to permute and map the input set values to a pseudorandom space that enables the computation of the intersection, and homomorphic encryption, which allows to pair the associated values during the PRF evaluation and then evaluate the intersection sum. In order to upgrade this general approach to malicious security we develop several new techniques, which can be of independent interest.

New Distributed OPRF. A central building block for our solution is a distributed oblivious PRF with malicious security. In order to achieve distributed oblivious evaluation with malicious security we leverage a PRF construction due to Dodis and Yampolskiy [DY05], for which we can construct proofs for honest evaluation with respect to a committed PRF key. An issue that we need to deal with is the fact that this PRF was proven secure only for polynomial domains. To circumvent this problem we introduce a weaker selective security notion for the PRF, which is is satisfied by the construction with exponential domain, and we show that this property suffices for our PSI-sum with cardinality protocol.

Verifiable Parameter Generation. We construct a distributed PRF evaluation protocol, which uses several times evaluations on committed and encrypted values. Thus, in order to achieve malicious security for this protocol we use proofs for relations among encrypted and committed values, which crucially rely on the assumption that the parameters for these schemes were generated honestly. Since we do not want to assume any trusted setup, we present protocols for verifiable generation of parameters for Pedersen commitments, Camenish-Shoup (CS) and ElGamal encryption with shared key.

Range Proofs with Slack. The final extension to the distributed OPRF is to enable a shuffle of the oblivious evaluations on multiple inputs that are executed in parallel, which hides the mapping to the original inputs and is required in order to hide what elements are in the intersection. In order to enable that we develop a proof protocol for shuffle decryption of Camenisch-Shoup encryptions. We leverage the Bayer-Groth shuffle proof [BG12], which allows to prove that two sets of cipheretexts encrypt the same set of plaintexts up to a permutation. In order to enable proving knowledge of exponents in this step, the prover needs to switch from Camenisch-Shoup encryption to ElGamal encryption, which have different domains. We introduce a proof technique for consistency of values encrypted under CS and ElGamal encryptions that uses range proofs with a slack.

Our construction leverages heavily sigma proof protocols [Dam02] in several places including the proofs for evaluation of the DOPRF, the re-encryption step for shuffling, the re-randomization for intersection-sum.

Batching for Range Proofs. We introduce new batching techniques for range proofs based on sigma protocols. While existing efficient batch proofs that do not work with the bit level representation of the values operate in a group of unknown order [Bou00,CS03], batching techniques for sigma protocols have been constructed only in the case of a known order group [GLSY04]. We show how to batch range proof over groups of unknown order while avoiding a large blowup in the slack of the range proof which is incurred if we adapt directly the batching approach for known group order to hidden order by providing sufficient space to avoid the need for modulus reduction.

Batching Proofs for CS and ElGamal Encryptions. We also use batching techniques for commitments and develop batching approaches for Camenisch-Shoup encryptions. We leverage multi-exponentiation arguments from the work of Bayer and Groth [BG12] in a new way to batch proofs for relations among ElGamal ciphertexts for which prover does not know the encryption randomness. Since we need an additively homomorphic encryption scheme that has a provable threshold decryption, we use exponential ElGamal to encrypt associated values. This means that our construction supports evaluations for which the final intersection-sum is within a polynomial domain where discrete log can be computed for decryption.

Implementation and Evaluation. We implemented our malicious secure private intersection-sum protocol and evaluated its performance on large-scale datasets. Our experiments show that, when we set parameters to minimize communication overhead, our protocol performs with communication cost approximately $4\times$ greater than the most communication-efficient semi-honest protocol based on DDH. A less aggressive choice of parameters leads to about $7\times$ expansion over the semi-honest DDH-based protocol, with a much improved computational efficiency. We also estimate the monetary cost of running our protocols using the pricing for Google Cloud and obtain that executing our PSI-Sum protocols on inputs of size 2^{20} costs 13 cents. The monetary cost is about $25\times$ more than that of the semihonest protocol, which we believe is a reasonable cost for the much stronger security guarantees. We present our experimental measurements in Section 6. Our costs give a large improvement in monetary cost over existing generic approaches for private intersection sum with cardinality. Our monetary costs are also within a factor of 2 of the most efficient protocols for Malicious PSI [RR17b], which we note only provide one-sided output and are not compatible with computing functions on the intersection.

Related Work. Before presenting the technical overview of our construction, we overview existing PSI solutions in the malicious setting [KS05, HL08, DSMRY09, CZ09, CKRS09, JL09, HN10, DCKT10, FHNP16, RR17a, RR17b] and discuss the challenges in extending the approaches from these works to the private intersection-sum problem. We restrict our discussion to constructions that provide linear communication complexity as our major goal is communication efficiency.

The work of De Cristofaro et al. [DCKT10] presents a PSI protocol, where only one party (P_2) learns the PSI output and nothing is revealed to the other party (P_1). Our goal is to obtain a protocol where both parties receive the output, and next we explain the challenges for achieving this functionality here. At a high level the protocol works as follows. First, the two parties jointly evaluate an oblivious pseudorandom function (OPRF) on every element of P_2 where P_1 holds the OPRF key k and only P_2 obtains the OPRF values. Second, P_1 computes the OPRF values on its own elements using the key k and sends to P_2. Finally, P_2 computes the intersection of the OPRF

values and the corresponding set intersection. The protocol used an OPRF defined as $F_k(x) = H_2(x||H_1(x)||H_1(x)^k)$, where $H_1(\cdot), H_2(\cdot)$ are hash functions modeled as random oracles [BR93]. In the OPRF protocol, P_2 learns $H_1(x)^k$ without revealing any information about x to P_1, and finally computes $H_2(x||H_1(x)||H_1(x)^k)$. Since we want both parties to learn the PSI output, one natural idea is to let P_2 send back its OPRF values to P_1, but P_2 has to prove that $H_2(\cdot)$ is computed correctly on desired inputs without revealing any information about x, which is a challenge. Another idea is to run the protocol twice with alternative roles, where the parties have to prove input consistency during the two executions. In other words, P_1 should prove in zero knowledge that its inputs to $F_k(\cdot)$ in the first execution are consistent with its inputs to the OPRF in the second execution, which is also challenging. More importantly, it is hard to extend this protocol to PSI-cardinality or private intersection-sum. In the last step of their OPRF protocol, P_2 computes H_2 on $x||H_1(x)||H_1(x)^k$ for each of its element x. It is crucial that P_2 knows the inputs to H_2 to compute the OPRF value. Therefore, the elements in the intersection must be known to P_2, making it hard to extend the protocol to even PSI-cardinality.

The PSI protocol of Jarecki and Liu [JL09] is also based on an OPRF protocol similarly as above, but the parties can prove consistency of their inputs to the OPRF with previously committed values. Therefore, the two parties can first commit to their inputs and then run the above protocol in both directions so that both parties learn the PSI output. However, the protocol has some limitations. First, their security proof requires the domain of the elements to be restricted to polynomial in the security parameter. Besides, the protocol requires a Common Reference String (CRS), where the CRS includes a safe RSA modulus that must be generated by a trusted third party, which is something we would like to avoid. To extend this protocol to PSI-cardinality, the receiver (P_2) of the OPRF protocol should learn the OPRF values without learning the correspondence between its elements $\{x\}_{x\in X}$ and OPRF values $\{F_k(x)\}_{x\in X}$, which requires shuffling techniques that we develop in this work. More ingredients and techniques are needed for extending the protocol to private intersection-sum as well as removing the above restrictions.

The idea in the protocol of Freedman et al. [FHNP16] to achieve malicious security is to require one party (P_1) to redo the other party's (P_2's) computation on the elements in the intersection and verify consistency. This is achieved as follows: P_1 generates a polynomial $Q(\cdot)$ of degree m, with roots set to the m elements of P_1's set, and sends the homomorphically encrypted coefficients of $Q(\cdot)$ to P_2. Then for each element x in P_2's set, P_2 replies with an encryption of $r \cdot Q(x) + s$ for random r and s. Importantly, the randomness used in this computation is taken from $H(s)$ where $H(\cdot)$ is a hash function modeled as a random oracle. If x is in the intersection, then P_1 can learn s and verify P_2's computation on x; otherwise nothing about x is revealed to P_1. This protocol crucially needs P_1 to learn the elements in the intersection, therefore extending the protocol to even PSI-cardinality seems to require innovative ideas. Moreover, the techniques of hashing into bins are leveraged in the protocol for achieving linear computational complexity. Computing PSI for each bin is sufficient for the PSI problem, however revealing intersection-cardinality or intersection-sum for each bin compromises security in the problem of PSI-cardinality or private intersection-sum.

Another option for constructing a private intersection-sum protocol with malicious security is to apply directly malicious two-party computation protocols to our functionality. Such protocols use the circuit representation of the evaluated functionality. The most efficient way to compute the intersection of two sets of size $O(n)$ uses oblivious sorting which reduces the number of needed comparisons from $O(n^2)$ to $O(n \log n)$. In our construction, in contrast, we aim for linear dependence on the number of inputs. Further, circuit solutions are bound to incur additional security

factor multiplicative overhead since they need to operate with the bit-level representation of the set values. In the case of garbled circuit-based solutions this is inherent in the constructions, and in the case of solutions using arithmetic circuits the need for using the bit representation comes from the fact that we will be computing comparisons over these values and the most efficient way to do this is using the binary representation of the values. The recent circuit-based PSI protocols [PSSZ15, CO18, PSTY19, FNO19] only provide security in the semi-honest setting and it is nontrivial to extend them to the malicious setting due to their use of specific primitives such as Cuckoo hashing. Moreover, their protocols require super-linear communication. The work of Pinkas et al. [PSTY19] presents a semi-honest circuit-based PSI construction that achieves linear communication, however, this construction achieves only linear number of comparison in the circuit by using oblivious programmable PRF techniques [KMP+17] and Cuckoo hashing [PR04]. Generalizing these techniques to the malicious setting presents many challenges. Our construction presents an approach to obtain oblivious PRF evaluation in the malicious setting.

2 Technical Overview

In this section we give a technical overview of our malicious secure private intersection-sum protocol. Our starting point is the semi-honest private intersection-sum protocol [IKN+17]. We identify the technical challenges to obtain malicious security from the semi-honest version and then present our approach to addressing them.

Semi-Honest Private Intersection-Sum. The semi-honest protocol of Ion et al. [IKN+17] leverages a cryptographic primitive called distributed oblivious pseudorandom function (DOPRF), which enables the following functionality. The key k of a DOPRF is shared between two parties, where each party can generate independently their share. The DOPRF has an oblivious evaluation functionality, which is a 2-party computation protocol, which the two parties jointly evaluate the PRF F, under key k, on an input x, held by one of the parties who receives the PRF output $F_k(x)$ and nothing more is revealed to either party.

The DOPRF functionality suffices to construct a PSI protocol as follows. First, the two parties generate independently key shares of the DOPRF key. Then, they use the oblivious evaluation protocol to evaluate the DOPRF on each of P_1's input elements x_i, from which P_2 learns $F_k(x_i)$ and then sends it back to P_1. Similarly, they evaluate the DOPRF on P_2's input elements y_j to obtain $F_k(y_j)$. Computing the intersection of the resulting two sets of PRF values enables both parties to compute the PSI since each party has the mapping from the intersecting PRF values to their corresponding input elements.

The above PSI protocol can be extended to obtain PSI-cardinality and private intersection-sum protocols. To achieve PSI-cardinality, it suffices to construct a shuffled DOPRF protocol, which allows n parallel executions of the oblivious PRF evaluation where the PRF value that one of the parties receives are randomly shuffled with a permutation selected by the other party. The party who receives the PRF values can still compute the intersection between the two sets of PRF values but no longer has a mapping between the intersecting PRF values and the inputs to which they correspond. Thus, the only thing this party can learn is the cardinality of the intersection. We can extend this idea to further obtain private intersection-sum in the setting where one party (say P_1) has associated integer values with its set elements. In this setting, the two parties first run the shuffled DOPRF for P_2's input set. For P_1's input set, the two parties evaluate the DOPRF on

each of P_1's inputs x_i. In addition, P_1 attaches an encryption of x_i's associated integer v_i under re-randomizable additive-homomorphic encryption for which P_1 holds the secret key. This allows P_2 to learn an $(F_k(x_i), \mathsf{Enc}_{pk}(v_i))$-pair for each x_i, so it can compute the set intersection from the two sets of PRF values and then homomorphically add up the corresponding ciphertexts. The resulting ciphertext is then re-randomized and sent back to P_1, who has the decryption key to recover the intersection-sum.

The primitives and protocols described above are only secure against semi-honest adversaries. In order to construct a private intersection-sum protocol that provides malicious security, we design malicious counterparts of these tools.

Malicious DOPRF. The semi-honest intersection-sum protocol of Ion et al. [IKN+17] uses the following Diffie-Hellman-based PRF construction, which is defined as $F_k(x) = H(x)^k$, where the hash function $H(\cdot)$ is modeled as a random oracle [BR93]. It can be instantiated as a DOPRF by sharing the PRF key as $k = k_1 k_2$. Specifically, the two parties can independently generate key shares k_1 and k_2. To evaluate the DOPRF on P_1's input x, P_1 sends $y = H(x)^{k_1}$ to P_2 and then P_2 can compute the PRF output $z = y^{k_2}$. When we switch to the malicious setting, a malicious P_1 may send $\widetilde{y} = H(x)^{r \cdot k_1}$ to P_2 for an arbitrary r and obtain $\widetilde{z} = H(x)^{r \cdot k_1 k_2}$, from which P_2 can learn the PRF output by raising $\widetilde{z}$ to the power r^{-1}. In order to upgrade this DOPRF protocol to the malicious setting especially with simulation-based security, P_1 needs to prove that the hash function $H(\cdot)$ was properly applied or equivalently prove the knowledge of a preimage for a hash value, which is a challenge.

In view of the above difficulties associated with the use of the DH-based DOPRF in the malicious setting, we choose to use a different PRF as a starting point for a new DOPRF construction, for which correct evaluation can be proven. We use the function $F_k(x) = g^{\frac{1}{k+x}}$, which is defined on a group $\langle g \rangle$ of prime order. This function was originally introduced as a weak signature in the work of Boneh-Boyen [BB04], and subsequently was proven to be a pseudorandom function under the decisional q-Diffie Hellman Inversion (q-DHI) assumption [MSK02] by Dodis-Yampolskiy [DY05]. We combine ideas from Belenkiy et al. [BCC+09] and Jarecki-Liu [JL09] to construct a distributed oblivious evaluation protocol for this PRF and prove its security in the malicious setting.

We start with a description of a distributed evaluation protocol for the above PRF that provides semi-honest security. We refer to the two parties as a sender and a receiver, where the party holding the input x is called the sender and the party obtaining the PRF output is called the receiver. For the distributed key generation the two parties randomly pick secret key shares k_s and k_r such that the PRF key k is set as $k = k_s + k_r$. The starting point for our distributed evaluation protocol is the following idea. The receiver encrypts its key share k_r using an additive-homomorphic public-key encryption scheme for which it holds the secret key, and sends the encryption $\mathsf{Enc}_{pk}(k_r)$ to the sender. The sender then homomorphically computes $\mathsf{Enc}_{pk}(k_s + k_r + x)$ and sends it back to the receiver. The receiver can decrypt the ciphertext to obtain $k_s + k_r + x$ and compute the PRF output $g^{\frac{1}{k_s + k_r + x}}$.

In the above protocol the receiver learns information beyond the PRF output, which consists of the value $k_s + k_r + x$. To remove this leakage we introduce a random multiplicative mask a on the sender's side. That is, the encrypted value that the receiver obtains is $a(k_s + k_r + x)$. We remove this mask during exponentiation by having the sender also send g^a to the receiver and letting the receiver compute $(g^a)^{\frac{1}{a(k_s + k_r + x)}}$. In fact, this randomization does not suffice for a simulation proof. Since $a(k_s + k_r + x)$ is homomorphically computed by the sender who cannot take modulo operation

under the homomorphic encryption, the value $a(k_s + k_r + x)$ learned by the receiver may still leak information about k_s+k_r+x. That is why we further modify the randomization to $a(k_s+k_r+x)+bq$ where b is random and q is the order of the group $\langle g \rangle$. This randomization guarantees that the value obtained by the receiver is simulatable and at the same time correct since the order of the group is q.

To obtain malicious security in the above protocol, the sender needs to prove the correctness of the homomorphic encryption and the consistency of a in the new ciphertext and in g^a. To achieve this we use Camenisch-Shoup encryption [CS03], for which we can use sigma protocols to provide zero-knowledge proofs for these operations.

Exponential Domain for Dodis-Yampolskiy PRF. The work of Dodis and Yampolsky [DY05] proved adaptive security for the PRF construction that we discussed above but only in the setting of polynomial size domains. However, this is not true for the inputs used in many real-world applications. Therefore, we revisit the security proof for this construction and show that for exponential size domains the PRF satisfies a weaker notion of selective security, where the inputs to the PRF are chosen by the adversary in advance in the security game, under the q-DHI assumption. Furthermore, this level of security for the PRF is sufficient for the security of our private intersection-sum protocol for the following reason. At a high level, we make the two parties first commit to their own input along with a zero-knowledge proof of knowledge and then jointly decide the PRF parameters. In the simulation-based proof, the simulator can first extract the adversary's input and then reduce to the security game of the PRF, where selective security suffices for our purpose.

Malicious PSI. As we discussed for the semi-honest setting, a secure DOPRF protocol suffices for a PSI protocol. In the malicious setting, to construct a malicious PSI protocol from the above malicious DOPRF protocol, the receiver should send back the PRF values to the sender and prove correctness of its computation $(g^a)^{\frac{1}{a(k_s+k_r+x)+bq}}$ with respect to g^a and the ciphertext $\mathsf{Enc}_{pk}(a(k_s + k_r + x) + bq)$, in a zero-knowledge fashion. This can also be achieved by sigma protocols.

Malicious Shuffled DOPRF. To extend the malicious PSI protocol to malicious PSI-cardinality, we need to additionally enable the shuffled DOPRF functionality that provides all the PRF outputs to the sender in a randomly shuffled (permuted) order determined by the receiver. While our malicious DOPRF protocol provides the receiver with the leverage to shuffle the PRF outputs before sending back to the sender, we still need a way to prove the correctness of the shuffle.

While it is possible to try to leverage generic zero-knowledge protocols to prove directly the correctness of the shuffled outputs, we choose to use a shuffle-and-decrypt protocol by Bayer-Groth [BG12], which can efficiently prove in zero-knowledge that given a set of ciphertexts and a set of plaintexts, the plaintexts correspond to the decryption of some permutation of the ciphertexts. To incorporate this shuffle proof in our protocol, the receiver no longer just sends the PRF outputs back to the sender after the DOPRF evaluation, but rather sends encryptions of these outputs together with proofs that each of them encrypts the correctly computed value $(g^a)^{\frac{1}{a(k_s+k_r+x)+bq}}$. In addition to this the receiver sends the PRF outputs in the clear in a shuffled order together with a Bayer-Groth shuffle proof that they are consistent with the decryption of the above ciphertexts in some permuted order.

In the above construction which we design in order to leverage an efficient shuffle proof, let $\beta := a(k_s + k_r + x) + bq$. The prover needs to switch from Camenisch-Shoup encryption to ElGaml

encryption because β was encrypted in Camenisch-Shoup encryption while the value to encrypt in this step is $\sigma = (g^a)^{\beta^{-1}}$ and what the prover needs to prove knowledge about is β_i^{-1} instead of σ. Encrypting σ using ElGamal in the group $\langle g \rangle$ enables proof of knowledge in the exponent. However, the prover needs to provide a proof that the Camenish-Shoup ciphertext, which has plaintext domain $\mathbb{Z}_N$, and the ElGamal cipheretext, which has plaintext domain $\mathbb{Z}_q$ where $q \ll N$, encrypt consistent values β and β^{-1}. To achieve this we observe that it suffices to prove the consistency of the two encrypted values in their respective domains (i.e., $x \mod N = x' \mod q$) and in addition to this prove that $x' < q$. For the later since $q \ll N$, it suffices to use range proofs that have slack for sigma protocols, which can only guarantee that $x' < q \cdot r$. This completes a malicious DOPRF protocol with randomly shuffled PRF outputs.

From Shuffled DOPRF to Intersection-Sum. The shuffled DOPRF protocol suffices to obtain PSI-cardinality in the semi-honest setting by running two shuffled DOPRF with the same key, where in one protocol P_1 holds the input and acts as the sender while in the other protocol their roles are reversed. In the malicious setting when the two protocols are executed in parallel, we have to additionally make sure the two parties are using consistent DOPRF key shares. Each party will first commit to their DOPRF key shares and then prove consistency of their key shares used in the two protocols, which can be done using sigma protocols.

To further achieve private intersection-sum, similar to the semi-honest setting, we encrypt the integer values associated with one of the sets using additive homomorphic encryption. The secret key for this encryption is now shared between the two parties, which will be important for preserving the secrecy guarantees of the shuffle proof. The sender appends these encryptions to the corresponding inputs in the malicious shuffled DOPRF evaluation. Now the receiver that applies the shuffle in this protocol additionally needs to re-randomize the encryptions of the associated values and provides a proof that the shuffle applied to these encryptions is the same as the shuffle on the PRF values. This can be achieved in the Bayer-Groth shuffle proof because in their protocol the prover commits to the permutation and we can use the same commitment through the two shuffle proofs. Different from the semi-honest setting, now both parties can compute the intersection of the two sets of PRF values and homomorphically add up the corresponding re-randomized ciphertexts. To jointly decrypt the resulting ciphertext, each party partially decrypts the ciphertext using their own key share and sends to the other party. They also have to prove the correctness of their partial decryption, again by sigma protocols.

Batching Protocol Components. In our construction outlined above we use sigma style protocols to provide proofs for the correctness of DOPRF evaluation, re-encryption for shuffling, and re-randomization for intersection-sum. In order to optimize the communication efficiency of such protocols, we utilize various techniques to batch components of the protocol. At a high level there are three types of batching we use: batching Pedersen commitments, batching Camenisch-Shoup encryptions, and batching sigma protocols.

These batching techniques are described in Section 5. Further care needs to be taken to ensure the compatibility between different batching techniques. We describe the detailed composition of these techniques in Appendix C.

We believe that these batching techniques may be of independent interest. For example, our batched sigma protocols include tighter bounds on proofs of ranges than known techniques, and our batched Camenisch-Shoup encryption enables batched proofs of decryption, which brings asymp-

totic efficiency gains.

Organization. We introduce our notations, security assumptions, important definitions and cryptographic schemes in Section 3 and present our private intersection-sum protocol in Section 4. We prove the selective security of the PRF used in our protocol in Appendix A and prove the malicious security of our protocol in Appendix B. Our batching techniques are described in Section 5 and the concrete sigma protocols are presented in Appendix C.

3 Preliminaries

3.1 Notation

We use λ to denote the security parameter. Let $\mathbb{Z}_n$ be the set $\{0, 1, 2, \ldots, n-1\}$. $\mathbb{Z}_n^*$ is defined as $\mathbb{Z}_n^* := \{x \in \mathbb{Z}_n | \gcd(x, n) = 1\}$. We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We use $\mathsf{ord}(\mathbb{G})$ to denote the order of a group $\mathbb{G}$. By $\mathsf{negl}(\lambda)$ we denote a negligible function, i.e., a function f such that $f(\lambda) < 1/p(\lambda)$ holds for any polynomial $p(\cdot)$ and sufficiently large λ.

3.2 Computational Assumptions

Decisional q-Diffie-Hellman Inversion (q-DHI) Assumption [MSK02]. The computational q-DHI problem in a group $\mathbb{G}$ with generator g and order p is to compute $g^{1/\alpha}$ given the tuple $(g, g^{\alpha}, \ldots, g^{\alpha^q})$ for random α in $\mathbb{Z}_p^*$. We define the hardness of the *decisional* version of this problem for any fixed constant q as follows. Let gGen be an algorithm which on input a security parameter 1^λ picks a modulus p and a generator g of a multiplicative group $\mathbb{G}$ of order p. We say that the *Decisional q-DHI Assumption* holds on group (family) $\mathbb{G}$ if for every efficient algorithm $\mathcal{A}$,

$$\Big|\mathsf{Pr}\left[\mathcal{A}(g, g^{\alpha}, \ldots, g^{\alpha^q}, g^{1/\alpha}) = 1 \middle| (g,p) \leftarrow \mathsf{gGen}(1^\lambda); \alpha \leftarrow \mathbb{Z}_p^*\right]$$
$$- \mathsf{Pr}\left[\mathcal{A}(g, g^{\alpha}, \ldots, g^{\alpha^q}, h) = 1 \middle| (g,p) \leftarrow \mathsf{gGen}(1^\lambda); \alpha \leftarrow \mathbb{Z}_p^*; h \leftarrow \mathbb{G}\right]\Big| \leq \mathsf{negl}(\lambda).$$

Strong RSA Assumption [BP97, FO97]. The strong RSA assumption states that given an RSA modulus N of unknown factorization and a random element $g \in \mathbb{Z}_N^*$, it is computationally hard to find any pair of $h \in \mathbb{Z}_N^*$ and $e > 1$ such that $h^e = g \mod N$.

3.3 Cryptographic Tools

Pedersen Commitment. The Pedersen commitment [Ped91] is a commitment scheme based the hardness of discrete logarithm which provides perfect hiding and computational binding. The parameters for the Pedersen commitment are group generators $g, h \in \mathbb{G}$ where $\log_g h$ is hard to compute. The commitment of a value x, denoted as $\mathsf{com}_{g,h}(x)$, is computed as $c = g^x \cdot h^r$ for some random $r \xleftarrow{\$} \mathsf{ord}(\mathbb{G})$. The decommitment of c is (x, r). The commitment scheme is additively homomorphic, namely, $\mathsf{com}_{g,h}(x_1) \cdot \mathsf{com}_{g,h}(x_2) = \mathsf{com}_{g,h}(x_1 + x_2)$.

Note that the Pedersen commitment scheme is perfectly hiding as long as g and h are indeed generators or more weakly, if g is in the subgroup generated by h. This perfectly hiding property

holds even if the receiving party knows all the secret information about g, h and $\mathbb{G}$. In our protocol we need to prove that the Pedersen parameters are generated correctly (i.e., that $g \in \langle h \rangle$) by sigma protocols.

In the more general case the Pedersen commitment scheme can be used to commit to vectors. The parameters for a vector commitment are group generators $g_1, \ldots, g_n, h \in \mathbb{G}$ ($\log_{g_i} h$ is hard to compute for each i). The commitment to a vector $\vec{x} = (x_1, \ldots, x_n)$ is $c = \prod_{i=1}^{n} g_i^{x_i} \cdot h^r$ where r is selected at random $r \xleftarrow{\$} \mathsf{ord}(\mathbb{G})$. The decommitment of c consists of $x_1, \ldots, x_n$ and r. The vector commitment is component-wise additively homomorphic, namely, $\mathsf{com}(\vec{x}_1) \cdot \mathsf{com}(\vec{x}_2) = \mathsf{com}(\vec{x}_1 + \vec{x}_2)$.

Camenisch-Shoup Encryption. In our protocol we will use the Camenisch-Shoup encryption scheme [CS03] which is additively homomorphic and supports verifiable encryption and decryption. The key generation algorithm $\mathsf{CS\_Gen}$ samples a random RSA modulus $N = pq$, where $p = 2p' + 1, q = 2q' + 1$, and p', q' are uniformly distributed over all ℓ-bit primes for some $\ell = \ell(\lambda)$ such that p, q are also primes and $p \neq q$. Then it chooses a $2N$-th residue $g = r^{2N} \mod N^2$ for a random $r \in \mathbb{Z}_{N^2}$ and sets $y = g^x \bmod N^2$ for a random $x \in \mathbb{Z}_{\lfloor N/4 \rfloor}$. The public key is $\mathsf{pk} = (N, g, y)$ and the secret key is $\mathsf{sk} = x$. The encryption $\mathsf{CS\_Enc}$ of a message $m \in \mathbb{Z}_N$ is $\mathsf{ct} = \left(g^r \mod N^2, (1+N)^m \cdot y^r \mod N^2\right)$ where $r \xleftarrow{\$} \mathbb{Z}_{\lfloor N/4 \rfloor}$. The decryption $\mathsf{CS\_Dec}$ of a ciphertext $\mathsf{ct} = (u, e)$ given the secret key is $m = \frac{\left(\frac{e}{u^x} - 1\right) \bmod N^2}{N}$. This encryption scheme slightly differs from [CS03] in that we drop the last component from the ciphertext since they further achieve non-malleability which is not needed in our setting.

In our protocol we have to prove that N is a product of two large safe primes (a safe prime p is a prime number of the form $2p' + 1$ where p' is also a prime), which is needed for the soundness of sigma protocols we use related to g and y. In particular, we make use of the fact that there exists a large subgroup of order $p'q'$ modulo N^2. See [GMR98, CM99] for details on how N can be proved to be a safe prime product. We also need to additionally prove that the Camenisch-Shoup public key is properly generated. In particular, the party generating the key must show that g is a random $2N$-th residue mod N^2, and furthermore that there exists some x such that $y = g^x \bmod N^2$. The former can be proved by using r equal to some fixed value (e.g. $r = 2$). Then, assuming N is a safe prime product, with high probability, r^{2N} will generate a large subgroup of order $p'q'$ mod N^2. To show $y = g^x \bmod N^2$ for a hidden x, one can use a standard sigma protocol.

ElGamal Encryption. The ElGamal encryption scheme [ElG85] is an multiplicatively homomorphic encryption scheme that is used in the shuffle proof [BG12]. For a cyclic group $\mathbb{G}$ of order q with generator g, the key generation algorithm $\mathsf{EG\_Gen}$ samples a random integer $x \xleftarrow{\$} \mathbb{Z}_q$ and computes $h = g^x$. The public key is $\mathsf{pk} = h$ and the secret key is $\mathsf{sk} = x$. The encryption $\mathsf{EG\_Enc}$ of a message $m \in \mathbb{G}$ is $\mathsf{ct} = (g^y, h^y \cdot m)$. The decryption $\mathsf{EG\_Dec}$ of a ciphertext $\mathsf{ct} = (u, e)$ given the secret key is $m = \frac{e}{u^x}$. Note that a ciphertext $\mathsf{ct} = (u, e)$ can easily be re-randomized as $\mathsf{ct}' = (u \cdot g^r, e \cdot h^r)$.

2-out-of-2 Threshold Encryption. We use the exponential ElGamal encryption scheme that is additively homomorphic as the 2-out-of-2 threshold encryption scheme for encrypting the integer values in our protocol. In the key generation procedure, each party P_b ($b = 1, 2$) can use $\mathsf{EG\_Gen}$ for the ElGamal encryption scheme to generate a public key share tpk_b and a secret key share tsk_b, where $\mathsf{tpk}_b = g_b^{\mathsf{tsk}}$. The public key for the threshold encryption scheme is $\mathsf{tpk} = \mathsf{tpk}_1 \cdot \mathsf{tpk}_2$ and the

secret key is $\mathsf{tsk} = \mathsf{tsk}_1 + \mathsf{tsk}_2$. The encryption $\mathsf{Exp\_EG\_Enc}$ of a message $m \in \mathbb{Z}_q$ is $\mathsf{ct} = (g^y, \mathsf{tpk}^y \cdot g^m)$. To decrypt a ciphertext $\mathsf{ct} = (u, e)$, one party (say P_1) half-decrypts it to be $\mathsf{ct}' = \left(u, e' = \frac{e}{u^{\mathsf{tpk}_1}}\right)$ and the other party (P_2) half-decrypts ct' to obtain $m = \log_g \frac{e'}{u^{\mathsf{tpk}_2}}$. In our protocol each party needs to provide proofs that their half-decryption is done correctly, which can be done by sigma protocols.

Zero-Knowledge Argument of Knowledge. We use the notation introduced in [CS97] for the various zero-knowledge argument of knowledge of discrete logarithms and arguments of the validity of statements about discrete logarithms. The following example is taken verbatim from [CS03].

$$\mathsf{ZK\text{-}AoK}\{(a, b, c) : y = g^a h^b \ \wedge \ \mathfrak{y} = \mathfrak{g}^a \mathfrak{h}^c \ \wedge \ (v < a < u)\}$$

denotes a "*zero-knowledge argument of knowledge of integers a, b, and c such that $y = g^a h^b$ and $\mathfrak{y} = \mathfrak{g}^a \mathfrak{h}^c$ hold, where $v < a < u$,*" in which $y, g, h, \mathfrak{y}, \mathfrak{g}, \mathfrak{h}$ are elements of some groups $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\mathfrak{G} = \langle \mathfrak{g} \rangle = \langle \mathfrak{h} \rangle$. The convention is that the elements listed in the round brackets denote quantities the knowledge of which is being proved (and are in general not known to the verifier), while all other parameters are known to the verifier. Using this notation, a proof-protocol can be described by just pointing out its aim while hiding all details.

We use similar notations for zero-knowledge proofs. As an example,

$$\mathsf{ZK}\{\exists x : h = g^x\}$$

denotes a zero-knowledge proof that there exists x such that $h = g^x$.

In our protocol we instantiate this form of zero-knowledge arguments of knowledge and zero-knowledge proofs by sigma protocols. We elaborate how this can be done and how batching techniques work for sigma protocols in Section 5. The concrete sigma protocols used in our construction are presented in Appendix C.

Fiat-Shamir Heuristic. All the sigma protocols presented in Appendix C are interactive and public-coin, where the messages from the verifier are all chosen uniformly at random and independently of the messages sent by the prover. We only prove they are honest-verifier zero-knowledge. By the Fiat-Shamir heuristic [FS86], these protocols can be turned into a non-interactive proof or argument where the prover computes the public-coin challenges with a cryptographic hash function instead of interacting with a verifier, which reduces rounds of communication as well as total communication cost. Furthermore, the resulting non-interactive protocol can be proved malicious secure in the random oracle model.

Shuffle Proof. Bayer-Groth [BG12] proposed a zero-knowledge argument of knowledge for the correctness of re-randomized and shuffled of homomorphic encryptions, which achieves sublinear communication complexity. More specifically, given the public key $\mathfrak{pk}$ of the homomorphic encryption, original ciphertexts $\{\mathfrak{ct}_i\}_{i \in [n]}$, a permutation π over $[n]$, re-randomized and shuffled ciphertexts $\{\mathfrak{ct}'_{\pi(i)}\}_{i \in [n]}$ where $\mathfrak{ct}'_{\pi(i)} = \mathfrak{ct}_i \cdot \mathsf{Enc}_{\mathfrak{pk}}(1; r_i)$. The following ZK-AOK

$$\mathsf{ZK\text{-}AoK}\left\{(\pi, \{r_i\}_{i \in [n]}) : \mathfrak{ct}_i \cdot \mathsf{Enc}_{\mathfrak{pk}}(1; r_i) \quad \forall i \in [n]\right\}$$

can be prove with communication complexity $O(\sqrt{n})$. In addition, two statements can be proved to use the same permutation π. The protocol is interactive with public-coins, hence it can be turned into a non-interactive malicious secure one using the Fiat-Shamir heuristic.

3.4 Security Model

We define security of a private intersection-sum protocol against malicious adversaries in the ideal/real world paradigm. The definition compares the output of a real-world execution to the output of an ideal-world execution involving a trusted third party, which we call an ideal functionality. The ideal functionality $\mathcal{F}$, defined in Figure 1, receives the two parties' inputs, computes the intersection-sum and returns the output to both parties. Loosely speaking, the protocol Π is secure if the output of the adversary in the real-world execution is computationally indistinguishable from the output of the adversary in the ideal-world execution, which means that a real-world execution of the protocol does not leak any more information than the ideal-world execution. Hence, the parties can only learn what they can infer from their inputs and the output.

Public Parameters: P_1's set size n_1 and P_2's set size n_2.

Inputs: Party P_1 inputs a set of identifiers along with associated integer values $(X, V) = \{(x_i, v_i)\}_{i\in[n_1]}$, Party P_2 inputs a set of identifiers $Y = \{y_i\}_{i\in[n_2]}$.

Output: Upon receiving the inputs from both parties, the ideal functionality $\mathcal{F}$ computes the intersection $I = X \cap Y$ and intersection-sum $S = \sum_{i:x_i\in I} v_i$ and outputs the intersection-cardinality $|I|$ and intersection-sum S first to the corrupted party, then to the honest party.

Corrupted Party: The corrupted party may deviate from its input, may abort the procedure at any time by sending abort to the ideal functionality, and may decide the time of message delivery.

Figure 1: Ideal functionality of malicious secure private intersection-sum.

Formally, we say a private intersection-sum protocol is secure against malicious adversaries if for every PPT adversary $\mathcal{A}$ in the real world, there exists a PPT adversary $\mathcal{S}$ in the ideal world such that for any input (X, V) and Y,

$$\mathsf{Real}_{\Pi,\mathcal{A}}((X, V), Y) \overset{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S}}((X, V), Y),$$

where $\mathsf{Real}_{\Pi,\mathcal{A}}((X, V), Y)$ denotes the output of $\mathcal{A}$ in the real-world execution of protocol Π, and $\mathsf{Ideal}_{\mathcal{F},\mathcal{S}}((X, V), Y)$ denotes the output of $\mathcal{S}$ in the ideal-world execution.

4 Protocol Description

Our constructions consists of two phases. The first one is an offline setup where the two parties jointly decide parameters for the cryptographic primitives, which will be used in the online computation. Note that we do not assume trusted setup for any of the primitives and provide secure two party computation protocols for those. The second phase is the online computation that is dependent on the input sets and uses the parameters from the setup. The main building block for our online phase is a shuffled distributed oblivious PRF (DOPRF) construction, which is a primitive of independent interest and other potential applications. Thus, we present the shuffled DOPRF construction separately.

Offline Setup. In our malicious secure private intersection-sum protocol, the two parties first run a (one-time) offline setup to generate the parameters for encryption and commitment schemes. The two parties first agree on a group $\mathbb{G}$ where $\max(n_1, n_2)$-DHI assumption holds. This group will

0. P_1 and P_2 agree on a group $\mathbb{G}$ of order q with a generator $\widetilde{g}$ for which the $\max(n_1, n_2)$-DHI assumption holds.

1. Each party P_b generates $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{CS\_Gen}(1^\lambda)$ where $g_b = (r_b)^{2N}$ for a random element $r_b \in \mathbb{Z}_{N^2}$, $\mathsf{pk}_b = (N_b, r_b, g_b, y_b)$ and $N_b \geq 2^{3\lambda} q^2$, $\mathsf{sk}_b = x_b$. Party P_b sends pk_b to the other party along with a ZK-proof that N_b is a product of two large safe primes and that y_b is correctly formed:
$$\mathsf{ZK}\left\{\exists x_b : y_b = (g_b)^{x_b} \bmod N_b^2\right\}.$$

2. Each party P_b generates Pedersen commitment parameters $(\mathfrak{g}_b, \mathfrak{h}_b)$ for the large subgroup of $\mathbb{Z}^*_{N_b}$ and sends $(\mathfrak{g}_b, \mathfrak{h}_b)$ to the other party together with a zero-knowledge proof that $\mathfrak{g}_b \in \langle \mathfrak{h}_b \rangle$:
$$\mathsf{ZK\text{-}AoK}\left\{\exists r_b : \mathfrak{g}_b = (\mathfrak{h}_b)^{r_b}\right\}.$$

3. Each party P_b generates $(\mathsf{tpk}_b, \mathsf{tsk}_b) \leftarrow \mathsf{EG\_Gen}(1^\lambda)$ for the 2-out-of-2 threshold encryption scheme on the group $\mathbb{G}$ with generator $\widetilde{g}$ and sends tpk_b to the other party along with a ZK-AOK of tsk_b:
$$\mathsf{ZK\text{-}AoK}\{\mathsf{tsk}_b : \mathsf{tpk}_b = (\widetilde{g})^{\mathsf{tsk}_b}\}.$$
Both parties compute the public key $\mathsf{tpk} = \mathsf{tpk}_1 \cdot \mathsf{tpk}_2$.

4. Each party P_b generates an ElGamal key pair $(\mathfrak{pk}_b, \mathfrak{sk}_b)$ for the group $\mathbb{G}$ with generator $\widetilde{g}$ and sends $\mathfrak{pk}_b$ to the other party with a proof:

Figure 2: One-time offline setup of the malicious secure private intersection-sum protocol.

be the group where they compute DOPRF on. Each party generates parameters for Camenisch-Shoup encryption, ElGamal encryption and Pedersen commitments, and sends the public parts to the other party with corresponding proofs for correct generation (which is done as discussed in Section 3.3 and Appendix C). The two parties generate parameters for the 2-out-of-2 threshold ElGamal encryption, which can be done by each party generating locally ElGamal parameters and setting the shared secret key to be the sum of the two local secret keys, and computing the corresponding public key. The detailed protocol is described in Figure 2.

Online Phase. After the one-time offline setup, for each private intersection-sum instance, the two parties run an online protocol described in Figure 3. The inputs for the two parties are as follows: P_1 has an input set of elements $X = \{x_i\}_{i\in[n_1]}$ with associated integer values $V = \{v_i\}_{i\in[n_1]}$, while P_2 has only a set of elements $Y = \{y_i\}_{i\in[n_2]}$. The output of the protocol is that either both parties abort, or both parties obtain the intersection sum $\sum_{i:x_i\in Y} v_i$.

At a high level this protocol uses the shuffled DOPRF to enable both parties to obtain shuffled PRF evaluations for the values in X and Y, where the PRF values from X are paired with ElGamal encryptions of the corresponding integer values from V, which are encrypted under the 2-out-of-2 threshold ElGamal. Afterwards, the two parties compute independently the ElGamal encryption of the intersection sum since they can compute the intersection on the PRF values and then sum the encryptions of the integer values. At that point, the two ciphertexts held by the parties should be identical. Now each party verifiably half-decrypts the ciphertexts it has obtained and sends the

1. Each party P_b samples a random PRF key share $k_b \xleftarrow{\$} [q]$.

2. P_1 computes $\mathsf{C}_{x_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(x_i)$ for all $i \in [n_1]$, sends C_{x_i} with ZK-AOK to P_2:

$$\mathsf{ZK\text{-}AoK}\left\{(x_i, r_i) : \mathsf{C}_{x_i} = (\mathfrak{g}_2)^{x_i} \cdot (\mathfrak{h}_2)^{r_i}\right\}.$$

   P_2 computes $\mathsf{C}_{y_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(y_i)$ for all $i \in [n_2]$, sends C_{y_i} with ZK-AOK to P_1:

$$\mathsf{ZK\text{-}AoK}\left\{(y_i, s_i) : \mathsf{C}_{y_i} = (\mathfrak{g}_1)^{y_i} \cdot (\mathfrak{h}_1)^{s_i}\right\}.$$

3. P_1 and P_2 jointly decide on a random generator g for the group $\mathbb{G}$.

4. P_1 and P_2 run two shuffled DOPRF protocols described in Figure 4 in parallel, one with P_1 holding the input and the other with P_2 holding the input:

   - **Shuffled DOPRF 1**: P_1 and P_2 perform the shuffled DOPRF protocol on P_1's input $X = \{x_i\}_{i\in[n_1]}$. The output PRF values are denoted as $\{\sigma_{\pi(i)}\}_{i\in[n_1]}$. In parallel to this protocol, they do the following:
     - **Round 2**: P_1 computes $\mathsf{ct}_{v_i} \leftarrow \mathsf{Exp\_EG\_Enc}_{\mathsf{tpk}}(v_i)$ for each $i \in [n_1]$ and sends $\{\mathsf{ct}_{v_i}\}_{i\in[n_1]}$ to P_2.
     - **Round 3**: P_2 re-randomizes $\{\mathsf{ct}_{v_i}\}_{i\in[n_1]}$ to obtain $\{\mathsf{ct}'_{v_i}\}_{i\in[n_1]}$, and then uses the permutation π (same as in the shuffled DOPRF protocol) to shuffle the re-randomized ciphertexts to obtain $\left\{\mathsf{ct}'_{v_{\pi(i)}}\right\}_{i\in[n_1]}$. P_2 sends $\left\{\mathsf{ct}'_{v_{\pi(i)}}\right\}_{i\in[n_1]}$ to P_1 along with a ZK-AOK:

$$\mathsf{ZK\text{-}AoK}\left\{(\pi, \{r_i\}_{i\in[n_1]}) : \mathsf{ct}'_{v_{\pi(i)}} = \mathsf{ct}_{v_i} \cdot \mathsf{Exp\_EG\_Enc}_{\mathsf{tpk}}(1; r_i) \quad \forall i \in [n_1]\right\}$$

   - **Shuffled DOPRF 2**: P_1 and P_2 perform the shuffled DOPRF protocol, with roles reversed, on P_2's input $Y = \{y_i\}_{i\in[n_2]}$. We denote the set of PRF values as $F_k(Y)$.

5. Each party P_b determines the intersection set $I := \{t : \sigma_t \in F_k(Y)\}$ and computes $\mathsf{ct}_S = \prod_{t\in I} \mathsf{ct}'_{v_t}$. P_b verifiably half-decrypts ct_S using tsk_b and sends to the other party.

6. Each party half-decrypts the ciphertext half-decrypted by the other party, and outputs the intersection sum S.

Figure 3: Online phase of the malicious secure private intersection-sum protocol.

resulting verifiable partial decryption to the other party. Then both parties can half-decrypt the partial decryption they received to obtain the output.

Shuffled DOPRF Protocol. We describe our malicious secure shuffled DOPRF construction as a stand-alone primitive in Figure 4. For the purposes of the following discussion P_1 is the party that holds input elements $\{x_i\}_{i\in[n_1]}$, and P_1 and P_2 jointly evaluate the shuffled DOPRF on these elements. First, P_2 commits to its PRF key share k_2 and also sends a Camenisch-Shoup

Round 1. Party P_2 computes $\mathsf{ct}_{k_2} \leftarrow \mathsf{CS\_Enc}_{\mathsf{pk}_2}(k_2)$ and $\mathsf{C}_{k_2} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(k_2)$. Recall that $\mathsf{pk}_2 = (N_2, g_2, y_2)$. P_2 sends $\mathsf{ct}_{k_2} = (u, e)$ and C_{k_2} to P_1 along with a ZK-AOK

$$\mathsf{ZK\text{-}AoK}\Big\{(k_2, r_1, r_2) : u = g_2^{r_1} \ \wedge \ e = (1+N_2)^{k_2} \cdot y_2^{r_1} \ \wedge \\ \mathsf{C}_{k_2} = (\mathfrak{g}_1)^{k_2} \cdot (\mathfrak{h}_1)^{r_2} \ \wedge \ k_2 \leq q \cdot 2^{2\lambda+1}\Big\}.$$

Round 2. For each input x_i where $i \in [n_1]$, party P_1 does the following:

(a) Choose a random $a_i \xleftarrow{\$} [q]$ and $b_i \xleftarrow{\$} [q \cdot 2^\lambda]$. Compute $\mathsf{g}_i = g^{a_i}$.

(b) Compute $\alpha_i = a_i \cdot (k_1 + x_i)$ and commitments $\mathsf{C}_{a_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(a_i)$, $\mathsf{C}_{b_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(b_i)$, $\mathsf{C}_{\alpha_i} = \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(\alpha_i)$.

(c) Let $\beta_i = a_i \cdot (k_1 + k_2 + x_i) + b_i \cdot q = a_i \cdot k_2 + \alpha_i + b_i \cdot q$ and compute $\mathsf{ct}_{\beta_i} \leftarrow (\mathsf{ct}_{k_2})^{a_i} \cdot \mathsf{CS\_Enc}_{\mathsf{pk}_2}(\alpha_i) \cdot (\mathsf{CS\_Enc}_{\mathsf{pk}_2}(b_i))^q$.

(d) Send $(\mathsf{C}_{a_i}, \mathsf{C}_{b_i}, \mathsf{C}_{\alpha_i}, \mathsf{ct}_{\beta_i}, \mathsf{g}_i)$ to P_2, together with a ZK-AOK

$$\mathsf{ZK\text{-}AoK}\big\{(a_i, b_i, \alpha_i, r_1, r_2, r_3, r_4, r_5, r_6) : \\ \mathsf{C}_{a_i} = (\mathfrak{g}_2)^{a_i} \cdot (\mathfrak{h}_2)^{r_1} \ \wedge \ a_i \leq q \cdot 2^{2\lambda+1} \ \wedge \\ \mathsf{C}_{b_i} = (\mathfrak{g}_2)^{b_i} \cdot (\mathfrak{h}_2)^{r_2} \ \wedge \ b_i \leq q \cdot 2^{3\lambda+1} \ \wedge \\ \mathsf{C}_{\alpha_i} = (\mathfrak{g}_2)^{\alpha_i} \cdot (\mathfrak{h}_2)^{r_3} \ \wedge \ \mathsf{C}_{\alpha_i} = (\mathsf{C}_{k_1} \cdot \mathsf{C}_{x_i})^{a_i} \cdot (\mathfrak{h}_2)^{r_4} \ \wedge \ \alpha_i \leq q \cdot 2^{2\lambda+1} \ \wedge \\ \mathsf{ct}_{\beta_i} = (\mathsf{ct}_{k_2})^{a_i} \cdot \mathsf{CS\_Enc}_{\mathsf{pk}_2}(\alpha_i; r_5) \cdot (\mathsf{CS\_Enc}_{\mathsf{pk}_2}(b_i; r_6))^q \wedge \\ \mathsf{g}_i = g^{a_i}\big\}.$$

Note that C_{x_i} was sent by P_1 in Step 2 of the online phase, and C_{k_1} was sent by P_1 in Round 1 of the other shuffled DOPRF protocol where P_2 holds the input.

Round 3. Party P_2 does the following:

(a) Verify all the ZK-AOKs received from P_1; otherwise abort.

(b) For each $i \in [n_1]$, compute $\beta_i \leftarrow \mathsf{CS\_Dec}_{\mathsf{sk}_2}(\mathsf{ct}_{\beta_i})$ and $\mathsf{C}_{\beta_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(\beta_i)$. Compute $\gamma_i = \beta_i^{-1} \bmod q$ and $\sigma_i = \mathsf{g}_i^{\gamma_i}$. Compute $\mathfrak{ct}_{\sigma_i} \leftarrow \mathsf{EG\_Enc}_{\mathfrak{pk}_2}(\sigma_i)$.

(c) Verify that $\{\sigma_i\}_{i\in[n_1]}$ are all distinct; otherwise abort.

(d) For each $i \in [n_1]$, send $(\mathsf{C}_{\beta_i}, \mathfrak{ct}_{\sigma_i})$ to P_2 together with a ZK-AOK

$$\mathsf{ZK\text{-}AoK}\Big\{(\mathsf{sk}_2, \beta_i, r_1, r_2) : \beta_i = \mathsf{CS\_Dec}_{\mathsf{sk}_2}(\mathsf{ct}_{\beta_i}) \wedge \\ \mathsf{C}_{\beta_i} = (\mathfrak{g}_1)^{\beta_i} \cdot (\mathfrak{h}_1)^{r_1} \ \wedge \ \beta_i \leq q^2 \cdot 2^{3\lambda+1} \ \wedge \\ \mathfrak{ct}_{\sigma_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}_2}\left((\mathsf{g}_i)^{\beta_i^{-1}}; r_2\right)\Big\}.$$

(e) Re-randomize $\{\mathfrak{ct}_{\sigma_i}\}_{i\in[n_1]}$ to obtain $\{\mathfrak{ct}'_{\sigma_i}\}_{i\in[n_1]}$ with randomness 0. Pick a random permutation π over $[n_1]$ and send $\left\{\mathfrak{ct}'_{\sigma_{\pi(i)}}\right\}_{i\in[n_1]}$ to P_1 together with a ZK-AOK:

$$\mathsf{ZK\text{-}AoK}\left\{(\pi, \{r_i\}_{i\in[n_1]}) : \mathsf{ct}'_{\sigma_{\pi(i)}} = \mathsf{ct}_{\sigma_i} \cdot \mathsf{EG\_Enc}_{\mathfrak{pk}_2}(1; r_i) \quad \forall i \in [n_1]\right\}.$$

As $\left\{\mathfrak{ct}'_{\sigma_{\pi(i)}}\right\}_{i\in[n_1]}$ has randomness 0, P_1 obtains $\left\{\sigma_{\pi(i)}\right\}_{i\in[n_1]}$.

Output. P_1 verifies all the ZK-AOKs received from P_2 and aborts otherwise. Both parties obtain $\{\sigma_{\pi(i)}\}_{i\in[n_1]}$.

Figure 4: Malicious secure shuffled DOPRF protocol where P_1 holds the input.

encryption of it under its own key to P_1 together with a proof that the encrypted and the committed values are the same. P_1 can then homomorphically compute $\mathsf{CS\_Enc}_{\mathsf{pk}_2}(k_1 + k_2 + x_i)$ for each of its element x_i. To mask the value $k_1 + k_2 + x_i$, P_1 chooses randomizing values a_i and b_i and compute $\mathsf{ct}_{\beta_i} = \mathsf{CS\_Enc}_{\mathsf{pk}_2}(a_i \cdot (k_1 + k_2 + x_i) + b_i \cdot q)$ and $g_i = g^{a_i}$. P_1 also commits to the values $a_i, b_i, \alpha_i = a_i \cdot (k_1 + x_i)$ together with proofs that these commitments and encryptions use consistent values. P_2 verifies the correctness of the proofs, decrypts ct_{β_i} to obtain $\beta_i = a_i \cdot (k_1 + k_2 + x_i) + b_i \cdot q$ and computes the PRF evaluation $\sigma_i = g_i^{\beta_i^{-1}} = g^{\frac{1}{k_1 + k_2 + x_i}}$. Then, P_2 computes an ElGamal encryption $\mathsf{EG\_Enc}_{\mathfrak{pk}_2}(\sigma_i)$ and a commitment C_{β_i} and sends them to P_1 together with a proof that these values encrypt and commit to the decryption of ct_{β_i}, which P_1 verifies. In addition P_2 re-randomizes and shuffles values $\mathfrak{ct}_{\sigma_i}$ with output $\{\mathfrak{ct}'_{\sigma_{\pi(i)}}\}_{i \in [n_1]}$, and sends these values together with a proof of shuffling. Finally, $\sigma_{\pi(i)}$ are revealed to P_1 if P_2 re-randomizes the ciphertexts using randomness 0. P_1 verifies the proofs and accepts the values $\sigma_{\pi(i)}$ as its output PRF values. In this step, P_2 switches from Camenisch-Shoup encryption to ElGaml encryption because the value to encrypt is $\sigma_i = g_i^{\beta_i^{-1}}$ and what P_2 needs to prove knowledge about is β_i^{-1} instead of σ_i. Encrypting σ_i using ElGamal in the group $\mathbb{G}$ enables this proof of knowledge. If the verification of any of the proofs during the execution so the protocol fails, then the parties abort.

Additionally, during the execution of the DOPRF on the inputs of P_1, the parties run the following additional steps in parallel with the DOPRF evaluation in order to facilitate keeping the values v_i paired with the appropriate PRF evaluations. In Round 2 of the DOPRF protocol, P_1 encrypts the v_i values using the ElGamal encryption parameters where the secret key is shared between the two parties. P_1 sends these encryptions paired with the partial PRF evaluations on its elements x_i. When P_2 returns the completed DOPRF evaluations in a permuted order, it also sends the re-randomized encryptions of the values v_i permuted in the same order along with a proof that these two sets were shuffled with the same permutation.

Enabling Batching. So far we described our shuffled DOPRF construction for each element x_i and the ZK-AOKs in the protocol are all sigma protocols for single statements. To reduce communication of the protocol we utilize various batching techniques which we describe in Section 5. The concrete instantiation of our private intersection-sum protocol does not use the shuffled DOPRF in a completely non-black box way, which we discuss in the following.

In Step 2 of the online phase, P_1 will commit implicitly to its inputs by committing to the values a_i and $\alpha_i = a_i(k_1 + x_i)$ and P_2 will implicitly commit to its inputs similarly. These values can be batched and the sigma protocols for the batched commitments can also be batched. In addition each party will commit to their DOPRF key share in this step. This change does not affect our security guarantee because the commitments of a_i and α_i suffice to extract the set elements in the simulation proofs before the PRF parameters are generated and hence security can still be reduced to the weaker selective security notion for the underlying PRF. Looking ahead, the commitments of a_i, α_i and k_b will be used directly later in Round 2 of the DOPRF protocol for further computation avoiding the need to prove the consistency of x_i, a_i and α_i in batched C_{x_i} and batched C_{α_i}, which would have been the case if the parties commit only to their elements before the PRF parameter generation.

To enable batching the first component of the Camenisch-Shoup ciphertexts, every batched Camenisch-Shoup ciphertext has t slots. In Round 1 of the DOPRF protocol, P_2 will encrypt t copies of k_2, where the $\mathfrak{i}$-th copy of k_2 is encrypted in the $\mathfrak{i}$-th slot and the other slots are all 0. These encryptions will be used later in Round 2 of the shuffled DOPRF protocol to enable batching

Camenisch-Shoup encryptions of β_i.

Finally, in Round 2 of the DOPRF protocol, P_1 can make use of previously committed a_i, α_i, k_1 along with encryption of k_2 to batch Camenisch-Shoup encryptions and Pedersen commitments of β_i. The sigma protocols in this step can also be batched. The details of batching each sigma protocol are presented in Appendix C.

5 Batching Techniques

In this section we discuss batching techniques in various parts of our protocol. These techniques have a significant effect on our protocol's communication cost and may be of independent interest.

5.1 Batching Pedersen Commitments

As mentioned in Section 3.3, Pedersen commitments can be genenralized to allow committing to *vectors* of values. For batched commitments of vectors of length t, the parameters are group generators $g_1, \ldots, g_t, h \in \mathbb{G}$ such that $\log_{g_i} h$ is hard to compute for each i, and $\log_{g_i} g_j$ is hard to compute for any pair i, j. The commitment to a vector $\vec{x} = (x_1, \ldots, x_t)$ is $c = \prod_{i=1}^{t} g_i^{x_i} \cdot h^r$ where r is selected at random $r \xleftarrow{\$} \mathsf{ord}(\mathbb{G})$.

Batched Pedersen commitments are also compatible with sigma protocols of the knowledge and equality of exponents. To do so, the prover simply proves knowledge of all exponents simultneously. Furthermore, if the group $\mathbb{G}$ is one in which the Strong RSA assumption holds, then the following generalization of Theorem 3 from [CS03] holds: given randomly chosen $g_1, \ldots, g_t, h \in \mathbb{G}$, it is hard to find $w \in \mathbb{G}$ and $(a_1, ..., a_t, b, c)$ such that

$$w^c = \prod_{i=1}^{t} g_i^{a_i} \cdot h^b$$

Unless $c \mid a_i$ for all $i \in [t]$, and also $c \mid b$. The proof of this generalization closely follows from the proofs of Theorems 2 and 3 from [CS03].

Given these properties, we can replace most commitments in our protocols with batched commitments, that is, we commit to t values together. To enable this, each of our sigma protocols will commit to and prove statements about t messages simultaneously. Note that this reduces the number of commitments we send by a factor of t, but we still need to send one element per committed value in the last step of each sigma protocol. At first this does not seem to lead to a significant gain in efficiency. However, sigma protocols for batched commitments can also be batched, enabling the prover to send a single set of t elements in the last step to verify ℓ sigma protocols simultaneously. Combining the two forms of batching by setting t and ℓ to approximately $\sqrt{n}$, we can reduce the overall communication cost of the sigma protocols to be sublinear. We will discuss how to batch sigma protocols in Section 5.3, and we refer the reader to Appendix C.4 for a concrete example of batching sigma protocols for batched commitments.

5.2 Batching Camenisch-Shoup Encryption

We notice that Camenisch Shoup encryption introduces a $4\times$ expansion in the ciphertext as compared to the plaintext. This is due to the fact that a ciphertext contains 2 elements mod N^2 of

total length $4n$ bits (where $n = \log N$), while the ciphertext can only hold a message of $|n|$ bits. This causes a significant constant expansion to our protocol messages.

We describe various types of batching that enable reducing the expansion of Camenisch-Shoup encryption to be as close to $1\times$ as desired.

5.2.1 Computing mod N^{s+1}

Analogous to the Damgård-Jurik extension to the Paillier cryptosystem [DJ01], one can generalize the Camenisch-Shoup cryptosystem to compute modulo N^{s+1}. In more detail, the public key in this generalization consists of (N, g, y, s) where N is generated same as before, g is a random $2N^s$-th residue modulo N^{s+1}, and $y = g^x \mod N^{s+1}$ for a random $x \in \mathbb{Z}_{\lfloor N/4 \rfloor}$, and x is the secret key.

Similarly to the Damgård-Jurik extension, this generalization of Camenisch-Shoup encryption enables encrypting messages of size up to N^s. Concretely, given $m \in \mathbb{Z}_{N^s}$, it would be encrypted as $\mathsf{ct} = (g^r \mod N^{s+1}, (1+N)^m y^r \mod N^{s+1})$, where $r \xleftarrow{\$} \mathbb{Z}_{\lfloor N/4 \rfloor}$. Decryption is slightly more involved. To decrypt $\mathsf{ct} = (u, e)$, one must compute $e/(u^x) \mod N^{s+1}$ and then perform a recursive decoding to recover m, exactly as described in Section 3 of [DJ01].

Additionally, similar to the proof of Theorem 1 in [DJ01], the security of the generalized Camenisch-Shoup scheme follows from the Decisional Composite Residuousity Assumption.

We note that, with this generalization, one can encrypt a message of length $n \cdot s$ using a ciphertext of size $2 \cdot n \cdot (s+1)$, meaning that the expansion factor is reduced from $4\times$ to $\frac{2(s+1)}{s}\times$, which becomes arbitrarily close to $2\times$ as s grows.

5.2.2 Sharing the first ciphertext component

A remaining source of ciphertext expansion is that each ciphertext has 2 components, (u, e). One way to reduce this type of expansion is to have multiple components e that all share the first component u.

More concretely, we modify the scheme so that the public key consists of $(N, g, \{y_i\}_{i=1}^t)$, where $y_i = g^{x_i} \mod N^2$ for random $x_i \in \mathbb{Z}_{\lfloor N/4 \rfloor}$. The secret key becomes $\{x_i\}_{i=1}^t$.

This scheme allows encrypting t messages by $t+1$ components. Specifically, to encrypt messages $\{m_i\}_{i=1}^t$, one computes $u = g^r \mod N^2$ for $r \xleftarrow{\$} \mathbb{Z}_{\lfloor N/4 \rfloor}$, and $e_i = (1+N)^{m_i} \cdot y_i^r \mod N^2$ for each $i \in [t]$, and sets $\mathsf{ct} = (u, \{e_i\}_{i=1}^t)$. To decrypt a particular ciphertext, one simply decrypts each piece, computing $m_i = \frac{\left(\frac{e_i}{u^{x_i}} - 1\right) \mod N^2}{N}$.

This scheme is also entry-wise additively homomorphic. Given $\mathsf{ct} = (u, \{e_i\}_{i=1}^t)$ encrypting $\{m_i\}_{i=1}^t$ and $\mathsf{ct}' = (u', \{e_i'\}_{i=1}^t)$ encrypting $\{m_i'\}_{i=1}^t$, the ciphertext $\mathsf{ct}_{\mathsf{sum}} = (u \cdot u' \mod N^2, \{e \cdot e_i' \mod N^2\}_{i=1}^t)$ is an encryption of $\{m_i + m_i' \mod N\}_{i=1}^t$. One can also homomorphically multiply each underlying m_i with a single scalar a by computing $\mathsf{ct}^a = (u^a \mod N^2, \{(e_i)^a \mod N^2\}_{i=1}^t)$, which is an encryption of $\{a \cdot m_i \mod N\}_{i=1}^t$.

This optimization enables t messages of size n bits to be encrypted using a ciphertext of size $(t+1) \cdot 2n$ bits, which corresponds to an expansion factor of $\frac{2(t+1)}{t}$.

The two optimizations can be combined, meaning that for any choice s and t, we can encrypt t messages each of size $n \cdot s$ bits using a ciphertext of size $(s+1) \cdot (t+1) \cdot n$ bits. This means the ciphertext has an expansion of $\frac{(s+1)\cdot(t+1)}{s \cdot t}\times$. As t and s grow, this means we can make the ciphertext expansion as close to 1 as we like.

5.2.3 Encrypting multiple messages in a single ciphertext

Utilizing the batching techniques in the previous two subsections, one can reduce the ciphertext expansion of the Camenisch-Shoup encryption scheme, but the plaintext space becomes as large as N^s. We now describe how the plaintext space can be decomposed into slots of size B each. More concretely, each ciphertext can be viewed as having $t \cdot s'$ "slots" of messages $\leq B$, where $s' = \lfloor \frac{N^s}{B} \rfloor$. Recall that t comes from the fact that we encrypt t messages each of size up to N^s with shared first component. The s' component comes from the fact that the message space N^s is now divided into s' slots of size B each. Specifically, given $t \cdot s'$ messages $\{m_{i,j}\}_{i \in [t], j \in [s']}$ in $\mathbb{Z}_B$, we compute $m_i = \sum_{j=1}^{s'} m_{i,j} \cdot B^{j-1}$ for each $i \in [t]$ and then encrypt the t messages $\{m_i\}_{i=1}^{t}$. (Note that each $m_i \leq N^s$.) Given a public key $(g, \{y_i\}_{i \in [t]})$ the ciphertext is computed as follows:

$$\mathsf{ct} = \begin{cases} u = (g)^r \\ e_1 = (1+N)^{\sum_{j=1}^{s'} m_{1,j} \cdot B^{j-1}} \cdot (h_1)^r \\ \quad \vdots \\ e_i = (1+N)^{\sum_{j=1}^{s'} m_{i,j} \cdot B^{j-1}} \cdot (h_i)^r \\ \quad \vdots \\ e_t = (1+N)^{\sum_{j=1}^{s'} m_{t,j} \cdot B^{j-1}} \cdot (h_t)^r \end{cases}$$

We observe that the resulting encryption is slot-wise additively homomorphic as long as the sum in each slot never exceeds B. In addition, all the slots can be homomorphically multiplied by a single scalar simultaneously as long as the resulting value in each slot does not exceed B.

These slotted encryptions are compatible with all the other pieces of our protocol. In particular the following needed properties of the Camenisch-Shoup encryption scheme can be extended to the slotted encryptions (including in combination):

1. Proof that the value encrypted in a ciphertext is identical to the value underlying another commitment.

2. Proof that a ciphertext decrypts to a value underlying another commitment.

3. Proof that a ciphertext was produced by homomorphically adding a committed value to another ciphertext, and rerandomizing.

4. Proof that a ciphertext was produced by homomorphically scalar-multiplying a committed value to another ciphertext and rerandomizing.

5.2.4 Batching commitments of decrypted values

In our protocol (see Appendix C.7 for the specific step), we need to commit to a set of values $\{\beta_i\}$ that are decrypted from the batched Camenisch-Shoup ciphertexts and prove consistency between the committed values and decrypted values. We can batch the commitments as described in Section 5.1, and prove consistency between batched commitments with batched decryption. The high-level idea is the following. Given a set of commitments and ciphertexts, the verifier first picks a set of random coefficients $\{c_i\}$. Then both parties can compute a single commitment and a single

1. Prover samples $\widetilde{x} \xleftarrow{\$} [q]$ and sends $\widetilde{y} = g^{\widetilde{x}}$ to Verifier.
2. Verifier chooses random challenges $c_i \xleftarrow{\$} \{0,1\}^\lambda$ for $i \in [\ell]$, and sends to Prover.
3. Prover computes $\widehat{x} = \widetilde{x} + \sum_{i=1}^{\ell} c_i \cdot x_i \bmod q$, and sends $\widehat{x}$ to Verifier.
4. Verifier verifies that $g^{\widehat{x}} = \widetilde{y} \cdot \prod_{i=1}^{\ell} (y_i)^{c_i}$.

Figure 5: Example for batching sigma protocols.

encryption of a random linear combination of the underlying values, namely $\sum c_i \beta_i$. After that, the prover simply proves consistency between the resulting commitment and encryption. Our batched proof for this step has sublinear communication complexity.

5.3 Batching Sigma Protocols

In certain circumstances, it is possible to batch a set of ℓ sigma protocols that prove similar statements, such that the batched protocol has communication cost similar to a single sigma protocol. Batching sigma protocols is well-known in the literature [GLSY04, Gro09]. In this section we describe a variant that is compatible with range proofs, and in particular, induces much less slack in the range-proof bound.

We describe the technique by an example. Let g be a generator of a group $\mathbb{G}$ of order q, and let $\{y_i = g^{x_i}\}_{i \in [\ell]}$, where each $x_i \in [q]$. We give a batched sigma protocol in Figure 5 for the following ZK-AOK:

$$\mathsf{ZK\text{-}AoK}\left\{\{x_i\}_{i\in[\ell]} : y_i = g^{x_i} \quad \forall i \in [\ell]\right\}.$$

We can see in the figure that the prover sends a single group element in its first message (as opposed to ℓ group elements in an unbatched execution) and a single element in its response to the verifier (as opposed to ℓ elements in an unbatched execution). The verifier sends ℓ challenges instead of one, but the communication cost of these can be ignored if we use the Fiat-Shamir heuristic to make the protocol non-interactive. This means that the communication cost is essentially the same as a single unbatched sigma-protocol execution. Completeness of the protocol is straightforward. Next we prove its soundness and zero-knowledge property.

Soundness and Extraction. We construct a PPT extractor that interacts with a cheating prover and extracts valid witnesses $\{x_i\}_{i\in[\ell]}$. The extractor first executes the protocol honestly with the prover and obtains a transcript $(\widetilde{y}, \{c_i\}_{i\in[\ell]}, \widehat{x})$ such that $g^{\widehat{x}} = \widetilde{y} \cdot \prod_{i=1}^{\ell} y_i^{c_i}$.

Now the extractor rewinds the protocol to Step 2 and sends a different random challenge c_1' while keeping all the other challenges the same, and obtains $\widehat{x}'$ such that $g^{\widehat{x}'} = \widetilde{y} \cdot (y_1)^{c_1'} \prod_{i=2}^{\ell} (y_i)^{c_i}$. Combining the two equations, the extractor gets $g^{\Delta\widehat{x}} = y_1^{\Delta c}$ where $\Delta\widehat{x} = \widehat{x} - \widehat{x}'$ and $\Delta c = c_1 - c_1'$. Now the extractor can compute $x_1 = \Delta\widehat{x} \cdot (\Delta c)^{-1} \bmod q$. This process can be repeated for all $i \in [\ell]$ to extract all x_i.

Zero-knowledge. We prove this protocol is honest-verifier zero-knowledge by constructing a PPT simulator that does the following. First it samples $c_i \xleftarrow{\$} \{0,1\}^\lambda$ for all $i \in [\ell]$ and $\widehat{x} \xleftarrow{\$} [q]$, and

then computes $\widetilde{y} = g^{\widehat{x}} / \prod_{i=1}^{\ell} (y_i)^{c_i}$. Finally it outputs the transcript $(\widetilde{x}, \{c_i\}_{i \in [\ell]}, \widehat{x})$. The simulated transcript is statistically identical to the real protocol.

This batching technique extends naturally to more complex sigma protocols that prove relations between multiple elements and consistency between exponents. Concrete examples of the batched sigma protocols we use in our protocol can be found in Appendix C.

Effect of batching on range proofs. Batching has a small effect on the slack of range proofs that we consider. Recall that the size bound on a particular exponent x is related to the size of $\widehat{x}$, that is, the part of the prover's response related to that element. Batching ℓ sigma protocols increases the size of each element of the prover's response by a factor of ℓ. This is because the value needs to be big enough to statistically mask $\sum_{i=1}^{\ell} c_i \cdot x_i$, which is ℓ times larger than the unbatched case. Therefore, batching introduces an additional factor of ℓ to the proved range.

5.4 Multi-exponentiation Argument

In our protocol (see Appendix C.7 for the specific step), we will need to batch a set of arguments that an ElGamal ciphertext $\mathfrak{ct}'_i$ is a re-randomization of another ciphertext $\mathfrak{ct}_i$ raised to a hidden committed value β_i. Our idea is to first take a random linear combination of these equations and then prove an ElGamal ciphertext $\widetilde{\mathfrak{ct}}$ is the product of a set of known ciphertexts $\{\widetilde{\mathfrak{ct}}_i\}$ raised to a set of hidden committed values $\{\beta_i\}$, where the commitments are batched as described in Section 5.1. We notice that this can be achieved by a multi-exponentiation argument from the work of Bayer and Groth [BG12], which has sublinear communication complexity. One subtlety is that the values $\{\beta_i\}$ are committed in the group of the Camenisch-Shoup encryption for proving consistency with the decrypted values, but to the apply multi-exponentiation argument, they must be committed in the group of the ElGamal encryption. Therefore, we commit to $\{\beta_i\}$ in both groups and prove consistency between the commitments. Since all the commitments and sigma protocols can be batched, the overall communication complexity is sublinear.

6 Communication, Computation and Monetary Costs

In this section, we present the communication, computation and monetary costs of our protocol. The offline phase for generating parameters for the different primitive we will use has a fixed cost, which includes four ZK-AoK of exponent per party plus one proof that a modulus N is a product of safe primes [CM99], which requires $O(\kappa^2 \log N)$ communication and computation where κ is the security parameter for the soundness of the last proof.

For our online phase, we have several batching optimizations described in Section 5 that allow us to achieve different trade-offs between communication and computation. Thus, we state our efficiency estimates parameterized with the different batching parameters presented in Table 1 that we apply for the commitments and encryptions. Our shuffled DOPRF has 3 rounds, each of which has an associated sigma protocol. Wherever the sigma protocols can be batched, we batch them into a single execution, and this is reflected in the costs. The specifics of the batching can be seen in Appendix C.

In Table 2 we present the computation and communication cost estimates for the different phases of out protocol. There are three different types of computational operations we perform in the protocol, namely group operations in $\mathbb{G}$, exponentiations mod N (for commitments), and

Notation	Parameter Meaning
n	number of inputs in each set
$\mathbb{G}$	group for OPRF
$size_{\mathbb{G}}$	size of elements in $\mathbb{G}$
N	RSA modulus
λ	security parameter for sigma protocol soundness and hiding
s_{cam}	modulus parameter for CS encryptions, their modulus will be $N^{s_{cam}+1}$
s'_{cam}	number of plaintexts that fit in the message space $N^{s_{cam}+1}$
t_{cam}	number of components e_i per CS encryption that share the first component u
N_{cam}	total number of CS ciphertexts ($\lceil n/(s'_{cam} \cdot t_{cam}) \rceil$)
s_{ped}	number of values committed in a Pedersen vector commitment in DOPRF round 2
n_{ped}	number of Pedersen vector commitments in DOPRF round 2 ($\lceil n/s_{ped} \rceil$)
n'_{cam}	number of batched CS ciphertexts per batched Pedersen commitment $\lceil s_{ped}/(s'_{cam} \cdot t_{cam}) \rceil$
$m_{multiexp}$	dimension m to use in the multiexponentation proof from Bayer et al [BG12] in DOPRF Round 3.

Table 1: Parameter notation

exponentiations mod $N^{s_{cam}+1}$ for Camenisch-Shoup encryption. There are also 4 types of elements we communicate: group elements in $\mathbb{G}$, elements modulo N, elements modulo N^{s+1}, and sigma protocol response messages from the prover. The entries of Table 2 reflect counts of each of these types of operations and elements transferred.

	Computation	Communication
DOPRF Round 1		
Messages	2 exp mod N $+t_{cam} \cdot (t_{cam}+1)$ exp mod $N^{s_{cam}+1}$	$\lvert N \rvert \cdot (1 + t_{cam} \cdot (t_{cam}+1) \cdot (s_{cam}+1))$
Sigma Protocol	5 exp mod N $+3t_{cam} \cdot (t_{cam}+1)$ exp mod $N^{s_{cam}+1}$	$\lvert N \rvert \cdot (t_{cam} + 3 + t_{cam} \cdot (t_{cam}+1) \cdot (s_{cam}+1))$
DOPRF Round 2		
Messages	$(n + n_{cam}) \cdot (t_{cam}+1)$ exp mod $N^{s_{cam}+1}$ $+(3n + 3n_{ped})$ exp mod N $+ n$ exp in $\mathbb{G}$	$(n_{cam} \cdot (t_{cam}+1)(s_{cam}+1) \cdot \lvert N \rvert))$ $+n \cdot size_{\mathbb{G}} + 3n_{ped} \cdot \lvert N \rvert$
Sigma Protocol	$2 \cdot (n_{cam} + s_{ped}) \cdot n_{sig}(t_{cam}+1)$ exp mod $N^{s_{cam}+1}$ $(10s_{ped}+10) + 5n_{ped}$ exp mod N $+ (2s_{ped}+n)$ exp in $\mathbb{G}$	$\lvert N \rvert \cdot n'_{cam}((s_{cam}+1) \cdot (t_{cam+1}) + \log n_{ped} + k)$ $+(5s_{ped}+8) \cdot \lvert N \rvert + s_{ped} \cdot size_{\mathbb{G}}$
DOPRF Round 3		
Messages	n/s'_{cam} exp mod $N^{s_{cam}+1}$ $+ (n + n_{ped})$ exp mod N $+4n + n_{ped}$ exp in $\mathbb{G}$	$(3n + n_{ped}) \cdot size_{\mathbb{G}} + n_{ped} \lvert N \rvert$
Sigma Protocol 1	$(2 + n_{ped}) \cdot (n_{cam}+1) \cdot (t_{cam}+1)$ exp mod $N^{s_{cam}+1}$ $+2(s_{ped}+1) + n_{ped}$ exp mod N $+2(s_{ped}+1) + n_{ped}$ exp in $\mathbb{G}$	$(n_{cam}+1) \cdot (s_{cam}+1) \cdot (t_{cam}+1) \lvert N \rvert$ $+(\lvert N \rvert + k) t_{cam}$ $+s_{ped} \cdot (3k + 2size_{\mathbb{G}})$
Sigma Protocol 2	$2n(m_{multiexp} + 6 \cdot \lceil n\ m_{multiexp} \rceil$ + exp in $\mathbb{G}$	$(5m_{multiexp} + \lceil n\ m_{multiexp} \rceil + 10) \cdot size_{\mathbb{G}}$

Table 2: Counts of various operations performed in each step of the DOPRF protocol, and corresponding communication cost.

We will compare our protocol's cost against the baseline, namely the semi-honest Diffie-Hellman based intersection-sum protocol [IKN+17]. In our concerete instantiation, we use safe RSA moduli of length 1536 bits. We use NIST curve prime256v1 as our group $\mathbb{G}$.

To minimize communication costs, in the first and seconds rounds of the shuffled DOPRF protocol, we set $s_{ped} = \sqrt{n}$ and batch $\sqrt{n}$ sigma protocols together. We further set $t_{cam} = 8$. $s_{cam} = 4$, $s'_{cam} = 8$ and $m_{multiexp} = 8$. We compare costs with the DDH-based shuffled DOPRF with semi-honest security. The measurements appear in Table 3.

We briefly discuss how we choose our parameters. First we discuss our choice of s_{ped}. In Round 2 of the DOPRF, batching Pedersen commitments allows us to send 1 element mod N instead of

	Our Protocol		DDH-based		
Input size	Comm. (KB)	Comp. (s)	Comm. (KB)	Comp. (s)	Comm. Expansion
2^{12}	1,287	1,150	256	0.71	5.03 ×
2^{16}	17,716	17,865	4,096	11.39	4.325 ×
2^{20}	275,675	284,075	65,536	182.29	4.21 ×

Table 3: Comparison of communication and computation costs between our shuffled DOPRF protocol with parameters set to minimize communication, and the baseline protocol, namely the semi-honest DDH-based shuffled DOPRF.

s_{ped} elements in the Round 2 messages. However, each sigma protocol statement in this round now also grows to be of length s_{ped}, since we must prove knowledge of all values contained in a commitment together. Since each sigma protocol is of size s_{ped} individually, the batched sigma protocol is also be of length s_{ped}. In order to minimize both the number of commitments sent and the size of the batched sigma protocol, we set $s_{ped} = \sqrt{n}$, and $b_{sig} = \sqrt{n}$.

We note that generating the messages of the DOPRF Round 2 constitutes the computation bottleneck of the protocol. In this round, for each entry in the Receiver's set, the Receiver has to perform a homomorphic Camenisch-Shoup scalar multiplication with the encrypted key, and homomorphically add it to its encrypted and masked entry. In fact, the overall computation scales with t_{cam}, the number of components in the Camenisch-Shoup ciphertext. This means that if we increase the number of components of the Camenisch-Shoup ciphertexts, we end up greatly increasing the computation. Furthermore, when we increase the parameter s_{cam}, we are performing operations in the substantially larger group $n^{s_{cam}+1}$, which induces non-linearly increasing computation cost. In Table 4, we attempt to minimize computation, by reducing t_{cam} to 2, s_{cam} to 1 and s'_{cam} to 2 . In this case, communication cost increases by about 60%, but computation cost drops by about 90%.

	Our Protocol			DDH-based			
Input size	Comm(KB)	Comp(s)	Cost(c)	Comm(KB)	Comp(s)	Cost(c)	Cost Increase
2^{12}	1,893	141	0.053	256	0.71	0.002	24.9×
2^{16}	28,289	2,215	0.831	4,096	11.39	0.034	24.2×
2^{20}	436,719	35,583	13.1	65,536	182.29	0.551	24.00×

Table 4: Comparison of communication and computation costs between our shuffled DOPRF protocol when we set parameters to minimize computational cost. These parameters also minimize monetary cost.

To compare monetary costs, we use the costs from Google Cloud Platform.[1] The costs are given in Table 5. For computation, we use the price for pre-emptible virtual CPUs, which correspond to machines with an Intel Xeon E5 processor and 3.75 GB of memory, which matches the machines we used for benchmarking. We consider pre-emptible computation to capture the offline batch-processing scenario described by works that deploy PSI in practice [IKN+17]. We also use the cheapest tier of network cost, considering the cost for internet egress, since that captures the scenario of the two parties being in different datacenters or clouds. We note that, at the time of publication, all the major cloud providers have costs that are within a tight range.

[1]See https://cloud.google.com/compute/network-pricing/ for the network cost and https://cloud.google.

Network cost(USD per GB)	Computational cost (USD per CPU-hour)
$0.08	0.01

Table 5: Costs for network and computation on Google Cloud Platform.

	Input size 2^{12}			Input size 2^{16}			Input size 2^{20}		
	Comm	Comp	Cost	Comm	Comp	Cost	Comm	Comp	Cost
DDH-DOPRF (semihonest)	256	0.71	0.002	4096	11.39	0.034	65536	182.29	0.55
Sort-Compare-Shuffle [HEK12]	209920	0.61	1.60	4941824	12.65	37.7	108691456	235.3	829.3
EC-ROM (one-sided PSI) [RR17b]	4915.2	0.19	0.037	80896	0.94	0.61	1353728	12.6	10.3
DE-ROM (one-sided PSI) [RR17b]	3584	0.23	0.027	62464	1.3	0.47	1118208	18	8.53
Our SDOPRF (low comm.)	1287	1150	0.329	17716	17865	5.09	275675	284075	81.01
Our SDOPRF (low comp.)	1893	141	0.05	28289	2215	0.83	436719	35583	13.21

Table 6: Comparison of computation, communication and monetary costs of our protocols compared to related works. Monetary costs use the values in Table 5. Communication cost is in KB, Time is in seconds, and Cost is in cents (USD).

Comparison with existing works. In Table 6, we compare concrete costs of our protocol against existing works that achieve security against malicious adversaries. The key comparison is against the Sort-Compare-Shuffle (SCS) approach of Huang et al [HEK12], which is the only existing work that is compatible with malicious security, two sided output, and computing a function on associated values in the intersection. We note that both our SDOPRFs have significantly lower communication, and crucially, lower concrete monetary cost. In particular, the "Low Computation" variant of our SDOPRF has monetary cost 30× less for 2^{12} entries, and 64× less for 2^{20} entries. We note that the SCS approach has lower computation costs and end-to-end running time, but that in the batch-processing setting, the computation cost is less of a factor than concrete monetary costs, since responses are not needed in real time.

We also compare against the most efficient one-sided malicious PSI works of Rindal et al. [RR17b], and show that our protocols are in the same ballpark of total monetary cost. In particular, the "Low Computation" variant of our SDOPRF has monetary cost about 1.5× that of the DE-ROM variant of [RR17b]. We note that [RR17b] do not easily support two sided output or computing over the intersection. We believe the modest increased cost of our protocol is reasonable in order to support these additional functionalities.[2]

References

[AES03] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003.

[ARF+10] Benny Applebaum, Haakon Ringberg, Michael J Freedman, Matthew Caesar, and Jennifer Rexford. Collaborative, privacy-preserving data aggregation at scale. In *Privacy Enhancing Technologies Symposium*, 2010.

`com/compute/vm-instance-pricing` for the computation cost.

[2]Concurrent to our work, Pinkas at el. [PRTY20] present a new one-sided malicious PSI that achieves better efficiency than [RR17b], but we note that their protocol also does not easily support our two-sided functionality.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *EUROCRYPT*, 2004.

[BBDC+11] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In *ACM CCS*, 2011.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO*. 2009.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, 2012.

[BHLB11] Elie Bursztein, Mike Hamburg, Jocelyn Lagarenne, and Dan Boneh. Openconflict: Preventing real time map hacks in online games. In *IEEE Symposium on Security and Privacy*, 2011.

[BMR10] Dan Boneh, Hart William Montgomery, and Ananth Raghunathan. Algebraic pseudorandom functions with improved efficiency from the augmented cascade. In *ACM CCS*, 2010.

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, 2000.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, 1997.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, 1993.

[CKRS09] Jan Camenisch, Markulf Kohlweiss, Alfredo Rial, and Caroline Sheedy. Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In *PKC*, 2009.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *EUROCRYPT*, 1999.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *SCN*, 2018.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO*, 1997.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, 2003.

[CZ09] Jan Camenisch and Gregory M Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security*, 2009.

[Dam02] Ivan Damgard. On Σ-protocols. 2002. http://www.cs.au.dk/~ivan/Sigma.pdf.

[DCGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS*, 2012.

[DCKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, 2010.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS*, 2013.

[DD15] Sumit Kumar Debnath and Ratna Dutta. Secure and efficient private set intersection cardinality using bloom filter. In *International Information Security Conference*, 2015.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC*, 2001.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, 2009.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, 2005.

[EFG+15] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *Australasian Conference on Information Security and Privacy*, 2015.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 1985.

[FHNP16] Michael J Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *J. of Cryptology*, 2016.

[FNO18] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. 2018.

[FNO19] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In *WPES@CCS*, 2019.

[FNP04] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, 1997.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *EUROCRYPT*, 1986.

[GLSY04] Rosario Gennaro, Darren Leigh, Ravi Sundaram, and William Yerazunis. Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In *ASIACRYPT*, 2004.

[GMR98] Rosario Gennaro, Daniele Micciancio, and Tal Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *ACM CCS*, 1998.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, 2009.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[HFH99] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *ACM conference on Electronic commerce*, 1999.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, 2008.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC*, 2010.

[IKN+17] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. Cryptology ePrint Archive, Report 2017/738, 2017. https://eprint.iacr.org/2017/738.

[JL09] Stanisław Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, 2009.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *ACM CCS*, 2016.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *ACM CCS*, 2017.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *CRYPTO*, 2005.

[Lam16] Mikkel Lambæk. Breaking and fixing private set intersection protocols. Cryptology ePrint Archive, Report 2016/665, 2016. https://eprint.iacr.org/2016/665.

[LCYL11] Ming Li, Ning Cao, Shucheng Yu, and Wenjing Lou. Findu: Privacy-preserving personal profile matching in mobile social networks. In *IEEE INFOCOM*, 2011.

[MSK02] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 2002.

[NAA+09] G Sathya Narayanan, T Aishwarya, Anugrah Agrawal, Arpita Patra, Ashish Choudhary, and C Pandu Rangan. Multi party distributed private matching, set disjointness and cardinality of set intersection with information theoretic security. In *CANS*, 2009.

[NDCD+13] Marcin Nagy, Emiliano De Cristofaro, Alexandra Dmitrienko, N Asokan, and Ahmad-Reza Sadeghi. Do i know you?: efficient and privacy-preserving common friend-finder protocols and applications. In *ACSAC*, 2013.

[NMH+10] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *USENIX Security*, 2010.

[NTL+11] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, Dan Boneh, et al. Location privacy via private proximity testing. In *NDSS*, volume 11, 2011.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 2004.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse ot extension. In *CRYPTO*, 2019.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *EUROCRYPT*, 2020.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: private set intersection using permutation-based hashing. In *USENIX Security*, 2015.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, 2019.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. In *EUROCRYPT*, 2018.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *USENIX Security*, 2014.

[RR17a] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT*, 2017.

[RR17b] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *ACM CCS*, 2017.

[SFF14] Aaron Segal, Bryan Ford, and Joan Feigenbaum. Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In *FOCI*, 2014.

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

A Selective Secure Pseudorandom Function

In this section we define and construct a selective secure pseudorandom function, where the inputs to the PRF are chosen by the adversary in advance in the security game. This weaker definition of PRF is sufficient for our private intersection-sum protocol. We prove our PRF construction is selective secure under the decisional q-DHI assumption.[3]

The above security notion for the PRF suffices for our private intersection-sum protocol since we make the two parties first commit to their own inputs along with a zero-knowledge proof of knowledge and then jointly decide the PRF parameters. Thus, the selective security of the PRF suffices for the simulation-based proof of the overall constructions since the simulator the private intersection-sum can first parties' inputs from the commitments and then rely on the selective security of the PRF.

We state formally the selectively security PRF definition as follows:

Definition A.1 (Selectively Secure PRF)**.** *Let group $\mathbb{G}$ be a group of order p for which the decisional q-DHI assumption holds. Let kGen be an algorithm which on input a security parameter 1^λ generates a public parameter pp and secret key k. We say that the function $F_k : \mathbb{Z}_p^* \to \mathbb{G}$ is a q-selective secure pseudorandom function (family) if*

$$\left| \mathsf{Pr}\left[\mathcal{A}\left(F_k(x_1), \ldots, F_k(x_q), F_k(x_{q+1})\right) = 1 \;\middle|\; \begin{array}{l}(x_1, \ldots, x_q, x_{q+1}) \leftarrow \mathcal{A}(1^\lambda); \\ (\mathsf{pp}, k) \leftarrow \mathsf{kGen}(1^\lambda)\end{array} \right] \right.$$
$$\left. - \mathsf{Pr}\left[\mathcal{A}\left(F_k(x_1), \ldots, F_k(x_q), y\right) = 1 \;\middle|\; \begin{array}{l}(x_1, \ldots, x_q, x_{q+1}) \leftarrow \mathcal{A}(1^\lambda); \\ (\mathsf{pp}, k) \leftarrow \mathsf{kGen}(1^\lambda); y \xleftarrow{\$} \mathbb{G}\end{array} \right] \right| \leq \mathsf{negl}(\lambda).$$

A.1 Construction

We construct a PRF that satisfies the above definition as follows: let $\mathbb{G}$ be a group of order p where the decisional q-DHI assumption holds, then our PRF is defined by the following algorithms:

- $(\mathsf{pp}, k) \leftarrow \mathsf{kGen}(1^\lambda)$: on input the security parameter λ, outputs public parameter pp and a secret key $k \xleftarrow{\$} \mathbb{Z}_p^*$.
- $\sigma \leftarrow F_k(x)$: on input x output PRF evaluation $F_k(x) = g^{\frac{1}{k+x}}$.

A.2 Security Proof

We prove that F_k is a q-selective secure PRF by showing that if there exists an adversary $\mathcal{A}$ that breaks its selective security, then we can construct another adversary $\mathcal{B}$ that breaks the decisional q-DHI assumption.

First, $\mathcal{B}$ is given elements $(g, g^\alpha, \ldots, g^{\alpha^q})$ along with h that could be $g^{1/\alpha}$ or a random group element. It is also given $(x_1, \ldots, x_q, x_{q+1})$ chosen by $\mathcal{A}$.

Let $\beta := \alpha - x_{q+1}$. Define $f(y)$ as a polynomial

$$f(y) = \prod_{i=1}^{q}(y - x_{q+1} + x_i) = \prod_{i=1}^{q}(y - x_{q+1} + x_i) = \sum_{i=0}^{q} c_i \cdot y^i$$

[3] We note that [BMR10] consider a similar question, but their construction is not oblivious.

for some coefficients $c_0, c_1, \ldots, c_q$. Notice that $f(\alpha) = \prod_{i=1}^{q}(\beta + x_i)$. Then $\mathcal{B}$ can compute

$$\widetilde{g} = g^{f(\alpha)} = g^{\sum_{i=0}^{q} c_i \cdot \alpha^i} = \prod_{i=0}^{q} \left(g^{\alpha^i}\right)^{c_i}.$$

$\mathcal{B}$ implicitly sets the public parameter $\mathsf{pp} = \widetilde{g}$ and secret key $k = \beta$ as the PRF key. Notice that $\widetilde{g}$ and k are both uniformly distributed in $\mathbb{G}$ and $\mathbb{Z}_p^*$, respectively. To compute $F_k(x_i)$ for $1 \leq i \leq q$,

$$F_k(x_i) = \widetilde{g}^{\frac{1}{\beta + x_i}} = g^{\prod_{j=1, j\neq i}^{q}(\beta + x_j)} = g^{\sum_{j=0}^{q-1} d_j^{(i)} \cdot \alpha^j} = \prod_{j=0}^{q-1} \left(g^{\alpha^j}\right)^{d_j^{(i)}}.$$

For the $q-1$ coefficients $d_j^{(i)}$ that result from viewing $\prod_{j=1, j\neq i}^{q}(\beta + x_j)$ as a polynomial in α. Finally, to compute $F_k(x_{q+1})$,

$$\begin{aligned} F_k(x_{q+1}) =& \widetilde{g}^{\frac{1}{\beta + x_{q+1}}} = \widetilde{g}^{\frac{1}{\alpha}} = g^{f(\alpha) \cdot \frac{1}{\alpha}} = g^{\sum_{i=0}^{q} c_i \cdot \alpha^i \cdot \frac{1}{\alpha}} \\ =& g^{\sum_{i=1}^{q} c_i \cdot \alpha^{i-1} + \frac{c_0}{\alpha}} = \prod_{i=1}^{q} \left(g^{\alpha^{i-1}}\right)^{c_i} \cdot \left(g^{\frac{1}{\alpha}}\right)^{c_0}. \end{aligned}$$

$\mathcal{B}$ computes

$$y = \prod_{i=1}^{q} \left(g^{\alpha^{i-1}}\right)^{c_i} \cdot h^{c_0},$$

and gives $(F_k(x_1), \ldots, F_k(x_q), y)$ to $\mathcal{A}$. Notice that if $h = g^{1/\alpha}$, then $y = F_k(x_{q+1})$; otherwise y is a random element in $\mathbb{G}$. Therefore, if $\mathcal{A}$ breaks the selective security of the PRF, then $\mathcal{B}$ breaks the decisional q-DHI assumption.

B Security Analysis

Correctness of the protocol for honest participants can be verified by inspection. Correctness holds even when one of the parties is malicious, following from the fact that each party proves it performed the computation each step honestly.

B.1 Security Against Malicious P_1

We first prove security against malicious P_1. Let $\mathsf{Real}_{\Pi,\mathcal{A}}((X, V), Y)$ denote the output of the adversary $\mathcal{A}$ (i.e., malicious P_1) in the real-world execution of our protocol Π. We construct a PPT simulator $\mathcal{S}$ such that

$$\mathsf{Real}_{\Pi,\mathcal{A}}((X, V), Y) \overset{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S}}((X, V), Y),$$

where $\mathsf{Ideal}_{\mathcal{F},\mathcal{S}}((X, V), Y)$ denotes the output of $\mathcal{S}$ in the ideal-world execution.

We construct a simulator $\mathcal{S}$ in Figure 6. The simulator executes a simulated protocol with $\mathcal{A}$ which we prove is indistinguishable from $\mathcal{A}$'s view in a real-world protocol with an honest P_2.

Theorem B.1. *For any PPT adversaries $\mathcal{A}$ and input $((X, V), Y)$,*

$$\mathsf{Real}_{\Pi,\mathcal{A}}((X, V), Y) \overset{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S}}((X, V), Y).$$

Run the protocol with $\mathcal{A}$ behaving like P_2 honestly expect the following:

One-time setup. Extract $\mathcal{A}$'s secret key share tsk_1 in Step 3 and compute $\mathsf{tsk} = \mathsf{tsk}_1 + \mathsf{tsk}_2$.

Online phase.

- In Step 2, extract $\mathcal{A}$'s input $\{x_i\}_{i\in[n_1]}$ from the ZK-AOKs. Commit to 0 instead of y_i and replace its ZK-AOKs with simulated ones.
- In Shuffled DOPRF 1 Round 2, decrypt ct_{v_i} by tsk to obtain v_i. Send $\{x_i, v_i\}_{i\in[n_1]}$ to $\mathcal{F}$ and get back the intersection-cardinality $|I|$ and intersection-sum S.
- In Shuffled DOPRF 2 Round 2, for each $i \in [n_2]$:
  (a) Sample $\sigma_i \xleftarrow{\$} \mathbb{G}$, $r_i \xleftarrow{\$} [q^2 \cdot 2^\lambda]$, and compute $\mathsf{g}_i = \sigma_i^{1/r_i}$. Let $R_2 = \{\sigma_i\}_{i\in[n_2}$.
  (b) Compute $\mathsf{C}_{a_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(0)$, $\mathsf{C}_{b_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(0)$, $\mathsf{C}_{\alpha_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(0)$, $\mathsf{ct}_{\beta_i} \leftarrow \mathsf{CS\_Enc}_{\mathsf{pk}_1}(r_i)$.
  (c) Send $(\mathsf{C}_{a_i}, \mathsf{C}_{b_i}, \mathsf{C}_{\alpha_i}, \mathsf{ct}_{\beta_i}, \mathsf{g}_i)$ to $\mathcal{A}$ together with a simulated ZK-AOK.
- In Shuffled DOPRF 1 Round 3, sample $R_1 = \{\sigma'_i\}_{i\in[n_1]}$ such that $|R_1 \cap R_2| = |I|$. In Step (e), send encryption of σ'_i along with a simulated ZK-AOK.
- In Step 5, send a fresh encryption of S under tpk_1 to $\mathcal{A}$ along with a simulated proof for correct half decryption.

Finally, output whatever $\mathcal{A}$ outputs.

Figure 6: The simulator for malicious P_1.

Proof. We prove the indistinguishability of $\mathcal{A}$'s view in the simulated protocol and in a real-world protocol via a hybrid argument.

Hyb_0 $\mathcal{A}$'s view of a real-world execution with P_2.

Hyb_1 Same as Hyb_0 except that in Shuffled DOPRF 2 Round 2, P_2 computes $\mathsf{C}_{a_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(0)$, $\mathsf{C}_{b_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(0)$, $\mathsf{C}_{\alpha_i} \leftarrow \mathsf{com}_{\mathfrak{g}_1,\mathfrak{h}_1}(0)$ for each $i \in [n_2]$ and sends to $\mathcal{A}$ together with simulated ZK-AOKs. Hyb_1 is computationally indistinguishable from Hyb_0 because of the hiding property of the commitment scheme and zero-knowledge property of the ZK-AOKs.

Hyb_2 Same as Hyb_1 except that P_2 does the following:

- In Step 2 of the online phase, extract $\mathcal{A}$'s input $\{x_i\}_{i\in[n_1]}$ from the ZK-AOKs.
- In Shuffled DOPRF 2 Round 1, extract k_1 from the ZK-AOK.
- In Shuffled DOPRF 2 Round 2, compute $\sigma_i = F_{k_1+k_2}(y_i)$, sample $r_i \xleftarrow{\$} [q^2 \cdot 2^{\lambda-2}]$, and compute $\mathsf{ct}_{\beta_i} \leftarrow \mathsf{CS\_Enc}_{\mathsf{pk}_1}(r_i), \mathsf{g}_i = \sigma_i^{1/r_i}$ for each $i \in [n_2]$. Update the corresponding simulated ZK-AOKs.
- In Shuffled DOPRF 1 Round 3, compute $\sigma'_i = F_{k_1+k_2}(x_i)$ for all $i \in [n_1]$ and use those with simulated ZK-AOKs.

Hyb_2 and Hyb_1 are computationally indistinguishable based on the zero-knowledge property of the ZK-AOKs sent by P_2 and soundness of the ZK-AOKs sent by $\mathcal{A}$.

Hyb_3 Same as Hyb_2 but in Shuffled DOPRF 1 Round 1, P_2 sends $\mathsf{CS\_Enc}_{\mathsf{pk}_2}(0)$ instead of ct_{k_2} along with a simulated ZK-AOK. Note that while the encryption of k_2 is changed to be to 0, P_2 continues to use a randomly chosen k_2 when computing $F_{k_1+k_2}(\cdot)$ in the remainder of the protocol. Hyb_3 and Hyb_2 are computationally indistinguishable based on the semantic security of ct_{k_2}.

Hyb_4 Same as Hyb_3 except that P_2 computes a random function $F(x)$ in the protocol instead of $F_{k_1+k_2}(x)$ and updates the simulated ZK-AOKs correspondingly. Hyb_4 and Hyb_3 are computationally indistinguishable based on the selective security of the PRF. In particular, if $\mathcal{A}$ can distinguish between Hyb_4 and Hyb_5, then we can construct a PPT adversary $\mathcal{B}$ that can break the selective security of the PRF. The adversary $\mathcal{B}$ runs the protocol with $\mathcal{A}$ as in Hyb_5 and extracts all the inputs $\{x_i\}_{i\in[n_1]}$ of $\mathcal{A}$. Then $\mathcal{B}$ feeds all the inputs to the security game of the PRF and obtains the public parameters pp along with values $\{\sigma_i\}_{i\in[n_1]}$, which could be PRF values or random values. In Step 3 of the online phase when the two parties jointly decide on a random generator g, $\mathcal{B}$ makes pp the generator. Later when P_2 computes the PRF values, $\mathcal{B}$ instead uses $\{\sigma_i\}_{i\in[n_1]}$. Since $\mathcal{A}$ can distinguish between Hyb_4 and Hyb_3, $\mathcal{B}$ can distinguish whether σ_i's are PRF values or truly random and break the selective secure PRF.

Hyb_5 Same as Hyb_4 except that P_2 uses random elements sampled from $\mathbb{G}$ instead of computing $F(x)$. Hyb_5 is statistically identical to Hyb_4.

Hyb_6 Same as Hyb_5 but in Shuffled DOPRF 1 Round 1, P_2 send $\mathsf{CS\_Enc}_{\mathsf{pk}_2}(k_2)$ instead of $\mathsf{CS\_Enc}_{\mathsf{pk}_2}(0)$ with updated simulated ZK-AOK. Hyb_6 and Hyb_5 are computationally indistinguishable based on the semantic security of ct_{k_2}.

Hyb_7 Same as Hyb_6 but in Step 2 of the online phase, P_2 commits to 0 instead of y_i's and simulates the ZK-AOKs. Hyb_7 and Hyb_6 are computationally indistinguishable because of the perfect hiding property of the commitment scheme.

Hyb_8 Same as Hyb_7 except that P_2 does the following:

- In Step 3 of the one-times setup, extract $\mathcal{A}$'s secret key share tsk_1 and compute $\mathsf{tsk} = \mathsf{tsk}_1 + \mathsf{tsk}_2$.
- In Shuffled DOPRF 1 Round 2, decrypt ct_{v_i} by tsk to obtain v_i for all $i \in [n_1]$. Compute the intersection-sum S.
- In Step 5 of the online phase, send a fresh encryption of S under tpk_1 to $\mathcal{A}$ along with an updated simulated proof.

Hyb_8 and Hyb_7 are computationally indistinguishable by the hiding property of the 2-out-of-2 threshold-encryption scheme together with the zero-knowledge property of the verifiable half-decryption proof.

Hyb_9 Same as Hyb_8 but in Shuffled DOPRF 1 Round 3 Step (e), P_2 sends randomly shuffled encryption of σ_i' with updated ZK-AOKs. Hyb_9 is computationally indistinguishable from Hyb_8 because of the hiding property of the ElGamal encryption scheme and zero-knowledge property of the ZK-AOKs.

Run the protocol with $\mathcal{A}$ behaving like P_1 honestly expect the following:

Online phase.

- In Step 2, extract $\mathcal{A}$'s input $\{y_i\}_{i\in[n_2]}$ from the ZK-AOKs. Commit to 0 instead of x_i and replace its ZK-AOKs with simulated ones. Send $\{y_i\}_{i\in[n_2]}$ to $\mathcal{F}$ and get back the intersection-cardinality $|I|$ and intersection-sum S.
- In Shuffled DOPRF 1 Round 2, for each $i \in [n_1]$:
  (a) Sample $\sigma_i \xleftarrow{\$} \mathbb{G}$, $r_i \xleftarrow{\$} [q^2 \cdot 2^\lambda]$, and compute $\mathsf{g}_i = \sigma_i^{1/r_i}$. Let $R_1 = \{\sigma_i\}_{i\in[n_1]}$.
  (b) Compute $\mathsf{C}_{a_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(0)$, $\mathsf{C}_{b_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(0)$, $\mathsf{C}_{\alpha_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(0)$, $\mathsf{ct}_{\beta_i} \leftarrow \mathsf{CS\_Enc}_{\mathsf{pk}_2}(r_i)$.
  (c) Send $(\mathsf{C}_{a_i}, \mathsf{C}_{b_i}, \mathsf{C}_{\alpha_i}, \mathsf{ct}_{\beta_i}, \mathsf{g}_i)$ to $\mathcal{A}$ together with a simulated ZK-AOK.
  (d) Compute $\mathsf{ct}_{v_i} \leftarrow \mathsf{Exp\_EG\_Enc}_{\mathsf{tpk}}(0)$ and send to $\mathcal{A}$.
- In Shuffled DOPRF 2 Round 3, sample $R_2 = \{\sigma'_i\}_{i\in[n_2]}$ such that $|R_1 \cap R_2| = |I|$. In Step (e), send encryption of σ'_i along with a simulated ZK-AOK.
- In Step 5, send a fresh encryption of S under tpk_2 to $\mathcal{A}$ along with a simulated proof for correct half decryption.

Finally, output whatever $\mathcal{A}$ outputs.

Figure 7: The simulator for malicious P_2.

Hyb_{10} Same as Hyb_9 except that in Shuffled DOPRF 1 Round 3 Step (e), σ'_i values are replaced with random strings, such that there are exactly $|I|$ repeated values in the two shuffled DOPRF protocols. Hyb_{10} and Hyb_9 are computationally indistinguishable based on the zero-knowledge property of the shuffle proof.

Hyb_{11} $\mathcal{A}$'s view in an ideal-world execution with $\mathcal{S}$. Hyb_{11} and Hyb_{10} are identical.

□

B.2 Security Against Malicious P_2

To prove security against malicious P_2, we construct another simulator in Figure 7 and prove $\mathsf{Real}_{\Pi,\mathcal{A}}((X,V),Y) \overset{c}{\approx} \mathsf{Ideal}_{\mathcal{F},\mathcal{S}}((X,V),Y)$ by arguing the indistinguishability of $\mathcal{A}$'s view in the simulated protocol and in a real-world protocol.

Hyb_0 $\mathcal{A}$'s view of a real-world execution with P_1.

Hyb_1 Same as Hyb_0 except that in Shuffled DOPRF 1 Round 2, P_1 computes $\mathsf{C}_{a_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(0)$, $\mathsf{C}_{b_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(0)$, $\mathsf{C}_{\alpha_i} \leftarrow \mathsf{com}_{\mathfrak{g}_2,\mathfrak{h}_2}(0)$ for each $i \in [n_1]$ and sends to $\mathcal{A}$ together with simulated ZK-AOKs. Hyb_1 is computationally indistinguishable from Hyb_0 because of the hiding property of the commitment scheme and zero-knowledge property of the ZK-AOKs.

Hyb_2 Same as Hyb_1 except that P_1 does the following:

- In Step 2 of the online phase, extract $\mathcal{A}$'s input $\{y_i\}_{i\in[n_2]}$ from the ZK-AOKs.
- In Shuffled DOPRF 1 Round 1, extract k_2 from the ZK-AOK.
- In Shuffled DOPRF 1 Round 2, compute $\sigma_i = F_{k_1+k_2}(x_i)$, sample $r_i \xleftarrow{\$} [q^2 \cdot 2^{\lambda-2}]$, and compute $\mathsf{ct}_{\beta_i} \leftarrow \mathsf{CS\_Enc}_{\mathsf{pk}_2}(r_i), \mathsf{g}_i = \sigma_i^{1/r_i}$ for each $i \in [n_1]$. Update the corresponding simulated ZK-AOKs.
- In Shuffled DOPRF 2 Round 3, compute $\sigma'_i = F_{k_1+k_2}(y_i)$ for all $i \in [n_2]$ and use those with simulated ZK-AOKs.

Hyb_2 and Hyb_1 are computationally indistinguishable based on the zero-knowledge property of the ZK-AOKs sent by P_1 and soundness of the ZK-AOKs sent by $\mathcal{A}$.

Hyb_3 Same as Hyb_2 but in Shuffled DOPRF 2 Round 1, P_1 sends $\mathsf{CS\_Enc}_{\mathsf{pk}_1}(0)$ instead of ct_{k_1} along with a simulated ZK-AOK. Hyb_3 and Hyb_2 are computationally indistinguishable based on the semantic security of ct_{k_1}.

Hyb_4 Same as Hyb_3 except that P_1 computes a random function $F(x)$ in the protocol instead of $F_{k_1+k_2}(x)$ and updates the simulated ZK-AOKs correspondingly. Hyb_4 and Hyb_3 are computationally indistinguishable based on the selective security of the PRF.

Hyb_5 Same as Hyb_4 except that P_1 uses random elements sampled from $\mathbb{G}$ instead of computing $F(x)$. Hyb_5 is statistically identical to Hyb_4.

Hyb_6 Same as Hyb_5 but in Shuffled DOPRF 2 Round 1, P_1 send $\mathsf{CS\_Enc}_{\mathsf{pk}_1}(k_1)$ instead of $\mathsf{CS\_Enc}_{\mathsf{pk}_1}(0)$ with updated simulated ZK-AOK. Hyb_6 and Hyb_5 are computationally indistinguishable based on the semantic security of ct_{k_1}.

Hyb_7 Same as Hyb_6 but in Step 2 of the online phase, P_2 commits to 0 instead of y_i's and simulates the ZK-AOKs. Hyb_7 and Hyb_6 are computationally indistinguishable because of the perfect hiding property of the commitment scheme.

Hyb_8 Same as Hyb_7 except that P_1 does the following:

- In Shuffled DOPRF 1 Round 2, compute $\mathsf{ct}_{v_i} \leftarrow \mathsf{Exp_EG_Enc}_{\mathsf{tpk}}(0)$for all $i \in [n_1]$ and send to $\mathcal{A}$ with updated ZK-AOKs.
- In Step 5 of the online phase, compute the intersection-sum S and send a fresh encryption of S under tpk_2 to $\mathcal{A}$ along with an updated simulated proof.

Hyb_8 and Hyb_7 are computationally indistinguishable by the hiding property of the 2-out-of-2 threshold-encryption scheme together with the zero-knowledge property of the verifiable half-decryption proof.

Hyb_9 Same as Hyb_8 but in Shuffled DOPRF 2 Round 3 Step (e), P_1 sends randomly shuffled σ'_i with updated ZK-AOKs. Hyb_9 is computationally indistinguishable from Hyb_8 because of the hiding property of the ElGamal encryption scheme and zero-knowledge property of the ZK-AOKs.

1. Prover samples $\widetilde{x} \xleftarrow{\$} [N/4 \cdot 2^{2\lambda}]$ and sends $\widetilde{y} = g^{\widetilde{x}} \bmod N^2$ to Verifier.
2. Verifier chooses a random challenge $c \xleftarrow{\$} \{0,1\}^{\lambda}$ and sends to Prover.
3. Prover computes $\widehat{x} = x \cdot c + \widetilde{y}$ and sends to Verifier.
4. Verifier verifies that $g^{\widehat{x}} = y^c \cdot \widetilde{y} \bmod N^2$.

Figure 8: Sigma protocol for $\mathsf{ZK}\left\{\exists x : y = g^x \bmod N^2\right\}$.

Hyb_{10} Same as Hyb_9 except that in Shuffled DOPRF 1 Round 3 Step (e), σ'_i values are replaced with random strings, such that there are exactly $|I|$ repeated values in the two shuffled DOPRF protocols. Hyb_{10} and Hyb_9 are computationally indistinguishable based on the zero-knowledge property of the shuffle proof.

Hyb_{11} $\mathcal{A}$'s view in an ideal-world execution with $\mathcal{S}$. Hyb_{11} and Hyb_{10} are identical.

C Sigma Protocols

In this section we describe all the sigma protocols used in our private intersection-sum protocol. We utilize various batching techniques to reduce the communication cost.

C.1 Step 1 of Offline Setup

In Step 1 of the offline setup of our protocol (see Figure 2), each party generates a Camenisch-Shoup encryption key pair $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{pk} = (N, r, g, y)$ and $\mathsf{sk} = x$. Each party sends pk to the other party along with a zero-knowledge proof that y is correctly formed:

$$\mathsf{ZK}\left\{\exists x : y = g^x \bmod N^2\right\}.$$

The sigma protocol for the zero-knowledge proof is shown in Figure 8. The completeness of the proof is straightforward. Next we prove its soundness and zero-knowledge property.

Soundness. If $y \neq g^x \bmod N^2$ for any x, then for any $\widetilde{y}$, we have $y^c \cdot \widetilde{y} \neq g^{\widehat{x}}$ for any $\widehat{x}$ with overwhelming probability and therefore the proof can only pass with negligible probability.

Zero-knowledge. We prove this protocol is honest-verifier zero-knowledge by constructing a PPT simulator that does the following. First it samples $c \xleftarrow{\$} \{0,1\}^{\lambda}$ and $\widehat{x} \xleftarrow{\$} [N/4 \cdot 2^{2\lambda}]$, and then computes $\widetilde{y} = g^{\widehat{x}}/y^c$. Finally it outputs the transcript $(\widetilde{y}, c, \widehat{x})$. The simulated transcript is statistically identical to the real protocol except when $x \cdot c + \widetilde{x} > N/4 \cdot 2^{2\lambda}$, which happens with negligible probability.

C.2 Step 2 of Offline Setup

In Step 2 of the offline setup of our protocol (see Figure 2), each party generates Pedersen commitment parameters $(\mathfrak{g}, \mathfrak{h})$ for the large subgroup of $\mathbb{Z}_N^*$ and prove in zero-knowledge that $\mathfrak{g} \in \langle \mathfrak{h} \rangle$.

1. Prover samples $\widetilde{r} \xleftarrow{\$} [N \cdot 2^{2\lambda}]$ and sends $\widetilde{\mathfrak{g}} = \mathfrak{h}^{\widetilde{r}}$ to Verifier.
2. Verifier chooses a random challenge $c \xleftarrow{\$} \{0,1\}^{\lambda}$ and sends to Prover.
3. Prover computes $\widehat{r} = r \cdot c + \widetilde{r}$ and sends to Verifier.
4. Verifier verifies that $\mathfrak{h}^{\widehat{r}} = \mathfrak{g}^{c} \cdot \widetilde{\mathfrak{g}}$.

Figure 9: Sigma protocol for $\mathsf{ZK}\{\exists r : \mathfrak{g} = (\mathfrak{h})^{r}\}$.

Looking ahead, to enable batching Pedersen commitments via vector commitment, in the actual protocol each party generates parameters $(\mathfrak{g}_1, \mathfrak{g}_2, \ldots, \mathfrak{g}_t, \mathfrak{h})$ for some t and prove in zero-knowledge that $\mathfrak{g}_i \in \langle \mathfrak{h} \rangle$ for each $i \in [t]$.

The sigma protocol for the zero-knowledge proof $\mathsf{ZK}\{\exists r : \mathfrak{g} = (\mathfrak{h})^{r}\}$ is shown in Figure 9. The completeness of the proof is straightforward. Next we prove its soundness and zero-knowledge property.

Soundness. If $\mathfrak{g} \notin \langle \mathfrak{h} \rangle$, then for any $\widetilde{\mathfrak{g}}$, we have $\mathfrak{g}^{c} \cdot \widetilde{\mathfrak{g}} \notin \langle \mathfrak{h} \rangle$ with overwhelming probability and therefore there does not exist $\widehat{r}$ such that $\mathfrak{h}^{\widehat{r}} = \mathfrak{g}^{c} \cdot \widetilde{\mathfrak{g}}$. Hence the proof can only pass with negligible probability.

Zero-knowledge. We prove this protocol is honest-verifier zero-knowledge by constructing a PPT simulator that does the following. First it samples $c \xleftarrow{\$} \{0,1\}^{\lambda}$ and $\widehat{r} \xleftarrow{\$} [N \cdot 2^{2\lambda}]$, and then computes $\widetilde{\mathfrak{g}} = \mathfrak{h}^{\widehat{r}} / \mathfrak{g}^{c}$. Finally it outputs the transcript $(\widetilde{\mathfrak{g}}, c, \widehat{r})$. The simulated transcript is statistically identical to the real protocol except when $r \cdot c + \widetilde{r} > N \cdot 2^{2\lambda}$, which happens with negligible probability.

C.3 Step 3 of Offline Setup

In Step 3 of the offline setup of our protocol (see Figure 2), each party generates $(\mathsf{tpk}, \mathsf{tsk}) \leftarrow \mathsf{EG\_Gen}(1^{\lambda})$ for the 2-out-of-2 threshold encryption scheme on the group $\mathbb{G}$ with order q and generator $\widetilde{g}$ and sends tpk to the other party along with a ZK-AOK of tsk:

$$\mathsf{ZK\text{-}AoK}\{\mathsf{tsk} : \mathsf{tpk} = (\widetilde{g})^{\mathsf{tsk}}\}.$$

The sigma protocol for the ZK-AOK is shown in Figure 10. The completeness of the proof can be easily verified. Next we prove its soundness and zero-knowledge property.

Soundness. We construct a PPT extractor that interacts with a cheating prover and extracts a valid witness tsk. The extractor first executes the protocol honestly with the prover and obtains a transcript $(\widetilde{\mathsf{tpk}}, c_1, \widehat{\mathsf{tsk}}_1)$ such that $(\widetilde{g})^{\widehat{\mathsf{tsk}}_1} = \mathsf{tpk}^{c_1} \cdot \widetilde{\mathsf{tpk}}$. Now the extractor rewinds the protocol to Step 2 and sends a different random challenge c_2 and obtains $(\widetilde{\mathsf{tpk}}, c_2, \widehat{\mathsf{tsk}}_2)$ such that $(\widetilde{g})^{\widehat{\mathsf{tsk}}_2} = \mathsf{tpk}^{c_2} \cdot \widetilde{\mathsf{tpk}}$. Combining the two equations, the extractor gets $(\widetilde{g})^{\Delta\widehat{\mathsf{tsk}}} = \mathsf{tpk}^{\Delta c}$ where $\Delta\widehat{\mathsf{tsk}} = \widehat{\mathsf{tsk}}_1 - \widehat{\mathsf{tsk}}_2$ and $\Delta c = c_1 - c_2$. Now the extractor can compute $\mathsf{tsk} = \Delta\widehat{\mathsf{tsk}} \cdot (\Delta c)^{-1} \mod q$.

1. Prover samples $\widetilde{\mathsf{tsk}} \xleftarrow{\$} [q]$ and sends $\widetilde{\mathsf{tpk}} = (\widetilde{g})^{\widetilde{\mathsf{tsk}}}$ to Verifier.
2. Verifier chooses a random challenge $c \xleftarrow{\$} \{0,1\}^\lambda$ and sends to Prover.
3. Prover computes $\widehat{\mathsf{tsk}} = \mathsf{tsk} \cdot c + \widetilde{\mathsf{tsk}} \mod q$ and sends to Verifier.
4. Verifier verifies that $(\widetilde{g})^{\widehat{\mathsf{tsk}}} = \mathsf{tpk}^c \cdot \widetilde{\mathsf{tpk}}$.

Figure 10: Sigma protocol for $\mathsf{ZK\text{-}AoK}\{\mathsf{tsk} : \mathsf{tpk} = (\widetilde{g})^{\mathsf{tsk}}\}$.

Zero-knowledge. We prove this protocol is honest-verifier zero-knowledge by constructing a PPT simulator that does the following. First it samples $c \xleftarrow{\$} \{0,1\}^\lambda$ and $\widehat{\mathsf{tsk}} \xleftarrow{\$} [q]$, and then computes $\widetilde{\mathsf{tpk}} = (\widetilde{g})^{\widehat{\mathsf{tsk}}}/\mathsf{tpk}^c$. Finally it outputs the transcript $(\widetilde{\mathsf{tpk}}, c, \widehat{\mathsf{tsk}})$. The simulated transcript is statistically identical to the real protocol.

C.4 Step 2 of Online Phase

In Step 2 of the online phase of our protocol (see Figure 3), each party commits to their input elements $\{x_i\}_{i\in[n]}$ on the other party's Pedersen commitment parameters $(\mathfrak{g}, \mathfrak{h})$ on the big subgroup of $\mathbb{Z}_N^*$ and gives a ZK-AOK:

$$\mathsf{ZK\text{-}AoK}\left\{(x_i, r_i) : \mathsf{C}_{x_i} = \mathfrak{g}^{x_i} \cdot \mathfrak{h}^{r_i}\right\}.$$

Recall that in order to enable batching Pedersen commitments later in the protocol, the Pedersen commitment parameters are in fact $(\mathfrak{g}_1, \ldots, \mathfrak{g}_t, \mathfrak{h})$. In order to enable batching sigma protocols for batched commitments later in Round 2 of the shuffled DOPRF protocol (see Figure 4 and Appendix C.6), we need to modify the protocol in this step.

Specifically, each party chooses a random $a_i \xleftarrow{\$} [q]$ and computes $\alpha_i = a_i \cdot (k + x_i) \mod q$ for each $i \in [n]$, where k is this party's PRF key share. Then it commits to k, a_i, and α_i, and proves knowledge of these values to the other party. This change does not affect our security proof, because the simulator can still extract k, a_i, α_i and deduce x_i from those values. Later in Round 2 of the shuffled DOPRF protocol, this party will use the commitments of a_i and α_i in this step for further computation (see Appendix C.6 for more details).

All the above commitments are batched in this step. In more detail, let $\ell = n/t$, then the commitments of $\{a_i\}_{i\in[n]}$ are computed and batched as follows:

$$\begin{aligned}
\mathsf{C}_{a:1} &= (\mathfrak{g}_1)^{a_1} \cdot (\mathfrak{g}_2)^{a_{\ell+1}} \cdots (\mathfrak{g}_t)^{a_{(t-1)\ell+1}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:1}} \\
&\vdots \\
\mathsf{C}_{a:s} &= (\mathfrak{g}_1)^{a_s} \cdot (\mathfrak{g}_2)^{a_{\ell+s}} \cdots (\mathfrak{g}_t)^{a_{(t-1)\ell+i}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:s}} = \prod_{\mathrm{i}=1}^{t} (\mathfrak{g}_\mathrm{i})^{a_{(\mathrm{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:s}} \\
&\vdots \\
\mathsf{C}_{a:\ell} &= (\mathfrak{g}_1)^{a_\ell} \cdot (\mathfrak{g}_2)^{a_{2\ell}} \cdots (\mathfrak{g}_t)^{a_{t\ell}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:\ell}}
\end{aligned}$$

Similarly, the commitments of $\{\alpha_i\}_{i\in[n]}$ are computed and batched as follows:

$$\mathsf{C}_{\alpha:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\alpha_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\alpha:s}} \qquad \forall s \in [\ell]$$

The PRF key share k is committed to $(\mathfrak{g}_1, \mathfrak{h})$, namely $\mathsf{C}_k = (\mathfrak{g}_1)^k \cdot \mathfrak{h}^{r_{\mathsf{C}:k}}$.
Now we prove the following ZK-AOK:

$$\begin{aligned}
\mathsf{ZK\text{-}AoK}\Big\{ \big(\{a_i, \alpha_i\}_{i\in[n]}, \{r_{\mathsf{C}:a:s}, r_{\mathsf{C}:\alpha:s}\}_{s\in[\ell]}, k, r_{\mathsf{C}:k}\big) : \\
\mathsf{C}_{a:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{a_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:s}} \quad \forall s \in [\ell] \wedge \\
\mathsf{C}_{\alpha:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\alpha_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\alpha:s}} \quad \forall s \in [\ell] \wedge \\
\mathsf{C}_k = (\mathfrak{g}_1)^k \cdot \mathfrak{h}^{r_{\mathsf{C}:k}} \Big\}.
\end{aligned} \tag{1}$$

The batched sigma protocol for the above in Figure 11. The completeness of the protocol can be checked. We prove its soundness and zero-knowledge property.

Soundness. We construct a PPT extractor that interacts with a cheating prover and extracts a valid witness $\big(\{a_i, \alpha_i\}_{i\in[n]}, \{r_{\mathsf{C}:a:s}, r_{\mathsf{C}:\alpha:s}\}_{s\in[\ell]}, k, r_{\mathsf{C}:k}\big)$. First we describe how to extract k and $r_{\mathsf{C}:k}$. The extractor first executes the protocol honestly with the prover and obtains a transcript that contains $(\widetilde{\mathsf{C}}_k, c_1, \widehat{k}, \widehat{r}_{\mathsf{C}:k})$ satisfying $(\mathfrak{g}_1)^{\widehat{k}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:k}} = (\mathsf{C}_k)^{c_1} \cdot \widetilde{\mathsf{C}}_k$. Now the extractor rewinds the protocol to Step 2 and sends a challenge containing a different c_1' and obtains $(\widehat{k}', \widehat{r}'_{\mathsf{C}:k})$ such that $(\mathfrak{g}_1)^{\widehat{k}'} \cdot \mathfrak{h}^{\widehat{r}'_{\mathsf{C}:k}} = (\mathsf{C}_k)^{c_1'} \cdot \widetilde{\mathsf{C}}_k$. Combining these two equations, the extractor gets $(\mathfrak{g}_1)^{\Delta\widehat{k}} \cdot \mathfrak{h}^{\Delta\widehat{r}_{\mathsf{C}:k}} = (\mathsf{C}_k)^{\Delta c_1}$. By the strong RSA assumption on $\mathbb{Z}_N^*$, we have $\Delta c_1 | \Delta\widehat{k}$ and $\Delta c_1 | \Delta\widehat{r}_{\mathsf{C}:k}$. Hence the extractor can compute $k = \frac{\Delta\widehat{k}}{\Delta c_1}$ and $r_{\mathsf{C}:k} = \frac{\Delta\widehat{r}_{\mathsf{C}:k}}{\Delta c_1}$.

To extract $\{a_{(\mathfrak{i}-1)\ell+1}\}_{\mathfrak{i}\in[t]}$ and $r_{\mathsf{C}:a:1}$, The extractor first executes the protocol honestly with the prover and obtains a transcript that contains $(\widetilde{\mathsf{C}}_a, \{c_s\}_{s\in[\ell]}, \{\widehat{a}_\mathfrak{i}\}_{\mathfrak{i}\in[t]}, \widehat{r}_{\mathsf{C}:a})$ satisfying $\prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\widehat{a}_\mathfrak{i}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:a}} = \prod_{s=1}^{\ell} (\mathsf{C}_{a:s})^{c_s} \cdot \widetilde{\mathsf{C}}_a$ Next the extractor rewinds the protocol to Step 2 and sends a challenge containing a different c_1' and obtains $(\{\widehat{a}'_\mathfrak{i}\}_{\mathfrak{i}\in[t]}, \widehat{r}'_{\mathsf{C}:a})$ satisfying $\prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\widehat{a}'_\mathfrak{i}} \cdot \mathfrak{h}^{\widehat{r}'_{\mathsf{C}:a}} = \prod_{s=1}^{\ell} (\mathsf{C}_{a:s})^{c'_s} \cdot \widetilde{\mathsf{C}}_a$. Combining these two equations, the extractor gets $\prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\Delta\widehat{a}_\mathfrak{i}} \cdot \mathfrak{h}^{\Delta\widehat{r}_{\mathsf{C}:a}} = (\mathsf{C}_{a:1})^{\Delta c_1}$. By the extended strong RSA assumption on $\mathbb{Z}_N^*$, we have $\Delta c_1 | \Delta\widehat{a}_\mathfrak{i}$ for all $\mathfrak{i} \in [t]$ and $\Delta c_1 | \Delta\widehat{r}_{\mathsf{C}:a}$. Hence the extractor can compute $a_{(\mathfrak{i}-1)\ell+1} = \frac{\Delta\widehat{a}_\mathfrak{i}}{\Delta c_1}$ and $r_{\mathsf{C}:a:1} = \frac{\Delta\widehat{r}_{\mathsf{C}:a}}{\Delta c_1}$.

Similarly the extractor can extract all the $\big(\{a_i, \alpha_i\}_{i\in[n]}, \{r_{\mathsf{C}:a:s}, r_{\mathsf{C}:\alpha:s}\}_{s\in[\ell]}\big)$.

Zero-knowledge. We prove this protocol is honest-verifier zero-knowledge by constructing a PPT simulator that does the following. First it samples $c_s \xleftarrow{\$} \{0,1\}^\lambda$ for all $s \in [\ell]$. Then it samples $\widehat{a}_\mathfrak{i}, \widehat{\alpha}_\mathfrak{i} \xleftarrow{\$} [q \cdot \ell \cdot 2^{2\lambda}]$ for all $\mathfrak{i} \in [t]$ and samples $\widehat{r}_{\mathsf{C}:a}, \widehat{r}_{\mathsf{C}:\alpha} \xleftarrow{\$} [N \cdot \ell \cdot 2^{2\lambda}]$, $\widehat{k} \xleftarrow{\$} [q \cdot 2^{2\lambda}]$, and $\widehat{r}_{\mathsf{C}:k} \xleftarrow{\$} [N \cdot 2^{2\lambda}]$. Now the simulator can compute $\widetilde{\mathsf{C}}_a = \frac{\prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\widehat{a}_\mathfrak{i}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:a}}}{\prod_{s=1}^{\ell} (\mathsf{C}_{a:s})^{c_s}}$, $\widetilde{\mathsf{C}}_\alpha = \frac{\prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\widehat{\alpha}_\mathfrak{i}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:\alpha}}}{\prod_{s=1}^{\ell} (\mathsf{C}_{\alpha:s})^{c_s}}$, and $\widetilde{\mathsf{C}}_k = \frac{(\mathfrak{g}_1)^{\widehat{k}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:k}}}{(\mathsf{C}_k)^{c_1}}$. Finally it outputs the simulated transcript, which is statistically identical to the real protocol except negligible probability.

1. Prover does the following:

   (a) Sample $\widetilde{a}_{\mathfrak{i}}, \widetilde{\alpha}_{\mathfrak{i}} \xleftarrow{\$} [q \cdot \ell \cdot 2^{2\lambda}]$ for all $\mathfrak{i} \in [t]$. Sample $\widetilde{r}_{\mathsf{C}:a}, \widetilde{r}_{\mathsf{C}:\alpha} \xleftarrow{\$} [N \cdot \ell \cdot 2^{2\lambda}]$, $\widetilde{k} \xleftarrow{\$} [q \cdot 2^{2\lambda}]$, and $\widetilde{r}_{\mathsf{C}:k} \xleftarrow{\$} [N \cdot 2^{2\lambda}]$.

   (b) Compute $\widetilde{\mathsf{C}}_a = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_i)^{a_{\mathfrak{i}}} \cdot \mathfrak{h}^{\widetilde{r}_{\mathsf{C}:a}}$, $\widetilde{\mathsf{C}}_\alpha = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_i)^{\alpha_{\mathfrak{i}}} \cdot \mathfrak{h}^{\widetilde{r}_{\mathsf{C}:\alpha}}$, and $\widetilde{\mathsf{C}}_k = (\mathfrak{g}_1)^{\widetilde{k}} \cdot \mathfrak{h}^{r_{\mathsf{C}:k}}$.

   (c) Send $(\widetilde{\mathsf{C}}_a, \widetilde{\mathsf{C}}_\alpha, \widetilde{\mathsf{C}}_k)$ to Verifier.

2. Verifier chooses random challenges $c_s \xleftarrow{\$} \{0,1\}^\lambda$ for all $s \in [\ell]$ and sends $\{c_s\}_{s \in [\ell]}$ to Prover.

3. Prover computes the following and sends to Verifier:

$$\widehat{a}_{\mathfrak{i}} = \sum_{s=1}^{\ell} \left(c_s \cdot a_{(\mathfrak{i}-1)\ell+s}\right) + \widetilde{a}_{\mathfrak{i}} \quad \forall \mathfrak{i} \in [t]; \qquad \widehat{r}_{\mathsf{C}:a} = \sum_{s=1}^{\ell} (c_s \cdot r_{\mathsf{C}:a:s}) + \widetilde{r}_{\mathsf{C}:a};$$

$$\widehat{\alpha}_{\mathfrak{i}} = \sum_{s=1}^{\ell} \left(c_s \cdot \alpha_{(\mathfrak{i}-1)\ell+s}\right) + \widetilde{\alpha}_{\mathfrak{i}} \quad \forall \mathfrak{i} \in [t]; \qquad \widehat{r}_{\mathsf{C}:\alpha} = \sum_{s=1}^{\ell} (c_s \cdot r_{\mathsf{C}:\alpha:s}) + \widetilde{r}_{\mathsf{C}:\alpha};$$

$$\widehat{k} = c_1 \cdot k + \widetilde{k}; \qquad \widehat{r}_{\mathsf{C}:k} = c_1 \cdot r_{\mathsf{C}:k} + \widetilde{r}_{\mathsf{C}:k}.$$

4. Verifier verifies the following:

$$\prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{\widehat{a}_{\mathfrak{i}}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:a}} = \prod_{s=1}^{\ell} (\mathsf{C}_{a:s})^{c_s} \cdot \widetilde{\mathsf{C}}_a; \qquad \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{\widehat{\alpha}_{\mathfrak{i}}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:\alpha}} = \prod_{s=1}^{\ell} (\mathsf{C}_{\alpha:s})^{c_s} \cdot \widetilde{\mathsf{C}}_\alpha$$

$$(\mathfrak{g}_1)^{\widehat{k}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:k}} = (\mathsf{C}_k)^{c_1} \cdot \widetilde{\mathsf{C}}_k.$$

Figure 11: Batched sigma protocol for ZK-AOK (1).

C.5 Round 1 of Shuffled DOPRF

In Round 1 of the shuffled DOPRF protocol where P_1 holds the input (see Figure 4), party P_2 computes $\mathsf{ct}_k \leftarrow \mathsf{CS\_Enc}_{\mathsf{pk}}(k)$ and $\mathsf{C}_k \leftarrow \mathsf{com}_{\mathfrak{g},\mathfrak{h}}(k)$, where k is P_2's PRF key share, pk is P_2's Camenisch-Shoup public key on the big subgroup of $\mathbb{Z}^*_{N_2}$, and $(\mathfrak{g}, \mathfrak{h})$ are P_1's Pedersen commitment parameters on the big subgroup of $\mathbb{Z}^*_{N_1}$. Recall that $\mathsf{pk} = (N_2, g, y)$. P_2 sends $\mathsf{ct}_k = (u, e)$ and C_k to P_1 along with a ZK-AOK

$$\mathsf{ZK\text{-}AoK}\Big\{(k, r_1, r_2) : u = g^{r_1} \ \wedge \ e = (1+N_2)^k \cdot y_2^{r_1} \ \wedge \\ \mathsf{C}_k = \mathfrak{g}^k \cdot \mathfrak{h}^{r_2} \ \wedge \ k \leq q \cdot 2^{2\lambda+1}\Big\}.$$

In fact, to enable batching commitments, P_1's Pedersen commitment parameters are $(\mathfrak{g}_1, \ldots, \mathfrak{g}_t, \mathfrak{h})$. To enable batching the first component of the Camenisch-Shoup ciphertexts, P_2's Camenisch-Shoup public key is in fact $\mathsf{pk} = (N_2, g, y_1, y_2, \ldots, y_t)$. In order to enable batching sigma protocols for batched commitments and Camenisch-Shoup encryptions later in Round 2 of the shuffled DOPRF protocol (see Figure 4 and Appendix C.6), we need to modify the protocol in this step.

Recall that k was committed as $\mathsf{C}_k = (\mathfrak{g}_1)^k \cdot \mathfrak{h}^{r_{\mathsf{C}:k}}$ in the modified Step 2 of the online phase of our protocol (see Appendix C.4).

To enable batching in the first component of Camenisch-Shoup encryption, in this step P_2 encrypts t copies of k, where the $\mathfrak{i}$-th copy of k is encrypted in the $\mathfrak{i}$-th slot and the other slots are all 0. In more detail, P_2 computes $\mathsf{ct}_{k:\mathfrak{i}} = (u_\mathfrak{i}, e_{\mathfrak{i},1}, \ldots, e_{\mathfrak{i},t})$ for each $\mathfrak{i} \in [t]$, where $u_\mathfrak{i} = g^{r_\mathfrak{i}}$, $e_{\mathfrak{i},\mathfrak{i}} = (1+N_2)^k \cdot (y_\mathfrak{i})^{r_\mathfrak{i}}$, and $e_{\mathfrak{i},\mathfrak{j}} = (y_\mathfrak{j})^{r_\mathfrak{i}}$ for all $\mathfrak{j} \in [t] \setminus \{\mathfrak{i}\}$.

Now we need to prove the following ZK-AOK:

$$\begin{aligned}\mathsf{ZK\text{-}AoK}\Big\{ \big(k, r_{\mathsf{C}:k}, \{r_\mathfrak{i}\}_{\mathfrak{i}\in[t]}\big) : \\ \forall \mathfrak{i} \in [t], \quad u_\mathfrak{i} = g^{r_\mathfrak{i}}, \quad e_{\mathfrak{i},\mathfrak{i}} = (1+N_2)^k \cdot (y_\mathfrak{i})^{r_\mathfrak{i}}, \quad e_{\mathfrak{i},\mathfrak{j}} = (y_\mathfrak{j})^{r_\mathfrak{i}} \quad \forall \mathfrak{j} \in [t] \setminus \{\mathfrak{i}\} \\ \mathsf{C}_k = (\mathfrak{g}_1)^k \cdot \mathfrak{h}^{r_{\mathsf{C}:k}} \ \wedge \ k \le q \cdot 2^{2\lambda+1} \Big\}.\end{aligned} \tag{2}$$

1. Prover does the following:
   (a) Sample $\widetilde{k} \xleftarrow{\$} [q \cdot 2^{2\lambda}]$, $\widetilde{r}_{\mathsf{C}:k} \xleftarrow{\$} [N_1 \cdot 2^{2\lambda}]$, and $\widetilde{r}_\mathfrak{i} \xleftarrow{\$} [N_2 \cdot 2^{2\lambda}]$ for all $\mathfrak{i} \in [t]$.
   (b) Compute $\widetilde{\mathsf{C}}_k = (\mathfrak{g}_1)^{\widetilde{k}} \cdot \mathfrak{h}^{r_{\mathsf{C}:k}}$.
   (c) For each $\mathfrak{i} \in [t]$, compute $\widetilde{\mathsf{ct}}_{k:\mathfrak{i}} = (\widetilde{u}_\mathfrak{i}, \widetilde{e}_{\mathfrak{i},1}, \ldots, \widetilde{e}_{\mathfrak{i},t})$, where $\widetilde{u}_\mathfrak{i} = g^{\widetilde{r}_\mathfrak{i}}$, $\widetilde{e}_{\mathfrak{i},\mathfrak{i}} = (1+N_2)^{\widetilde{k}} \cdot (y_\mathfrak{i})^{\widetilde{r}_\mathfrak{i}}$, and $\widetilde{e}_{\mathfrak{i},\mathfrak{j}} = (y_\mathfrak{j})^{\widetilde{r}_\mathfrak{i}}$ for all $\mathfrak{j} \in [t] \setminus \{\mathfrak{i}\}$.
   (d) Send $\Big(\widetilde{\mathsf{C}}_k, \{\widetilde{\mathsf{ct}}_{k:\mathfrak{i}}\}_{\mathfrak{i}\in[t]}\Big)$ to Verifier.
2. Verifier chooses a random challenge $c \xleftarrow{\$} \{0,1\}^\lambda$ and sends to Prover.
3. Prover computes $\widehat{k} = c \cdot k + \widetilde{k}$, $\widehat{r}_{\mathsf{C}:k} = c \cdot r_{\mathsf{C}:k} + \widetilde{r}_{\mathsf{C}:k}$, and $\widehat{r}_\mathfrak{i} = c \cdot r_\mathfrak{i} + \widetilde{r}_\mathfrak{i}$ for all $\mathfrak{i} \in [t]$, and sends to Verifier.
4. Verifier verifies the following:

$$\begin{aligned}
&g^{\widehat{r}_\mathfrak{i}} = (u_\mathfrak{i})^c \cdot \widetilde{u}_\mathfrak{i} \qquad \forall \mathfrak{i} \in [t]; \\
&(1+N_2)^{\widehat{k}} \cdot (y_\mathfrak{i})^{\widehat{r}_\mathfrak{i}} = (e_{\mathfrak{i},\mathfrak{i}})^c \cdot \widetilde{e}_{\mathfrak{i},\mathfrak{i}} \qquad \forall \mathfrak{i} \in [t]; \\
&(y_\mathfrak{j})^{\widehat{r}_\mathfrak{i}} = (e_{\mathfrak{i},\mathfrak{j}})^c \cdot \widetilde{e}_{\mathfrak{i},\mathfrak{j}} \qquad \forall \mathfrak{i} \in [t], \mathfrak{j} \in [t] \setminus \{\mathfrak{i}\}; \\
&(\mathfrak{g}_1)^{\widehat{k}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:k}} = (\mathsf{C}_k)^{c_1} \cdot \widetilde{\mathsf{C}}_k; \\
&\widehat{k} \le q \cdot 2^{2\lambda}
\end{aligned}$$

Figure 12: Sigma protocol for the ZK-AOK (2).

The sigma protocol is shown in Figure 12. The completeness of the protocol can be naturally verified. The only subtlety is that $\widehat{k} > q \cdot 2^{2\lambda}$ with negligible probability, hence the protocol is complete with all but negligible probability. Next we prove its soundness and zero-knowledge property.

Soundness. We construct a PPT extractor that interacts with a cheating prover and extracts a valid witness $\big(k, r_{\mathsf{C}:k}, \{r_\mathfrak{i}\}_{\mathfrak{i}\in[t]}\big)$. We focus on the case to k and $r_{\mathsf{C}:k}$ and the other extractions are similar. The extraction of k is same as in Appendix C.4. In the extraction, since $\Delta k < q \cdot 2^{2\lambda+1}$,

we conclude that $k \leq q \cdot 2^{2\lambda+1}$. In addition, the extracted value k in the commitments is equal to the extracted k from the Camenisch-Shoup encryptions. Since N_2 is sufficiently large, this follows from the fact that $k = \frac{\Delta k}{\Delta c}$ is a valid solution in all the extractions.

Zero-knowledge. Similarly as the previous sigma protocols, we prove this protocol is honest-verifier zero-knowledge by constructing a PPT simulator that samples $c \xleftarrow{\$} \{0,1\}^\lambda$ and $\widehat{k} \xleftarrow{\$} [q \cdot 2^{2\lambda}]$, $\widehat{r}_{\mathsf{C}:k} \xleftarrow{\$} [N_1 \cdot 2^{2\lambda}]$, and $\widehat{r}_{\mathfrak{i}} \xleftarrow{\$} [N_2 \cdot 2^{2\lambda}]$ for all $\mathfrak{i} \in [t]$, and then computes the corresponding $\left(\widetilde{\mathsf{C}}_k, \{\widetilde{\mathsf{ct}}_{k:\mathfrak{i}}\}_{\mathfrak{i}\in[t]}\right)$ and outputs the simulated transcript. The simulated transcript is statistically indistinguishable to the real protocol.

C.6 Round 2 of Shuffled DOPRF

In Round 2 of the shuffled DOPRF protocol where P_1 holds the input (see Figure 4), for each $i \in [n_1]$, party P_1 computes $\mathsf{g}_i = g^{a_i}$ for a random $a_i \xleftarrow{\$} [q]$ where g is the generator of the $\mathbb{G}$ with order q. P_1 also computes $\mathsf{C}_{a_i} \leftarrow \mathsf{com}_{\mathfrak{g},\mathfrak{h}}(a_i)$, $\mathsf{C}_{b_i} \leftarrow \mathsf{com}_{\mathfrak{g},\mathfrak{h}}(b_i)$ for a random $b_i \xleftarrow{\$} [q \cdot 2^\lambda]$, and $\mathsf{C}_{\alpha_i} = \mathsf{com}_{\mathfrak{g},\mathfrak{h}}(\alpha_i)$ for $\alpha_i = a_i \cdot (k_1 + x_i)$, where $(\mathfrak{g}, \mathfrak{h})$ are P_2's Pedersen commitment parameters on $\mathbb{Z}^*_{N_2}$. In addition it computes $\mathsf{ct}_{\beta_i} \leftarrow (\mathsf{ct}_{k_2})^{a_i} \cdot \mathsf{CS\_Enc}_{\mathsf{pk}_2}(\alpha_i) \cdot (\mathsf{CS\_Enc}_{\mathsf{pk}}(b_i))^q$ for $\beta_i = a_i \cdot (k_1 + k_2 + x_i) + b_i \cdot q = a_i \cdot k_2 + \alpha_i + b_i \cdot q$, where pk is P_2's Camenisch-Shoup public key on $\mathbb{Z}^*_{N_2}$. Note that C_{x_i} was sent by P_1 in Step 2 of the online phase, and C_{k_1} was sent by P_1 in Round 1 of the other shuffled DOPRF protocol where P_2 holds the input. In this round P_1 proves the following ZK-AOK:

$$
\begin{aligned}
&\mathsf{ZK\text{-}AoK}\big\{(a_i, b_i, \alpha_i, r_1, r_2, r_3, r_4, r_5, r_6) : \\
&\qquad \mathsf{C}_{a_i} = (\mathfrak{g}_2)^{a_i} \cdot (\mathfrak{h}_2)^{r_1} \ \wedge\ a_i \leq q \cdot 2^{2\lambda+1} \ \wedge \\
&\qquad \mathsf{C}_{b_i} = (\mathfrak{g}_2)^{b_i} \cdot (\mathfrak{h}_2)^{r_2} \ \wedge\ b_i \leq q \cdot 2^{3\lambda+1} \ \wedge \\
&\qquad \mathsf{C}_{\alpha_i} = (\mathfrak{g}_2)^{\alpha_i} \cdot (\mathfrak{h}_2)^{r_3} \ \wedge\ \mathsf{C}_{\alpha_i} = (\mathsf{C}_{k_1} \cdot \mathsf{C}_{x_i})^{a_i} \cdot (\mathfrak{h}_2)^{r_4} \ \wedge\ \alpha_i \leq q \cdot 2^{2\lambda+1} \ \wedge \\
&\qquad \mathsf{ct}_{\beta_i} = (\mathsf{ct}_{k_2})^{a_i} \cdot \mathsf{CS\_Enc}_{\mathsf{pk}_2}(\alpha_i; r_5) \cdot (\mathsf{CS\_Enc}_{\mathsf{pk}_2}(b_i; r_6))^q \wedge \\
&\qquad \mathsf{g}_i = g^{a_i}\big\}.
\end{aligned}
$$

Recall that in order to enable batching, P_2's Pedersen commitment parameters are in fact $(\mathfrak{g}_1, \ldots, \mathfrak{g}_t, \mathfrak{h})$, and P_2's Camenisch-Shoup public key is in fact $\mathsf{pk} = (N_2, g_2, y_1, y_2, \ldots, y_t)$. We discussed in Section 4 that in order to enable batching sigma protocols for batched commitments and batched encryptions in this step, we need to modify our main protocol to make non-black-box use of the DOPRF protocol. Specifically, we have modified Step 2 of the online phase of our protocol (see Appendix C.4) and Round 1 of our shuffled DOPRF protocol (see Appendix C.5). Next we elaborate how to batch commitments and encryptions in this round using commitments and encryptions from previous steps and how to batch sigma protocols in this round.

First, recall that the commitments of $\{a_i\}_{i\in[n_1]}$ and $\{\alpha_i\}_{i\in[n_1]}$ are computed in Step 2 of the online phase of our protocol and batched as follows (see Appendix C.4):

$$
\mathsf{C}_{a:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{a_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:s}} \qquad \forall s \in [\ell]
$$

$$
\mathsf{C}_{\alpha:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{\alpha_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\alpha:s}} \qquad \forall s \in [\ell]
$$

The commitments of $\{b_i\}_{i\in[n_1]}$ can be computed and batched similarly in this round:

$$\mathsf{C}_{b:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{b_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:b:s}} \qquad \forall s \in [\ell]$$

Recall that in Round 1 of our shuffled DOPRF protocol (see Appendix C.5), P_2 encrypts t copies of k_2, where the $\mathfrak{i}$-th copy of k_2 is encrypted in the $\mathfrak{i}$-th slot and the other slots are all 0. In more detail, P_2 computes $\mathsf{ct}_{k_2:\mathfrak{i}} = (u_{k_2:\mathfrak{i}}, e_{k_2:\mathfrak{i},1}, \ldots, e_{k_2:\mathfrak{i},t})$ for each $\mathfrak{i} \in [t]$, where $u_{k_2:\mathfrak{i}} = (g_2)^{r_{k_2:\mathfrak{i}}}$, $e_{k_2:\mathfrak{i},\mathfrak{i}} = (1+N_2)^{k_2} \cdot (y_{\mathfrak{i}})^{r_{k_2:\mathfrak{i}}}$, and $e_{k_2:\mathfrak{i},\mathfrak{j}} = (y_{\mathfrak{j}})^{r_{k_2:\mathfrak{i}}}$ for all $\mathfrak{j} \in [t] \setminus \{\mathfrak{i}\}$. In this round, $(\mathsf{ct}_{k_2})^{a_i}$ can be computed and batched as follows:

$$(\mathsf{ct}_{k_2})^{a:1} = \begin{cases} u_{a:1} = (u_{k_2:1})^{a_1} \cdot (u_{k_2:2})^{a_{\ell+1}} \cdots (u_{k_2:t})^{a_{(t-1)\ell+1}} \\ e_{a:1,1} = (e_{k_2:1,1})^{a_1} \cdot (e_{k_2:2,1})^{a_{\ell+1}} \cdots (e_{k_2:t,1})^{a_{(t-1)\ell+1}} \\ \quad \vdots \\ e_{a:1,\mathfrak{i}} = (e_{k_2:1,\mathfrak{i}})^{a_1} \cdot (e_{k_2:2,\mathfrak{i}})^{a_{\ell+1}} \cdots (e_{k_2:t,\mathfrak{i}})^{a_{(t-1)\ell+1}} \\ \quad \vdots \\ e_{a:1,t} = (e_{k_2:1,t})^{a_1} \cdot (e_{k_2:2,t})^{a_{\ell+1}} \cdots (e_{k_2:t,t})^{a_{(t-1)\ell+1}} \end{cases}$$

$$\vdots$$

$$(\mathsf{ct}_{k_2})^{a:s} = \begin{cases} u_{a:s} = (u_{k_2:1})^{a_s} \cdot (u_{k_2:2})^{a_{\ell+s}} \cdots (u_{k_2:t})^{a_{(t-1)\ell+s}} = \prod_{\mathfrak{j}=1}^{t} (u_{k_2:\mathfrak{j}})^{a_{(\mathfrak{j}-1)\ell+s}} \\ e_{a:s,\mathfrak{i}} = (e_{k_2:1,\mathfrak{i}})^{a_s} \cdot (e_{k_2:2,\mathfrak{i}})^{a_{\ell+s}} \cdots (e_{k_2:t,\mathfrak{i}})^{a_{(t-1)\ell+s}} = \prod_{\mathfrak{j}=1}^{t} (e_{k_2:2,\mathfrak{i}})^{a_{(\mathfrak{j}-1)\ell+s}} \quad \forall \mathfrak{i} \in [t] \end{cases}$$

$$\vdots$$

$$(\mathsf{ct}_{k_2})^{a:\ell} = \begin{cases} u_{a:\ell} = (u_{k_2:1})^{a_\ell} \cdot (u_{k_2:2})^{a_{2\ell}} \cdots (u_{k_2:t})^{a_{t\ell}} \\ e_{a:\ell,1} = (e_{k_2:1,1})^{a_\ell} \cdot (e_{k_2:2,1})^{a_{2\ell}} \cdots (e_{k_2:t,1})^{a_{t\ell}} \\ \quad \vdots \\ e_{a:\ell,j} = (e_{k_2:1,j})^{a_\ell} \cdot (e_{k_2:2,j})^{a_{2\ell}} \cdots (e_{k_2:t,j})^{a_{t\ell}} \\ \quad \vdots \\ e_{a:\ell,t} = (e_{k_2:1,t})^{a_\ell} \cdot (e_{k_2:2,t})^{a_{2\ell}} \cdots (e_{k_2:t,t})^{a_{t\ell}} \end{cases}$$

In this round, $\mathsf{CS\_Enc}_{\mathsf{pk}}(\alpha_i)$ can be computed and batched as follows:

$$\mathsf{ct}_{\alpha:1} = \begin{cases} u_{\alpha:1} = (g_2)^{r_{\mathsf{ct}:\alpha:1}} \\ e_{\alpha:1,1} = (1+N_2)^{\alpha_1} \cdot (h_1)^{r_{\mathsf{ct}:\alpha:1}} \\ \quad \vdots \\ e_{\alpha:1,\mathfrak{i}} = (1+N_2)^{\alpha_{(\mathfrak{i}-1)\ell+1}} \cdot (h_{\mathfrak{i}})^{r_{\mathsf{ct}:\alpha:1}} \\ \quad \vdots \\ e_{\alpha:1,t} = (1+N_2)^{\alpha_{(t-1)\ell+1}} \cdot (h_t)^{r_{\mathsf{ct}:\alpha:1}} \end{cases}$$

$$\vdots$$

$$\mathsf{ct}_{\alpha:s} = \begin{cases} u_{\alpha:s} = (g_2)^{r_{\mathsf{ct}:\alpha:s}} \\ e_{\alpha:s,\mathfrak{i}} = (1+N_2)^{\alpha_{(\mathfrak{i}-1)\ell+s}} \cdot (h_{\mathfrak{i}})^{r_{\mathsf{ct}:\alpha:s}} \qquad \forall \mathfrak{i} \in [t] \end{cases}$$

$$\vdots$$

$$\mathsf{ct}_{\alpha:\ell} = \begin{cases} u_{\alpha:\ell} = (g_2)^{r_{\mathsf{ct}:\alpha:\ell}} \\ e_{\alpha:\ell,1} = (1+N_2)^{\alpha_\ell} \cdot (h_1)^{r_{\mathsf{ct}:\alpha:\ell}} \\ \quad\vdots \\ e_{\alpha:\ell,\mathfrak{i}} = (1+N_2)^{\alpha_{\mathfrak{i}\ell}} \cdot (h_{\mathfrak{i}})^{r_{\mathsf{ct}:\alpha:\ell}} \\ \quad\vdots \\ e_{\alpha:\ell,t} = (1+N_2)^{\alpha_{t\ell}} \cdot (h_t)^{r_{\mathsf{ct}:\alpha:\ell}} \end{cases}$$

$(\mathsf{CS\_Enc}_{\mathsf{pk}}(b_i))^q$ can be computed and batched similarly as

$$\forall s \in [\ell], \quad \mathsf{ct}_{b:s} = \begin{cases} u_{b:s} = (g_2)^{r_{\mathsf{ct}:b:s}} \\ e_{b:s,\mathfrak{i}} = ((1+N_2)^q)^{b_{(\mathfrak{i}-1)\ell+s}} \cdot (h_{\mathfrak{i}})^{r_{\mathsf{ct}:b:s}} \qquad \forall \mathfrak{i} \in [t] \end{cases}$$

Since $\beta_i = a_i \cdot (k_1 + k_2 + x_i) + b_i \cdot q = a_i \cdot k_2 + \alpha_i + b_i \cdot q$, $\mathsf{CS\_Enc}_{\mathsf{pk}}(\beta_i)$ can be computed and batched as follows: for $\forall s \in [\ell]$,

$$\mathsf{ct}_{\beta:s} = \begin{cases} u_{\beta:s} = \left(\prod_{\mathfrak{j}=1}^{t} (u_{k_2:\mathfrak{j}})^{a_{(\mathfrak{j}-1)\ell+s}}\right) \cdot (g_2)^{r_{\mathsf{ct}:\beta:s}} \\ e_{\beta:s,\mathfrak{i}} = \left(\prod_{\mathfrak{j}=1}^{t} (e_{k_2:\mathfrak{j},\mathfrak{i}})^{a_{(\mathfrak{j}-1)\ell+s}}\right) \cdot (1+N_2)^{\alpha_{(\mathfrak{i}-1)\ell+s} + q \cdot b_{(\mathfrak{i}-1)\ell+s}} \cdot (h_{\mathfrak{i}})^{r_{\mathsf{ct}:\beta:s}} \quad \forall \mathfrak{i} \in [t] \end{cases}$$

where $r_{\mathsf{ct}:\beta:s} = r_{\mathsf{ct}:\alpha:s} + r_{\mathsf{ct}:b:s}$

Now the prover proves the following ZK-AOK:

$$\begin{aligned} \mathsf{ZK\text{-}AoK}\Big\{ \big(\{a_i, b_i, \alpha_i\}_{i\in[n_1]}, \{r_{\mathsf{C}:a:s}, r_{\mathsf{C}:b:s}, r_{\mathsf{C}:\alpha:s}, r_{\mathsf{ct}:\beta:s}\}_{s\in[\ell]}\big) : \\ \mathsf{C}_{a:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{a_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:a:s}} \quad \forall s \in [\ell] \ \wedge \ a_i \le q \cdot \ell \cdot 2^{2\lambda+1} \quad \forall i \in [n_1] \ \wedge \\ \mathsf{C}_{b:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{b_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:b:s}} \quad \forall s \in [\ell] \ \wedge \ b_i \le q \cdot \ell \cdot 2^{3\lambda+1} \quad \forall i \in [n_1] \ \wedge \\ \mathsf{C}_{\alpha:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_{\mathfrak{i}})^{\alpha_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\alpha:s}} \quad \forall s \in [\ell] \ \wedge \ \alpha_i \le q \cdot \ell \cdot 2^{2\lambda+1} \quad \forall i \in [n_1] \ \wedge \\ u_{\beta:s} = \left(\prod_{\mathfrak{j}=1}^{t} (u_{k_2:\mathfrak{j}})^{a_{(\mathfrak{j}-1)\ell+s}}\right) \cdot (g_2)^{r_{\mathsf{ct}:\beta:s}} \quad \forall s \in [\ell] \ \wedge \\ e_{\beta:s,\mathfrak{i}} = \left(\prod_{\mathfrak{j}=1}^{t} (e_{k_2:\mathfrak{j},\mathfrak{i}})^{a_{(\mathfrak{j}-1)\ell+s}}\right) \cdot (1+N_2)^{\alpha_{(\mathfrak{i}-1)\ell+s} + q \cdot b_{(\mathfrak{i}-1)\ell+s}} \cdot (h_{\mathfrak{i}})^{r_{\mathsf{ct}:\beta:s}} \quad \forall s \in [\ell], \mathfrak{i} \in [t] \\ \mathfrak{g}_i = g^{a_i} \quad \forall i \in [n_1] \Big\}. \end{aligned} \tag{3}$$

The batched sigma protocol is presented in Figure 13. The completeness of the protocol holds with all but negligible probability. Next we argue its soundness and zero-knowledge property.

Soundness. The way to extract $\{a_i, b_i, \alpha_i\}_{i\in[n_1]}$ and the corresponding randomness from the protocol is the same as in Appendix C.4. The range proofs for these extracted values can be naturally checked. In addition, the extracted values from the commitments are equal to the extracted values from the encryptions because they can be deduced in the same way.

1. Prover does the following:

   (a) Sample $\widetilde{a}_{\mathsf{i}}, \widetilde{\alpha}_{\mathsf{i}} \xleftarrow{\$} [q \cdot \ell \cdot 2^{2\lambda}]$ and $\widetilde{b}_{\mathsf{i}} \xleftarrow{\$} [q \cdot \ell \cdot 2^{3\lambda}]$ for each $\mathsf{i} \in [t]$ and $\widetilde{r}_{\mathsf{C}:a}, \widetilde{r}_{\mathsf{C}:b}, \widetilde{r}_{\mathsf{C}:\alpha}, \widetilde{r}_{\mathsf{C}:\beta} \xleftarrow{\$} [N_2 \cdot \ell \cdot 2^{2\lambda}]$.

   (b) Compute the following and send to Verifier:

$$\widetilde{\mathsf{C}}_a = \prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widetilde{a}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widetilde{r}_{\mathsf{C}:a}}; \quad \widetilde{\mathsf{C}}_\alpha = \prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widetilde{\alpha}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widetilde{r}_{\mathsf{C}:\alpha}}; \quad \widetilde{\mathsf{C}}_b = \prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widetilde{b}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widetilde{r}_{\mathsf{C}:b}};$$
$$\widetilde{u}_\beta = \left(\prod_{\mathsf{j}=1}^{t} (u_{k_2:\mathsf{j}})^{\widetilde{a}_{\mathsf{j}}} \right) \cdot (g_2)^{\widetilde{r}_{\mathsf{ct}:\beta}};$$
$$\widetilde{e}_{\beta:\mathsf{i}} = \left(\prod_{\mathsf{j}=1}^{t} (e_{k_2:\mathsf{j},\mathsf{i}})^{\widetilde{a}_{\mathsf{j}}} \right) \cdot (1+N_2)^{\widetilde{\alpha}_{\mathsf{i}} + q \cdot \widetilde{b}_{\mathsf{i}}} \cdot (h_{\mathsf{i}})^{\widetilde{r}_{\mathsf{ct}:\beta}} \qquad \forall \mathsf{i} \in [t];$$
$$\widetilde{\mathfrak{g}}_{\mathsf{i}} = g^{\widetilde{a}_{\mathsf{i}}} \qquad \forall \mathsf{i} \in [t].$$

2. Verifier chooses random challenges $c_1, \ldots, c_\ell \xleftarrow{\$} \{0,1\}^\lambda$ and sends to Prover.

3. Prover computes the following and sends to Verifier:

$$\widehat{a}_{\mathsf{i}} = \sum_{s=1}^{\ell} c_s \cdot a_{(\mathsf{i}-1)\ell+s} + \widetilde{a}_{\mathsf{i}} \qquad \forall \mathsf{i} \in [t]; \qquad \widehat{r}_{\mathsf{C}:a} = \sum_{s=1}^{\ell} c_s \cdot r_{\mathsf{C}:a:s} + \widetilde{r}_{\mathsf{C}:a};$$
$$\widehat{b}_{\mathsf{i}} = \sum_{s=1}^{\ell} c_s \cdot b_{(\mathsf{i}-1)\ell+s} + \widetilde{b}_{\mathsf{i}} \qquad \forall \mathsf{i} \in [t]; \qquad \widehat{r}_{\mathsf{C}:b} = \sum_{s=1}^{\ell} c_s \cdot r_{\mathsf{C}:b:s} + \widetilde{r}_{\mathsf{C}:b};$$
$$\widehat{\alpha}_{\mathsf{i}} = \sum_{s=1}^{\ell} c_s \cdot \alpha_{(\mathsf{i}-1)\ell+s} + \widetilde{\alpha}_{\mathsf{i}} \qquad \forall \mathsf{i} \in [t]; \qquad \widehat{r}_{\mathsf{C}:\alpha} = \sum_{s=1}^{\ell} c_s \cdot r_{\mathsf{C}:\alpha:s} + \widetilde{r}_{\mathsf{C}:\alpha};$$
$$\widehat{r}_{\mathsf{C}:\beta} = \sum_{s=1}^{\ell} c_s \cdot r_{\mathsf{C}:\beta:s} + \widetilde{r}_{\mathsf{C}:\beta}.$$

4. Verifier verifies the following:

$$\prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widehat{a}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:a}} = \prod_{s=1}^{\ell} (\mathsf{C}_{a:s})^{c_s} \cdot \widetilde{\mathsf{C}}_a; \qquad \prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widehat{\alpha}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:\alpha}} = \prod_{s=1}^{\ell} (\mathsf{C}_{\alpha:s})^{c_s} \cdot \widetilde{\mathsf{C}}_\alpha;$$
$$\prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widehat{b}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:b}} = \prod_{s=1}^{\ell} (\mathsf{C}_{b:s})^{c_s} \cdot \widetilde{\mathsf{C}}_b; \qquad \prod_{\mathsf{j}=1}^{t} (u_{k_2:\mathsf{j}})^{\widehat{a}_{\mathsf{j}}} \cdot (g_2)^{\widehat{r}_{\mathsf{ct}:\beta}} = \prod_{s=1}^{\ell} (u_{\beta:s})^{c_s} \cdot \widetilde{u}_\beta;$$
$$\prod_{\mathsf{j}=1}^{t} (e_{k_2:\mathsf{j},\mathsf{i}})^{\widehat{a}_{\mathsf{j}}} \cdot (1+N_2)^{\widehat{\alpha}_{\mathsf{i}} + q \cdot \widehat{b}_{\mathsf{i}}} \cdot (h_{\mathsf{i}})^{\widehat{r}_{\mathsf{ct}:\beta}} = \prod_{s=1}^{\ell} (e_{\beta:s,\mathsf{i}})^{c_s} \cdot \widetilde{e}_{\beta:s,\mathsf{i}} \qquad \forall \mathsf{i} \in [t];$$
$$g^{\widehat{a}_{\mathsf{i}}} = \prod_{s=1}^{\ell} (\mathfrak{g}_{(\mathsf{i}-1)\ell+s})^{c_s} \cdot \widetilde{\mathfrak{g}}_{\mathsf{i}} \qquad \forall \mathsf{i} \in [t];$$
$$\widehat{a}_{\mathsf{i}} \leq q \cdot \ell \cdot 2^{2\lambda}; \qquad \widehat{b}_{\mathsf{i}} \leq q \cdot \ell \cdot 2^{3\lambda}; \qquad \widehat{\alpha}_{\mathsf{i}} \leq q \cdot \ell \cdot 2^{2\lambda}.$$

Figure 13: Sigma protocol for the ZK-AOK (3).

Zero-knowledge. Similarly as previous sections, the simulator first samples random challenges in Step 2 and Prover's response in Step 3. Then it computes the corresponding messages in Step 1 and finally outputs the simulated transcript. The simulated transcript is statistically indistinguishable to the real protocol.

Damgård-Jurik style batched encryption. The sigma protocol description above does not incorporate the Damgård-Jurik style batching we described in Sections 5.2.1 and 5.2.3, namely extending the modulus to N^{s+1} and dividing the message space of size N^s into slots of size B each, and combining $s' = \lfloor N^s/B \rfloor$ messages $\{m_i\}_{i\in[s']}$ into a single large message $m = \sum_{i=1}^{s'} m_i \cdot B^{i-1}$ to fully utilize the large plaintext space.

However, we observe that the sigma protocol in this section extends naturally to this type of batching. Notice that the current sigma protocol proves that a component $e_{\beta:s,\mathfrak{i}}$ encrypts the value $\alpha_i + q \cdot b_i$ for committed α_i and b_i. That is, the sigma protocol already proves that a ciphertext contains a linear combination of committed values, and this is done by exponentiating $(1 + N_2)$ to the power q, and using $(1 + N_2)^q$ as the base relative to which we prove the sigma protocol.

We can extend the same technique, to show that $e_{\beta:s,\mathfrak{i}}$ contains $\sum_{i=1}^{s'}(\alpha_i \cdot B^{i-1} + q \cdot B^{i-1} \cdot b_i)$ for committed α_i and b_i. That is, we exponentiate the corresponding bases to the powers B^{i-1} and $q \cdot B^{i-1}$. We note that this technique relies on the range proof to guarantee that none of the extracted values overflow into neighboring slots, and that the extracted sum does not wrap around in the message space.

C.7 Round 3 of Shuffled DOPRF

In Round 3 of the shuffled DOPRF protocol where P_1 holds the input (see Figure 4), for each $i \in [n_1]$, party P_2 computes $\beta_i \leftarrow \mathsf{CS\_Dec}_{\mathsf{sk}}(\mathsf{ct}_{\beta_i})$ where sk is P_2's Camenisch-Shoup secret key, and $\mathsf{C}_{\beta_i} \leftarrow \mathsf{com}_{\mathfrak{g},\mathfrak{h}}(\beta_i)$ where $(\mathfrak{g}, \mathfrak{h})$ are P_1's Pedersen commitment parameters. P_2 also computes $\gamma_i = \beta_i^{-1} \bmod q$ and $\sigma_i = \mathfrak{g}_i^{\gamma_i}$. In addition it computes $\mathfrak{ct}_{\sigma_i} \leftarrow \mathsf{EG\_Enc}_{\mathfrak{pk}}(\sigma_i)$ where $\mathfrak{pk}$ is P_2's ElGamal public key. In this round P_2 proves the following ZK-AOK:

$$\mathsf{ZK\text{-}AoK}\Big\{(\mathsf{sk}_2, \beta_i, r_1, r_2) : \beta_i = \mathsf{CS\_Dec}_{\mathsf{sk}}(\mathsf{ct}_{\beta_i}) \wedge \mathsf{C}_{\beta_i} = \mathfrak{g}^{\beta_i} \cdot \mathfrak{h}^{r_1} \wedge \beta_i \leq q^2 \cdot 2^{3\lambda+1} \wedge \mathfrak{ct}_{\sigma_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}\left((\mathfrak{g}_i)^{\beta_i^{-1}}; r_2\right)\Big\}.$$

Note that $\mathfrak{g}_i$ can be viewed as an ElGamal encryption with randomness 0. If we let $\mathfrak{ct}_{\mathfrak{g}_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(\mathfrak{g}_i; 0)$, then we want to prove $\mathfrak{ct}_{\mathfrak{g}_i} = (\mathfrak{ct}_{\sigma_i})^{\beta_i} \cdot \mathsf{EG\_Enc}_{\mathfrak{pk}}(1; -r_2 \cdot \beta_i)$ for a committed β_i. To batch this proof, we can use the multi-exponentiation argument from the work of Bayer and Groth [BG12]. However, it requires β_i to be committed under the same group as the ElGamal encryption, hence we will also commit to β_i in the ElGamal group and prove consistency of the two commitments of β_i.

At a high level, our proof consists of three steps. In the first step, we generate the Pedersen commitment parameters in the ElGamal group and prove correctness of the parameter generation. Second, we commit to β_i under both the Camenisch-Shoup encryption group and the ElGamal encryption group, and prove they are both consistent with the Camenisch-Shoup decrypted values. In the last step, we prove correctness of the ElGamal encryption via multi-exponentiation argument.

C.7.1 Generating Pedersen commitment parameters

In this step, the verifier generates Pedersen commitment parameters $(g_1, g_2, \ldots, g_t, h)$ for the group ElGamal group $\mathbb{G}$ of order q and sends to the prover , together with a proof that $g_i \in < h >$.

The format of this latter proof depends on the specific ElGamal group $\mathbb{G}$. When $\mathbb{G}$ is an appropriate elliptic curve, the prover can simply check that each g_i is a curve point, and a separate proof is not necessary. When $\mathbb{G}$ is the prime-order multiplicative group modulo a safe prime, the verifier must show that each g_i is a square.

C.7.2 Committing to β_i

In this step, the prover commits to β_i using both the parameters $(\mathfrak{g}_1, \ldots, \mathfrak{g}_t, \mathfrak{h})$ and $(g_1, \ldots, g_t, h)$, and proves both commitments are both consistent with the Camenisch-Shoup decrypted values.

We can batch the commitments of $\{\beta_i\}_{i\in[n_1]}$ as follows:

$$\begin{aligned}
\mathsf{C}_{\beta:1} &= (\mathfrak{g}_1)^{\beta_1} \cdot (\mathfrak{g}_2)^{\beta_{\ell+1}} \cdots (\mathfrak{g}_t)^{\beta_{(t-1)\ell+1}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\beta:1}} \\
&\vdots \\
\mathsf{C}_{\beta:s} &= (\mathfrak{g}_1)^{\beta_s} \cdot (\mathfrak{g}_2)^{\beta_{\ell+s}} \cdots (\mathfrak{g}_t)^{\beta_{(t-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\beta:s}} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\beta_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\beta:s}} \\
&\vdots \\
\mathsf{C}_{\beta:\ell} &= (\mathfrak{g}_1)^{\beta_\ell} \cdot (\mathfrak{g}_2)^{\beta_{2\ell}} \cdots (\mathfrak{g}_t)^{\beta_{t\ell}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\beta:\ell}}
\end{aligned}$$

Similarly, commitments under parameters $(g_1, \ldots, g_t, h)$:

$$C_{\beta:s} = (g_1)^{\beta_s} \cdot (g_2)^{\beta_{\ell+s}} \cdots (g_t)^{\beta_{(t-1)\ell+s}} \cdot h^{r_{C:\beta:s}} = \prod_{\mathfrak{i}=1}^{t} (g_\mathfrak{i})^{\beta_{(\mathfrak{i}-1)\ell+s}} \cdot h^{r_{C:\beta:s}} \qquad \forall s \in [\ell].$$

Recall that P_2's Camenisch-Shoup secret key is in fact $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_t)$. The encryptions of $\{\beta_i\}_{i\in[n_1]}$ are batched in the previous round as $\mathsf{ct}_{\beta:s} = (u_{\beta:s}, e_{\beta:s:1}, \ldots, e_{\beta:s:t})$ for all $s \in [\ell]$. In this round we want to prove

$$e_{\beta:s:\mathfrak{i}} = (1+N_2)^{\beta_{(\mathfrak{i}-1)\ell+s}} \cdot (u_{\beta:s})^{\mathsf{sk}_\mathfrak{i}} \qquad \forall s \in [\ell], \mathfrak{i} \in [t].$$

We now prove the consistency of $\{\beta_i\}_{i\in[n_1]}$ between batched Camenisch-Shoup decryption and batched commitments:

$$\begin{aligned}
\mathsf{ZK\text{-}AoK}\Big\{ \big(\{\beta_i\}_{i\in[n_1]}, \{r_{\mathsf{C}:\beta:s}, r_{C:\beta:s}\}_{s\in[\ell]}, \{\mathsf{sk}_\mathfrak{i}\}_{\mathfrak{i}\in[t]}\big) : \\
e_{\beta:s:\mathfrak{i}} = (1+N_2)^{\beta_{(\mathfrak{i}-1)\ell+s}} \cdot (u_{\beta:s})^{\mathsf{sk}_\mathfrak{i}} \quad \forall s \in [\ell], \mathfrak{i} \in [t] \ \wedge \\
\mathsf{C}_{\beta:s} = \prod_{\mathfrak{i}=1}^{t} (\mathfrak{g}_\mathfrak{i})^{\beta_{(\mathfrak{i}-1)\ell+s}} \cdot \mathfrak{h}^{r_{\mathsf{C}:\beta:s}} \quad \forall s \in [\ell] \ \wedge \ \beta_i \le q^2 \cdot \ell \cdot 2^{3\lambda+1} \ \wedge \\
C_{\beta:s} = \prod_{\mathfrak{i}=1}^{t} (g_\mathfrak{i})^{\beta_{(\mathfrak{i}-1)\ell+s}} \cdot h^{r_{C:\beta:s}} \quad \forall s \in [\ell] \Big\}.
\end{aligned} \tag{4}$$

The batched sigma protocol is shown in Figure 14. The protocol is complete with all but negligible probability. Next we prove its soundness and zero-knowledge.

1. Prover does the following:
   (a) Sample $\widetilde{\beta}_{\mathsf{i}} \xleftarrow{\$} [q^2 \cdot \ell \cdot 2^{3\lambda}]$ for all $\mathsf{i} \in [t]$ and $\widetilde{r}_{\mathsf{C}:\beta} \xleftarrow{\$} [N_1 \cdot \ell \cdot 2^{2\lambda}]$, $\widetilde{r}_{C:\beta} \xleftarrow{\$} [q]$.
   (b) Compute $\widetilde{\mathsf{C}}_\beta = \prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widetilde{\beta}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widetilde{r}_{\mathsf{C}:\beta}}$ and $\widetilde{C}_\beta = \prod_{\mathsf{i}=1}^{t} (g_{\mathsf{i}})^{\widetilde{\beta}_{\mathsf{i}}} \cdot h^{\widetilde{r}_{C:\beta}}$.
   (c) Compute a batched Camenisch-Shoup encryption of $\left\{\widetilde{\beta}_{\mathsf{i}}\right\}_{\mathsf{i}\in[t]}$. Let the encryption be $(\widetilde{u}_\beta, \widetilde{e}_{\beta:1}, \ldots, \widetilde{e}_{\beta:t})$, then
   $$\widetilde{e}_{\beta:\mathsf{i}} = (1+N_2)^{\widetilde{\beta}_{\mathsf{i}}} \cdot (\widetilde{u}_\beta)^{\mathsf{sk}_{\mathsf{i}}} \quad \forall \mathsf{i} \in [t].$$
   (d) Send $\left(\widetilde{\mathsf{C}}_\beta, \widetilde{C}_\beta, (\widetilde{u}_\beta, \widetilde{e}_{\beta:1}, \ldots, \widetilde{e}_{\beta:t})\right)$ to Verifier.
2. Verifier chooses random $c_1, \ldots, c_\ell \xleftarrow{\$} \{0,1\}^\lambda$ and sends to Prover.
3. Prover does the following:
   (a) Compute the following and send to Verifier:
   $$\widehat{\beta}_{\mathsf{i}} = \sum_{s\in[\ell]} c_s \cdot \beta_{(\mathsf{i}-1)\ell+s} + \widetilde{\beta}_{\mathsf{i}} \quad \forall \mathsf{i} \in [t];$$
   $$\widehat{r}_{\mathsf{C}:\beta} = \sum_{s\in[\ell]} c_s \cdot r_{\mathsf{C}:\beta:s} + \widetilde{r}_{\mathsf{C}:\beta}; \qquad \widehat{r}_{C:\beta} = \sum_{s\in[\ell]} c_s \cdot r_{C:\beta:s} + \widetilde{r}_{C:\beta}$$
   (b) Let
   $$u_{\widehat{\beta}} = \widetilde{u}_\beta \cdot \prod_{s\in[\ell]} (u_{\beta:s})^{c_s}; \qquad e_{\widehat{\beta}:\mathsf{i}} = \widetilde{e}_{\beta:\mathsf{i}} \cdot \prod_{s\in[\ell]} (e_{\beta:s:\mathsf{i}})^{c_s} \quad \forall \mathsf{i} \in [t].$$
   Then $(u_{\widehat{\beta}}, e_{\widehat{\beta}:1}, \ldots, e_{\widehat{\beta}:t})$ is a batched Camenisch-Shoup encryption of $\left\{\widehat{\beta}_{\mathsf{i}}\right\}_{\mathsf{i}\in[t]}$.
   (c) Sample $\widetilde{\mathsf{sk}}_{\mathsf{i}} \xleftarrow{\$} [N_2 \cdot 2^{2\lambda}]$ for all $\mathsf{i} \in [t]$ and send the following to Verifier:
   $$\widetilde{e}_{\widehat{\beta}:\mathsf{i}} = (1+N_2)^{\widehat{\beta}_{\mathsf{i}}} \cdot \left(u_{\widehat{\beta}}\right)^{\widetilde{\mathsf{sk}}_{\mathsf{i}}} \quad \forall \mathsf{i} \in [t].$$
4. Verifier chooses random challenges $c \xleftarrow{\$} \{0,1\}^\lambda$ and sends to Prover.
5. Prover computes $\widehat{\mathsf{sk}}_{\mathsf{i}} = c \cdot \mathsf{sk}_{\mathsf{i}} + \widetilde{\mathsf{sk}}_{\mathsf{i}}$ for all $\mathsf{i} \in [t]$ and sends to Verifier.
6. Verifier verifies the following:
   $$\prod_{\mathsf{i}=1}^{t} (\mathfrak{g}_{\mathsf{i}})^{\widehat{\beta}_{\mathsf{i}}} \cdot \mathfrak{h}^{\widehat{r}_{\mathsf{C}:\beta}} = \widetilde{\mathsf{C}}_\beta \cdot \prod_{s\in[\ell]} (\mathsf{C}_{\beta:s})^{c_s}; \qquad \prod_{\mathsf{i}=1}^{t} (g_{\mathsf{i}})^{\widehat{\beta}_{\mathsf{i}}} \cdot h^{\widehat{r}_{C:\beta}} = \widetilde{C}_\beta \cdot \prod_{s\in[\ell]} (C_{\beta:s})^{c_s};$$
   $$(1+N_2)^{\widehat{\beta}_{\mathsf{i}} \cdot (c+1)} \cdot \left(u_{\widehat{\beta}}\right)^{\widehat{\mathsf{sk}}_{\mathsf{i}}} = \widetilde{e}_{\widehat{\beta}:\mathsf{i}} \cdot \left(e_{\widehat{\beta}:\mathsf{i}}\right)^{c} \quad \forall \mathsf{i} \in [t];$$
   $$\widehat{\beta}_{\mathsf{i}} \leq q^2 \cdot \ell \cdot 2^{3\lambda} \quad \forall \mathsf{i} \in [t].$$

Figure 14: Sigma protocol for the ZK-AOK (4).

Soundness. Starting from Step 3c, the way to extract $\{\mathsf{sk_i}\}_{\mathsf{i}\in[t]}$ is the same as in Appendix C.4. Given the extracted $\{\mathsf{sk_i}\}_{\mathsf{i}\in[t]}$, we can extract $\{\beta_i\}_{i\in[n_1]}$ and $\{r_{\mathsf{C}:\beta:s}, r_{C:\beta:s}\}_{s\in[\ell]}$ in the same way as in Appendix C.4. The range proofs for these extracted values can be easily checked. In addition, the extracted values from the commitments are consistent to the extracted values from the decryptions because they can be deduced in the same way.

Zero-knowledge. The simulator first samples random challenges in Step 2 and Prover's response in Step 3a, from which it can compute $\widetilde{\mathsf{C}}_\beta$ and $\widetilde{C}_\beta$ in Step 1b similarly as in previous sections. Then the simulator computes a batched Camenisch-Shoup encryption of $\left\{\widehat{\beta}_\mathsf{i}\right\}_{\mathsf{i}\in[t]}$, denoted as $(u_{\widehat{\beta}}, e_{\widehat{\beta}:1}, \ldots, e_{\widehat{\beta}:t})$, from which it can compute the messages in Step 1c. Afterwards, the simulator samples the random challenge in Step 4 and Prover's response in Step 5. Then it computes the corresponding messages in Step 3c and finally outputs the simulated transcript. The simulated transcript is statistically indistinguishable to the real protocol.

Damgård-Jurik style batching. The sigma protocol described in Figure 14 proved that a Camenisch-Shoup ciphertext component decrypts to a committed value. As for the batched sigma protocol for Round 2, this protocol also can be extended naturally to proving that a Camenisch-Shoup ciphertext component decrypts to a *linear combination* of committed values. The idea is largely the same as described in the intuition for Round 2, and we omit the details. This extension of the proof allows us to use Damgård-Jurik style ciphertexts and better utilize a large plaintext space of size N^s.

C.7.3 Multi-exponentiation argument

Recall that we have $\mathfrak{ct}_{\sigma_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}\left((\mathsf{g}_i)^{\beta_i^{-1}}; r_i\right)$, $\mathfrak{ct}_{\mathsf{g}_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(\mathsf{g}_i; 0)$, hence

$$\mathfrak{ct}_{\mathsf{g}_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1; -\beta_i \cdot r_i) \cdot (\mathfrak{ct}_{\sigma_i})^{\beta_i}. \tag{5}$$

Note that from the previous step we have batched commitments of β_i. Let $\rho_i = -\beta_i \cdot r_i$, then we want to prove the following:

$$\begin{aligned}\mathsf{ZK\text{-}AoK}\Big\{ \big(\{\beta_i\}_{i\in[n_1]}, \{r_{C:\beta:\mathsf{i}}\}_{\mathsf{i}\in[t]}, \{\rho_i\}_{i\in[n_1]}\big) : \\ \mathfrak{ct}_{\mathsf{g}_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1; \rho_i) \cdot (\mathfrak{ct}_{\sigma_i})^{\beta_i} \quad \forall i \in [n_1] \wedge \\ C_{\beta:s} = \prod_{\mathsf{i}=1}^{t} (g_\mathsf{i})^{\beta_{(\mathsf{i}-1)\ell+s}} \cdot h^{r_{C:\beta:s}} \quad \forall s \in [\ell]\Big\}.\end{aligned} \tag{6}$$

At a high level, the verifier first picks random coefficients $(c_1, \ldots, c_{n_1})$. If we raise Equation 5 to the power c_i and multiply all the equations, we get

$$\mathfrak{ct} = \prod_{i\in[n_1]} (\mathfrak{ct}_{\mathsf{g}_i})^{c_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}\left(1; \sum_{i\in[n_1]} c_i\rho_i\right) \cdot \prod_{i\in[n_1]} \left(\mathfrak{ct}_{\sigma_i}^{c_i}\right)^{\beta_i}.$$

To prove knowledge of $\{\beta_i\}_{i\in[n_1]}$ and $\rho = \sum_{i\in[n_1]} c_i\rho_i$ in the above equation, we can use the multi-exponentiation argument from the work of Bayer and Groth [BG12]:

$$\mathsf{ZK\text{-}AoK}\Big\{ \big(\{\beta_i\}_{i\in[n_1]}, \{r_{C:\beta:\mathsf{i}}\}_{\mathsf{i}\in[t]}, \rho\big) :$$

$$\mathfrak{ct} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1;\rho) \cdot \prod_{i\in[n_1]} \left(\mathfrak{ct}_{\sigma_i}^{c_i}\right)^{\beta_i} \ \wedge$$

$$C_{\beta:s} = \prod_{\mathsf{i}=1}^{t} (g_{\mathsf{i}})^{\beta_{(\mathsf{i}-1)\ell+s}} \cdot h^{r_{C:\beta:s}} \quad \forall s \in [\ell] \Big\}.$$

1. Verifier chooses random challenges $c_1, \ldots, c_{n_1} \xleftarrow{\$} \{0,1\}^\lambda$ and sends to Prover.
2. Both parties compute $\mathfrak{ct} = \prod_{i\in[n_1]} (\mathfrak{ct}_{\mathfrak{g}_i})^{c_i}$.
3. Parties run a multi-exponentiation argument:

$$\mathsf{ZK\text{-}AoK}\Big\{ \left(\{\beta_i\}_{i\in[n_1]}, \{r_{C:\beta:\mathsf{i}}\}_{\mathsf{i}\in[t]}, \rho\right) :$$

$$\mathfrak{ct} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1;\rho) \cdot \prod_{i\in[n_1]} \left(\mathfrak{ct}_{\sigma_i}^{c_i}\right)^{\beta_i} \ \wedge$$

$$C_{\beta:s} = \prod_{\mathsf{i}=1}^{t} (g_{\mathsf{i}})^{\beta_{(\mathsf{i}-1)\ell+s}} \cdot h^{r_{C:\beta:s}} \quad \forall s \in [\ell] \Big\}.$$

Figure 15: ZK-AOK protocol for (6).

The ZK-AOK protocol is presented in Figure 15. The completeness of the protocol holds with all but negligible probability. Next we argue its soundness and zero-knowledge property.

Soundness. The extractor first picks random challenges $c_1, \ldots, c_{n_1} \xleftarrow{\$} \{0,1\}^\lambda$ and then extract $\left(\{\beta_i\}_{i\in[n_1]}, \{r_{C:\beta:\mathsf{i}}\}_{\mathsf{i}\in[t]}, \rho\right)$ from the multi-exponentiation argument. To further extract ρ_1, the extractor re-runs the protocol with challenges $c_1', c_2, \ldots, c_{n_1}$ and extracts ρ'. We claim that $\rho_1 = (c_1 - c_1')^{-1}(\rho - \rho')$. Since

$$(\mathfrak{ct}_{\mathfrak{g}_1})^{c_1'} \cdot \prod_{i=2}^{n_1} (\mathfrak{ct}_{\mathfrak{g}_i})^{c_i} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1;\rho') \cdot \left(\mathfrak{ct}_{\sigma_1}^{c_1'}\right)^{\beta_1} \cdot \prod_{i=2}^{n_1} \left(\mathfrak{ct}_{\sigma_i}^{c_i}\right)^{\beta_i}.$$

Combining the equation with the first run, the extractor gets

$$(\mathfrak{ct}_{\mathfrak{g}_1})^{\Delta c_1} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1;\Delta c_1 \cdot \rho_1) \cdot \left(\mathfrak{ct}_{\sigma_1}^{\beta_1}\right)^{\Delta c_1}.$$

Therefore $\mathfrak{ct}_{\mathfrak{g}_1} = \mathsf{EG\_Enc}_{\mathfrak{pk}}(1;\rho_1) \cdot (\mathfrak{ct}_{\sigma_1})^{\beta_1}$. Similarly we can extract all the $\{\rho_i\}_{i\in[n_1]}$.

Zero-knowledge. The simulator first follows the first two steps honestly and then launch the simulator for the multi-exponentiation argument.