Private Intersection-Sum Protocol with Applications to Attributing Aggregate Ad Conversions

Mihaela Ion[†], Ben Kreuter[†], Erhan Nergiz[†], Sarvar Patel[†],
Shobhit Saxena[†], Karn Seth[†], David Shanahan[†] and Moti Yung[‡*]
[†]{mion, benkreuter, anergiz, sarvar,
shobhitsaxena, karn, dshanahan}@google.com
Google Inc.
[‡]moti@cs.columbia.edu
Columbia University and Snap Inc.

July 31, 2017

Abstract

In this work, we consider the Intersection-Sum problem: two parties hold datasets containing user identifiers, and the second party additionally has an integer value associated with each user identifier. The parties want to learn the number of users they have in common, and the *sum* of the associated integer values, but "nothing more". We present a novel protocol tackling this problem using Diffie-Hellman style Private Set Intersection techniques together with Paillier homomorphic encryption. We prove security of our protocol in the honest-but-curious model. We also discuss applications for the protocol for attributing aggregate ad conversions. Finally, we present a variant of the protocol, which allows aborting if the intersection is too small, in which case neither party learns the intersection-sum.

1 Introduction

Protocols for private set intersection (PSI) allow two or more parties to compute an intersection over their privately held input sets, without revealing anything more to the other party beyond the elements in the intersection. Related protocols allow parties to learn only restricted functions of the intersection, such as the cardinality of the intersection, or whether the size of the intersection exceeds some threshold. Various approaches have been presented in previous work, in both the honest-but-curious and malicious security models.

[*]Work done while at Google Inc.

In this work, we consider a particular variant of the PSI problem, which we call the Private Intersection Sum problem. In this setting, there are two parties that have private input sets consisting of identifiers, and one of the parties' datasets additionally has an integer value associated with each identifier. The parties want to learn cardinality of the intersection, as well as the *sum* of the associated integer values for each identifier in the intersection, but nothing more. In particular, neither party should learn the actual identifiers in the intersection, nor should they learn any additional information about the other party's data (beyond its size).

Our work is motivated by the general class of business problems of attributing online-to-offline ad conversions. An online-to-offline ad conversion occurs when a user sees an ad for some company on a website, and then later makes a purchase in that company's store. The company would like to know how much of its revenue it can attribute to online ads. However, the data needed to compute these attribution statistics is split across two parties: the ad supplier, who knows which users have seen which ads, and the company, which knows who made a purchase and what they spent. The two parties are typically unwilling or unable to expose the underlying data, but both parties would still like to compute an aggregate measurement: how many users both saw an ad and made a corresponding purchase, and how much those users spent in total. This is exactly an instance of the Private Intersection-Sum problem.

In this work, we present a protocol that allows two parties to privately compute the intersection-sum functionality. We show security of the protocol in the *honest-but-curious* model. In this model, we assume participants follow the steps of the protocol honestly by generating well-formed messages, but may attempt to extract as much information as possible afterwards from the protocol transcript. A protocol is secure in this model if the transcript does not reveal any additional information beyond the functionality being computed; this is analogous to the concept of "perfect forward secrecy" in TLS, which ensures that once a session ends the transcript reveals nothing even given the parties' secret keys.

1.1 Paper Organization

In Section 2, we provide some useful definitions for our protocol. In Section 3, we give a description of our Intersection-Sum protocol, with a detailed security analysis in the honest-but-curious model in Section 3.1. We also present a "reverse" variant of our protocol in Section 4. In Section 5 we mention several related works.

2 Security Primitives and Cryptographic Assumptions

Definition 1 (Paillier Homomorphic Encryption)**.** *The Paillier encryption scheme [Pai99] is an additively homomorphic encryption scheme, consisting of the following probabilistic polynomial-time algorithms:*

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

$\mathsf{Pai}.\mathbf{Enc}$ *Given the public key pk and a plaintext message* $m \in \mathcal{M}$, *one can compute a ciphertext* $\mathsf{Pai}.Enc(pk,m)$, *a Paillier encryption of m under pk. (We shorten this to just* $\mathsf{Pai}(m)$ *when pk is clear from the context).*

$\mathsf{Pai}.\mathbf{Dec}$ *Given the secret key sk and a ciphertext* $\mathsf{Pai}.Enc(pk,m)$, *one can run* $\mathsf{Pai}.Dec$ *to recover the plaintext m.*

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

$$\mathsf{Pai}.Enc(pk,\sum_i m_i) = \mathsf{Pai}.Sum(\{\mathsf{Pai}.Enc(pk,m_i)\}_i)$$

The scheme satisfies the standard notion of CPA security of encryption, meaning, informally, that without knowledge of the private key *sk*, encryptions of different messages are computationally indistinguishable.

In addition, we will make use of the property that $\mathsf{Pai}.Sum(\{\mathsf{Pai}.Enc(pk,m_i)\})$ and $\mathsf{Pai}.Enc(pk,\sum_i m_i)$ have identical distributions. This property is not described in the original scheme description [Pai99], but can easily be added to $\mathsf{Pai}.Sum$ by always including an additional fresh encryption of 0 in the ciphertexts to be summed.

Definition 2 (Decisional Diffie-Hellman assumption (DDH))**.** *[DH76] Let* $\mathcal{G}(\lambda)$ *be a group family parameterized by security parameter* λ*. For every probabilistic adversary M that runs in time polynomial in* λ*, we define the advantage of M to be:*

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

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

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

3 Protocol Description

A detailed description of our Intersection Sum protocol is found in Figure 2. In the protocol presented, there are two participating parties, of which only Party 1 learns the cardinality of the intersection, and only Party 2 learns the intersection-sum.

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

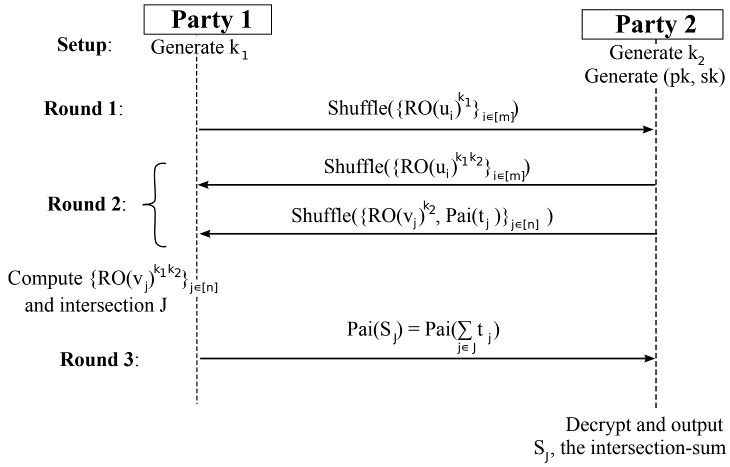

Figure 1: Summary of the Intersection-Sum protocol

At a high-level, the two parties interact to hash and “double-encrypt” each entry in their datasets, and compare the double-encrypted values. The “double-encryption” we perform is similar to the deterministic Pohlig-Hellman cipher [HP84].

The group $\mathcal{G}$ can be any group in which the DDH assumption is believed to hold. Several candidate groups are widely used, such as subgroups of the multiplication group of a finite field and elliptic and hyperelliptic curve groups. In practice, carefully chosen elliptic curves like Curve25519 [Ber06] offer a good balance between security and performance.

We also note that our protocol has both parties make use of a Random Oracle RO, which must be different for each protocol instance. We can instantiate this Random Oracle in practice using a cryptographic hash function. Hash functions such as SHA-256 can be adapted to hash into a specific group $\mathcal{G}$ using rejection sampling. In the case where $\mathcal{G}$ is Curve25519, hashing to the curve is straightforward, as every 256 bit string can be interpreted as a curve point. To simulate using a different Random Oracle for each protocol instance, parties can simply prepend an instance identifier to their inputs to the hash function.

3.1 Security Analysis

As discussed earlier, we will prove security of our protocol in the honest-but-curious model, where we assume participants follow the steps of the protocol honestly, but try to extract as much information as possible afterwards from the protocol transcript. This model still requires some degree of trust between the two parties not to deviate from the prescribed protocol.

We prove security in the honest-but-curious model. We show security by giving a simulator that can indistinguishably simulate the view of each honest party in the protocol given only that party’s input, the cardinality of the intersection, and the intersection-sum (but not the input of the other party). Intuitively, this will show that each party learns nothing more by participating in the protocol than the cardinality of the intersection and the intersection sum.

In such a protocol execution, the *view* of a party consists of its internal state (including its

Private Intersection-Sum Protocol

- **Setup**:
  - Both parties agree on a security parameter λ and a $\mathcal{G} \in \mathcal{G}(\lambda)$, and a user identifier space $\mathcal{U} = \mathcal{U}(\lambda)$. Both parties have access to a Random Oracle $\mathsf{RO} : \mathcal{U} \to \mathcal{G}$ that maps user identifiers to random elements of $\mathcal{G}$.
  - Party 1 has as input a set $U_1 = \{u_i\}_{i \in [m]}$ of m user identifiers, where each $u_i \in \mathcal{U}$.
  - Party 2 has as input a set $\{(v_j, t_j)\}_{j \in [n]}$ of n user identifiers paired with transaction values, where each $v_j \in \mathcal{U}$, and each $t_j \in \mathbb{Z}^+$, such that $\sum t_j$ fits comfortably into the Paillier message space for security parameter λ. We define $U_2 = \{v_j\}_{j \in [n]}$.
  - Each Party i chooses a random private exponent k_i in the group $\mathcal{G}$.
  - Party 2 generates a fresh key-pair $(pk, sk) \leftarrow \mathsf{Pai}.Gen(\lambda)$ for the Paillier encryption scheme and shares the public key pk with Party 1.

- **Round 1 (Party 1)**:
  1. For each element u_i in its set, Party 1 applies the Random Oracle and then single-encrypts them using its key k_1, thus computing $\mathsf{RO}(u_i)^{k_1}$.
  2. Party 1 sends $\{\mathsf{RO}(u_i)^{k_1}\}_{i \in [m]}$ to Party 2 in shuffled order.

- **Round 2 (Party 2)**:
  1. For each element $\mathsf{RO}(u_i)^{k_1}$ received from Party 1 in the previous step, Party 2 double-encrypts them using its key k_2, computing $\mathsf{RO}(u_i)^{k_1 k_2}$.
  2. Party 2 sends $Z = \{\mathsf{RO}(u_i)^{k_1 k_2}\}_{i \in [m]}$ to Party 1 in shuffled order.
  3. For each item (v_j, t_j) in its input set, Party 2 applies the Random Oracle to the first element of the pair and encrypts it using key k_2. It encrypts the second element of the pair using the Paillier key pk. It thus computes the pair. $\mathsf{RO}(v_j)^{k_2}$ and $\mathsf{Pai}(t_j)$.
  4. Party 2 sends the set $\{(\mathsf{RO}(v_j)^{k_2}, \mathsf{Pai}(t_j))\}_{j \in [n]}$ to Party 1 in shuffled order.

- **Round 3 (Party 1)**:
  1. For each item $(\mathsf{RO}(v_j)^{k_2}, \mathsf{Pai}(t_j))$ received from Party 2 in Round 2 Step 4, Party 1 double-encrypts the first member of the pair using k_1, thus computing $(\mathsf{RO}(v_j)^{k_1 k_2}, \mathsf{Pai}(t_j))$.
  2. Party 1 computes the intersection set J:
  $$J = \{j : \mathsf{RO}(v_j)^{k_1 k_2} \in Z\}$$
  where Z is the set received from Party 1 in Round 1.
  3. For all items in the intersection, Party 1 homomorphically adds the associated ciphertexts, and computes a ciphertext encrypting the intersection-sum S_J:
  $$\mathsf{Pai}(pk, S_J) = \mathsf{Pai}.Sum\left(\{\mathsf{Pai}(t_j)\}_{j \in J}\right) = \mathsf{Pai}\left(\sum_{j \in J} t_j\right)$$
  4. Party 1 sends this ciphertext to Party 2.

- **Output (Party 2)**: Party 2 decrypts the Paillier ciphertext received in Round 3 using the Paillier secret key sk to recover the intersection-sum S_J.

Figure 2: Detailed description of the Private Intersection-Sum protocol.

input and randomness) and all messages this party received from the other party (the messages sent by this party do not need to be part of the view because they can be determined using the other elements of its view).

Let $\mathtt{REAL}^{i,\lambda}(\{u_i\}_{i\in[m]},\{(v_j,t_j)\}_{j\in[n]})$ be a random variable representing the view of Party i in a real protocol execution, where the random variable ranges over the internal randomness of all parties, and the randomness in the setup phase (including that of the Random Oracle).

Our first theorem shows that Party 1's view in the protocol can be simulated given only that Party 1's input and the size of the intersection (but not the input of Party 1).

Theorem 1 (Honest But Curious Security, against Party 1)**.** *There exists a PPT simulator* $\mathtt{SIM}_1$ *such that for all security parameters* λ *and inputs* $\{u_i\}_{i\in[m]}$, $\{(v_j,t_j)\}_{j\in[n]}$,

$$\mathtt{REAL}^{1,\lambda}(\{u_i\}_{i\in[m]},\{(v_j,t_j)\}_{j\in[n]})$$
$$\approx$$
$$\mathtt{SIM}_1(1^\lambda,\{u_i\}_{i\in[m]},n,|J|)$$

Where n is the size of Party 2's input, $J=\{j : v_j \in \{u_i\}_{i\in[m]}\}$ is the intersection set, and $|J|$ is its cardinality.

Proof. We describe the simulator algorithm $\mathtt{SIM}_1$ in Algorithm 1.

Algorithm 1: The simulator for Party 1
Input: $(\lambda,\{u_i\}_{i\in[m]},n,|J|)$
Output: $SimView(P_1)$
$\mathtt{SIM}_1(\lambda,\{u_i\}_{i\in[m]},|J|)$
(1) Generate key $k_1\in\mathcal{G}$, and Paillier key-pair (pk,sk).
(2) Honestly generate and send $\{\mathsf{RO}(u_i)^{k_1}\}_{i\in[m]}$ in shuffled order as Party 1's message in Round 1.
(3) Create a dummy set $U_1^*=\{g_i\}_{i\in[m]}$, where each g_i is randomly selected from $\mathcal{G}$. Send $\{g_i^{k_1}\}_{i\in[m]}$ in shuffled order as Party 2's message in Step 2 of Round 2.
(4) Create a dummy set $\mathcal{U}_2^*=\{h_j\}_{j\in[n]}$ for Party 2 by setting $h_j=g_j$ for $j\in[1,|J|)$, and each h_j for $j\in[|J|,m)$ is randomly selected from $\mathcal{G}$.
(5) Send $\{(h_j,\mathsf{Pai}(pk,0))\}_{j\in[n]}$ in shuffled order as Party 2's message in Step 4 of Round 2, where each $\mathsf{Pai}(0)$ is freshly generated.
(6) Honestly generate Party 1's message in Round 3 using Party 2's dummy messages from the previous step.
(7) Output Party 1's view in the simulated execution above.

Notice that the main difference between $\mathtt{SIM}_1$ and a real protocol execution is in Round 2: instead of sending $\{\mathsf{RO}(u_i)^{k_1k_2}\}$ and $\{(\mathsf{RO}(v_j)^{k_2}\}$ as in a real execution, $\mathtt{SIM}_1$ instead uses random group elements $\{g_i\}$ and $\{h_j\}$ which have an intersection of the same size, and Paillier encryptions of 0. We argue that

$$\mathtt{REAL}^{1,\lambda}(\{u_i\}_{i\in[m]}, \{(v_j,t_j)\}_{j\in[n]}) \approx \mathtt{SIM}_1(\lambda, \{u_i\}_{i\in[m]}, |J|)$$

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

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

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

$\mathsf{Hyb}_{1,i}$ for $i \in [m-|J|]$: The same as $\mathsf{Hyb}_{1,i-1}$, except with $\mathsf{RO}(u_{i^*})^{k_1 k_2}$ replaced by $g_{i^*}^{k_1}$ in Party 2's message in Step 2 of Round 2, where where u_{i^*} is the lexicographically smallest as-yet-unreplaced element of $\{u_i\}_{i\in[m]} \setminus \{v_j\}_{j\in[n]}$, and g_{i^*} is a random element of $\mathcal{G}$.

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

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

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

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

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

where $u_{k^*} = v_{k^*}$ is the lexicographically smallest as-yet-unreplaced element of $\{v_j\}_{j\in[n]} \cap \{g_i\}_{i\in[m]}$, and g_{k^*} is a random element of $\mathcal{G}$.

Hyb_4 The view of Party 1 output by $\mathtt{SIM}_1$.

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

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

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

Consider Algorithm 2 below, that takes as input a DDH tuple (g, g^a, g^b, g^c) and hybrid index i, and simulates $\mathsf{Hyb}_{1,i}$:

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

Input: $(\lambda, i, (g, g^a, g^b, g^c), \{u_i\}_{i\in[m]}, \{v_j\}_{j\in[n]})$

Output: $SimView(P_1)$ in $\mathsf{Hyb}_{1,i}$

$\mathtt{SIM}_{\mathsf{Hyb}_{1,i}}(\lambda, i^*, (g, g^a, g^b, g^c), \{u_i\}_{i\in[m]}, \{v_j\}_{j\in[n]})$

(1) Reprogram the Random Oracle using the DDH tuple as follows:

$$\begin{aligned} \mathsf{RO}(u_i) &= g^{r_i} && \text{for } u_i \neq u_{i^*} \\ &= g^a && \text{for } u_i = u_{i^*} \\ \mathsf{RO}(v_j) &= g^{s_j} && \forall j \in [n] \end{aligned}$$

where each r_i and s_j is randomly chosen in the range $[1, |\mathcal{G}|)$, and u_{i^*} is the newest element replaced with a random one in $\mathsf{Hyb}_{1,i}$.

(2) Generate key $k_1 \in \mathcal{G}$, and Paillier key-pair (pk, sk).

(3) Send $\{\mathsf{RO}(u_i)^{k_1}\}_{i\in[m]}$ in shuffled order as Party 1's message in Round 1.

(4) Create a dummy set $U_1^* = \{g_i\}_{i\in[m]}$ as follows:

$$\begin{aligned} g_i &= g^c && \text{for } u_i = u_{i^*} \\ &= \text{random element of } \mathcal{G} && \text{for } u_i \notin \{v_j\}_{j\in[n]}, u_i < u_{i^*} \\ &= (g^b)^{s_i} && \text{for all other } u_i \end{aligned}$$

Send $\{g_i^{k_1}\}_{i\in[m]}$ in shuffled order as Party 2's message in Step 2 of Round 2.

(5) Send $\{((g^b)^{s_j}, \mathsf{Pai}(0))\}_{j\in[n]}$ in shuffled order as Party 2's message in Step 4 of Round 2, where each $\mathsf{Pai}(0)$ is freshly generated.

(6) Honestly generate Party 1's message in Round 3 using Party 2's dummy messages from the previous step.

(7) Output Party 1's view in the simulated execution above.

We observe that the output distribution produced by Algorithm 2 on input i and a DDH tuple (g, g^a, g^b, g^c) for uniformly random a, b, c is identical to $\mathsf{Hyb}_{1,i}$. To see this, we first observe that the Random Oracle has uniformly random outputs even after reprogramming, since all the reprogrammed values are random powers of a generator. Next, interpreting the hidden exponent b as Party 2's key k_2, all the simulated messages sent by Party 2 in Round 2 are of the correct form for $\mathsf{Hyb}_{1,i}$: un-replaced messages in Round 2 Step 2 have the form $\mathsf{RO}(u_i)^{k_1k_2}$, and messages sent in Round 2 Step 4 have the form $(\mathsf{RO}(v_j)^{k_2}, \mathsf{Pai}(0))$.

We now replace the DDH tuple given as input to Algorithm 2 to have the form (g, g^a, g^b, g^{ab}). The only effect is that, instead of $g_{i^*} = g^c$, we have $g_{i^*} = g^{ab} = \mathsf{RO}(u_{i^*})^b$. From our earlier interpretation of b as k_2, this means $g_{i^*}^{k_1} = \mathsf{RO}(u_i)^{k_1k_2}$. This change is exactly the difference between $\mathsf{Hyb}_{1,i-1}$ and $\mathsf{Hyb}_{1,i}$. Thus, the output of Algorithm 2 on inputs i and (g, g^a, g^b, g^{ab}) is identical to $\mathsf{Hyb}_{1,i-1}$.

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

Our second theorem shows that Party 2's view in the protocol can be simulated given only that Party 2's input and the intersection-sum (but not the input of Party 1).

Theorem 2 (Honest But Curious Security, against Party 2)**.** *There exists a PPT simulator* SIM *such that for all security parameters* λ *and inputs* $\{u_i\}_{i\in[m]}$, $\{(v_j,t_j)\}_{j\in[n]}$,

$$\begin{aligned}&\mathtt{REAL}^{2,\lambda}(\{u_i\}_{i\in[m]},\{(v_j,t_j)\}_{j\in[n]})\\ &\approx\\ &\mathtt{SIM}_2(1^\lambda,\{(v_j,t_j)\}_{j\in[n]},m,S_J)\end{aligned}$$

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

Proof. We define $\mathtt{SIM}_2$ to perform the Setup phase honestly, and honestly performs the operations corresponding to Party 2. $\mathtt{SIM}_2$ simulates the messages sent by Party 1 as follows:

- In Round 1, instead of sending $\{\mathsf{RO}(u_i)^{k_1}\}_{i\in[m]}$ as Party 1's message, $\mathtt{SIM}_2$ instead sends m randomly selected elements of $\mathcal{G}$.
- In Round 3, instead of performing the intersection and computing the intersection-sum, $\mathtt{SIM}_2$ instead sends a fresh Paillier ciphertext encrypting the value S_J it received as input.

We note that the only difference between the output of $\mathtt{SIM}_2$ and the view of Party 2 in a real execution is in the Round 1 messages. However, the Round 1 messages output by $\mathtt{SIM}_2$ can be shown to be indistinguishable from those in a real execution by using a simple hybrid argument: Define m hybrids, where, in each successive hybrid, $\mathtt{SIM}_2$ replaces one additional "real" Round 1 message of the form $\mathsf{RO}(u_i)^{k_1}$ with a random element of $\mathcal{G}$. Then, each pair of neighboring hybrids can be shown to be indistinguishable based on the fact that k_1 is secret and that DDH is hard in $\mathcal{G}$. The details are very similar to the proof of Theorem 1, and we leave them as an exercise.

□

3.2 Additional Security Precautions

Our security analysis shows that each party learns no more than the size of the intersection and the intersection-sum. However, unless appropriate care is taken, these values may themselves leak private information. For example, if the intersection size is very small, it may be possible to guess the user identifiers in the intersection based on the intersection-sum. To guarantee enough mixing-privacy between the users, parties should ensure that the intersection is sufficiently large. The "reverse" protocol variant we present in Section 4 allows parties to enforce a minimum intersection size, by allowing them to abort before either party learns the intersection sum if the if the intersection is too small.

In general, though, privacy may be violated as a consequence of certain input distributions. For example, if there are "outlier" v_j values that are unusually large, the sum will be large; a

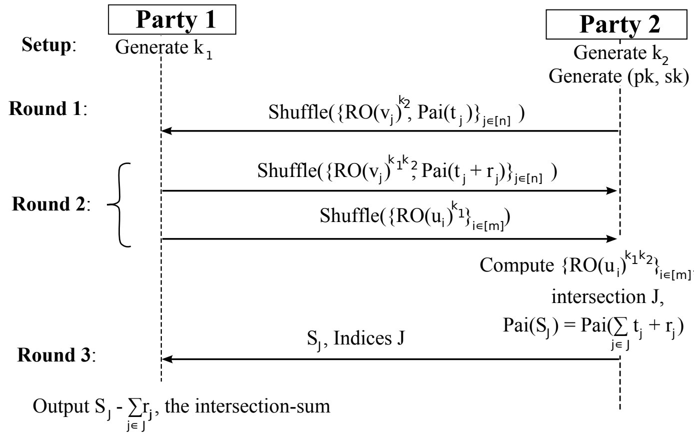

Figure 3: Summary of the Reverse Intersection-Sum protocol

priori knowledge of such values will allow a party to identify users. It is also possible that repeatedly executing the protocol in sequence will leak information due to correlated inputs in different sessions. Such problems are an artifact of the functionality itself [2] and would affect *any* intersection-sum protocol. One strategy for resolving this issue would be to compose differential privacy techniques [DR14] with the cryptographic protocol, by adding appropriately sampled noise to the inputs.

4 Protocol Variants: The "Reverse" Protocol

The protocol we presented in Section 3 can be modified in a straightforward way to allow both parties to learn the intersection-sum or intersection-size. It is also possible to ensure that one or the other party performs the actual intersection operation, for example, to allow that party to abort if the intersection is below some threshold, which might be imposed for policy reasons. We present one such variant in Figure 4, which we refer to as the "reverse" protocol. In this protocol, Party 2 performs the intersection, and can abort the protocol if the intersection size is too small, without either party learning the intersection-sum. In addition, both parties learn the intersection size, but only Party 1 learns the intersection-sum. To implement this, we additionally need Party 1 to blind the Paillier ciphertext with random values, as can be seen in Figure 4.

4.1 Security Analysis

The security proof for the reverse protocol is similar to the ordinary protocol, but with the roles of the parties reversed, with Party 2 learning only the intersection size, and Party 1 learning both

[2]This leakage is implicit in the security proof. The simulator will receive the result of the function evaluated on both parties' inputs; this result itself is subject to the problems above. The security proof shows that a party will learn only as much as the functionality reveals, which includes leakage due to the input distribution.

Reverse Intersection-Sum Protocol

- **Setup**:
  - Both parties agree on a security parameter λ and a $\mathcal{G} \in \mathcal{G}(\lambda)$, and a user identifier space $\mathcal{U} = \mathcal{U}(\lambda)$. Both parties have access to a Random Oracle $\mathsf{RO} : \mathcal{U} \to \mathcal{G}$ that maps user identifiers to random elements of $\mathcal{G}$.
  - Party 1 has as input a set $\{u_i\}_{i\in[m]}$ of m user identifiers, where each $u_i \in \mathcal{U}$.
  - Party 2 has as input a set $\{(v_j,t_j)\}_{j\in[n]}$ of n user identifiers paired with transaction values , where each $v_j \in \mathcal{U}$, and each $t_j \in \mathbb{Z}^+$, such that $\sum t_j$ fits comfortably into the Paillier message space.
  - Each Party i chooses a random private exponent k_i in the group $\mathcal{G}$.
  - Party 2 generates a fresh key-pair $(pk,sk) \leftarrow \mathsf{Pai}.Gen(\lambda)$ for the Paillier encryption scheme and shares the public key pk with Party 1.
- **Round 1 (Party 2)**:
  1. For each element (v_j,t_j) in its set, Party 2 applies the Random Oracle and then single-encrypts v_j using its key k_2, thus computing $\mathsf{RO}(v_j)^{k_2}$.
  2. Party 2 sends $\{(\mathsf{RO}(v_j)^{k_2}, \mathsf{Pai}(t_j))\}_{j\in[n]}$ to Party 1 in shuffled order.
- **Round 2 (Party 1)**:
  1. For each element $(\mathsf{RO}(v_j)^{k_2}, \mathsf{Pai}(t_j))$ received from Party 2 in the previous step, Party 1 double-encrypts them using its key k_1 and homomorphically computes a one-time pad encryption of t_j under addition modulo the Paillier modulus N, computing $(\mathsf{RO}(v_j)^{k_1k_2}, \mathsf{Pai}(t_j + r_j))$.
  2. Party 1 sends $\{(\mathsf{RO}(v_j)^{k_1k_2}, \mathsf{Pai}(t_j + r_j)\}_{j\in[n]}$ to Party 2 in shuffled order. The (shuffled $j \to r_j$) map is saved for a future step.
  3. For each item u_i in its input set, Party 1 applies the Random Oracle to the first element of the pair and encrypts it using key k_1. It encrypts the second element of the pair using the Paillier key pk. It thus computes the pair. $\mathsf{RO}(u_i)^{k_1}$.
  4. Party 1 sends the set $\{\mathsf{RO}(u_i)^{k_1}\}_{i\in[m]}$ to Party 2 in shuffled order.
- **Round 3 (Party 2)**:
  1. For each item $\mathsf{RO}(u_j)^{k_1}$ received from Party 1 in Round 2 Step 4, Party 1 double-encrypts the using k_2, thus computing $\mathsf{RO}(u_i)^{k_1k_2}$.
  2. Party 2 computes the intersection set J:
  $$J = \{j : \mathsf{RO}(v_j)^{k_1k_2} \in \{\mathsf{RO}(u_i)^{k_1k_2}\}_{i\in[m]}\}$$
  3. For all items in the intersection, Party 2 adds the associated (one-time pad encrypted) ciphertexts, and computes a ciphertext encrypting the intersection-sum $S_J = \sum_{j\in J} t_j + r_j$
  4. Party 2 sends S_J together with the indexes J corresponding to the Paillier ciphertexts in the intersection, to Party 1.
- **Output (Party 1)**: Party 1 computes $S_J - \sum_{j\in J} r_j$ to recover $\sum_{j\in J} t_j$.

Figure 4: Detailed description of the "Reverse" Private Intersection-Sum protocol.

the intersection size and the intersection sum. The simulator for Party 1 is almost identical to the simulator for Party 2 in the original protocol, but must also provide indices J in Round 3 to allow Party 1 to compute $S_J - \sum_{j \in J} r_j$. For Party 2, the simulator is very similar to the original Party 1 simulator $\mathtt{SIM}_1$ in Algorithm 1. We omit details of the proof.

5 Related Work

Private Set Intersection is a well-studied problem. The goal there is for both parties to learn the items in the intersection, but nothing more. There are many existing approaches in the literature, including works based on DDH-type assumptions [HFH99, DCKT10, Lam16, SFF14], works based on Oblivious Transfer [PSZ14, PSSZ15, DCW13, RR16], works based on Oblivious Polynomial Evaluation [FNP04, DSMRY09] and works based on generic Secure Two-party Computation techniques [HEK12, PSSZ15]. In our setting the universe of possible set members is large, so techniques assuming bit-vector representations of the set are inapplicable.

Closer to our goal, there are several works that limit the parties to learning only the *cardinality* of the intersection [FNP04, KS05, VC05, DCGT12, NAA+09]. Previous works using garbled circuits also allow cardinality, as well as more general functions of the intersection, however the communication overhead of garbled circuit approaches would be too high for our application. In Huang et al.'s work, the Sort-Compare-Shuffle approach, which is the most applicable to our setting, requires $O(n \log n)$ communication [HEK12], and even with state-of-the-art garbling schemes will require far more communication than our protocol. Similarly, the Phasing technique of Pinkas et al. [PSSZ15] also requires quasi-linear communication.

Due to the "offline" nature of our use-case, which allows for higher latency, comparisons based on overall running time are less valuable than the concrete resource costs required to run the protocol. For our application the most limited resource was network capacity, so we base our comparisons on the amount of communication required to compute the result.

6 Conclusion

We have developed an efficient family of protocols for computing linear functions over the intersection of two parties' data sets. We have also proven security in the honest-but-curious model. Our protocol can be coupled with a post-facto "audit" protocol to achieve a security notion similar to the covert model.

In our protocol the functions that can be computed over the intersection are determined by the particular homomorphic encryption scheme that is used. For our use-case Paillier encryption was sufficient, but it would be straightforward to use BGN to support quadratic functions or even an FHE scheme for more general functionalities, although such changes would impose heavier resource requirements. It may also be useful to use generic approaches such as garbled circuits as subprotocols to compute more general functions; we leave a detailed analysis for future work.

References

[Ber06] Daniel J. Bernstein. Curve25519: new diffie-hellman speed records. In *In Public Key Cryptography (PKC), Springer-Verlag LNCS 3958*, page 2006, 2006. (Cited on page 4.)

[DCGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *International Conference on Cryptology and Network Security*, pages 218–231. Springer, 2012. (Cited on page 12.)

[DCKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 213–231. Springer, 2010. (Cited on page 12.)

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800. ACM, 2013. (Cited on page 12.)

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976. (Cited on page 3.)

[DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014. (Cited on page 10.)

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *International Conference on Applied Cryptography and Network Security*, pages 125–142. Springer, 2009. (Cited on page 12.)

[FNP04] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*, pages 1–19. Springer, 2004. (Cited on page 12.)

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012. (Cited on page 12.)

[HFH99] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86. ACM, 1999. (Cited on page 12.)

[HP84] Martin E Hellman and Stephen C Pohlig. Exponentiation cryptographic apparatus and method, January 3 1984. US Patent 4,424,414. (Cited on page 4.)

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, pages 241–257, Berlin, Heidelberg, 2005. Springer-Verlag. (Cited on page 12.)

[Lam16] Mikkel Lambæk. Breaking and fixing private set intersection protocols. Technical report, Cryptology ePrint Archive, Report 2016/665, 2016. http://eprint. iacr. org/2016/665, 2016. (Cited on page 12.)

[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 *International Conference on Cryptology and Network Security*, pages 21–40. Springer, 2009. (Cited on page 12.)

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999. (Cited on pages 2 and 3.)

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security*, volume 15, pages 515–530, 2015. (Cited on page 12.)

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *Usenix Security*, volume 14, pages 797–812, 2014. (Cited on page 12.)

[RR16] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. Technical report, 2016. (Cited on page 12.)

[SFF14] Aaron Segal, Bryan Ford, and Joan Feigenbaum. Catching bandits and only bandits: Privacy-preserving intersection warrants for lawful surveillance. In *FOCI*, 2014. (Cited on page 12.)

[VC05] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005. (Cited on page 12.)