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:

\[ \text{Pai.Gen} \]  
Given a security parameter \( \lambda \), \text{Pai.Gen}(\lambda) \) returns outputs a public-private key pair \((pk, sk)\), and specifies a message space \( \mathcal{M} \).
Given the public key $pk$ and a plaintext message $m \in M$, one can compute a ciphertext $\text{Pai.Enc}(pk, m)$, a Paillier encryption of $m$ under $pk$. (We shorten this to just $\text{Pai}(m)$ when $pk$ is clear from the context).

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

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

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

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

**Definition 2** (Decisional Diffie-Hellman assumption (DDH)). [DH76] Let $G(\lambda)$ be a group family parameterized by security parameter $\lambda$. For every probabilistic adversary $M$ that runs in time polynomial in $\lambda$, 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 $G$ from $G(\lambda)$, ran generator $g$ of $G$, random $a, b, c \in [1, |G|]$ and the randomness of $M$. We say that the Decisional Diffie Hellman assumption holds for $G$ if for every such $M$, there exists a negligible function $\varepsilon$ 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 $M$. In this work, we only consider messages and sums that are too small to wrap around.
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 $G$ 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 $G$ using rejection sampling. In the case where $G$ 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 $\lambda$ and a $G \in G(\lambda)$, and a user identifier space $\mathcal{U} = \mathcal{U}(\lambda)$. Both parties have access to a Random Oracle $RO : \mathcal{U} \rightarrow G$ that maps user identifiers to random elements of $G$.
  - 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 $\lambda$. We define $U_2 = \{v_j\}_{j \in [n]}$.
  - Each Party $i$ chooses a random private exponent $k_i$ in the group $G$.
  - Party 2 generates a fresh key-pair $(pk, sk) \leftarrow \text{Pai.Gen}(\lambda)$ 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 $RO(u_i)^{k_1}$.
  2. Party 1 sends $\{RO(u_i)^{k_1}\}_{i \in [m]}$ to Party 2 in shuffled order.

- **Round 2 (Party 2):**
  1. For each element $RO(u_i)^{k_1}$ received from Party 1 in the previous step, Party 2 double-encrypts them using its key $k_2$, computing $RO(u_i)^{k_1 k_2}$.
  2. Party 2 sends $Z = \{RO(u_i)^{k_1 k_2}\}_{i \in [m]}$ 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. $RO(v_j)^{k_2}$ and $\text{Pai}(t_j)$.
  4. Party 2 sends the set $\{\langle RO(v_j)^{k_2}, \text{Pai}(t_j) \rangle \}_{j \in [n]}$ to Party 1 in shuffled order.

- **Round 3 (Party 1):**
  1. For each item $(RO(v_j)^{k_2}, \text{Pai}(t_j))$ received from Party 2 in Round 2 Step 4, Party 1 double-encrypts the first member of the pair using $k_1$, thus computing $(RO(v_j)^{k_1 k_2}, \text{Pai}(t_j))$.
  2. Party 1 computes the intersection set $J$:
     \[ J = \{ j : RO(v_j)^{k_1 k_2} \in Z \} \]
     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 $S_J$:
     \[ \text{Pai}(pk, S_J) = \text{Pai.Sum}(\{\text{Pai}(t_j)\}_{j \in J}) = \text{Pai}\left(\sum_{j \in J} t_j\right) \]
  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 \( \text{REAL}^{i,\lambda}(\{u_i\}_{i \in [m]}, \{(v_j, t_j)\}_{j \in [n]}) \) 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 \( \text{SIM}_1 \) such that for all security parameters \( \lambda \) and inputs \( \{u_i\}_{i \in [m]}, \{(v_j, t_j)\}_{j \in [n]} \),

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

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 \( \text{SIM}_1 \) in Algorithm 1.

**Algorithm 1:** The simulator for Party 1

**Input:** \( (\lambda, \{u_i\}_{i \in [m]}, n, |J|) \)

**Output:** \( \text{SimView}(P_1) \)

\[
\text{SIM}_1(1^\lambda, \{u_i\}_{i \in [m]}, n, |J|)
\]

(1) Generate key \( k_1 \in G \), and Paillier key-pair \( (pk, sk) \).

(2) Honestly generate and send \( \{\text{RO}(u_i)^{k_1}\}_{i \in [m]} \) 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 \( G \). 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 \( U_2^* = \{h_j\}_{j \in [n]} \) 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 \( G \).

(5) Send \( \{(h_j, \text{Pai}(pk, 0))\}_{j \in [n]} \) in shuffled order as Party 2’s message in Step 4 of Round 2, where each \( \text{Pai}(0) \) 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 \( \text{SIM}_1 \) and a real protocol execution is in Round 2: instead of sending \( \{\text{RO}(u_i)^{k_1,k_2}\} \) and \( \{(\text{RO}(v_j)^{k_1})\} \) as in a real execution, \( \text{SIM}_1 \) 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

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

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.

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.

Hyb\(_{1,i}\) for \( i \in [m - |J|] \): The same as Hyb\(_{1,i-1}\), except with \( \text{RO}(u_r)^{k_1k_2} \) replaced by \( g_r^{k_1} \) in Party 2’s message in Step 2 of Round 2, where \( u_r \) is the lexicographically smallest as-yet-unreplaced element of \( \{u_i\}_{i \in [m]} \setminus \{v_j\}_{j \in [n]} \), and \( g_r \) is a random element of \( \mathcal{G} \).

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

Hyb\(_{2,j}\) for \( j \in [n - |J|] \): The same as Hyb\(_{2,j-1}\), except with \( \text{RO}(v_j)^{k_2} \) replaced by \( h_j \) in Party 2’s message in Step 4 of Round 2, 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 \( \mathcal{G} \).

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

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

- \( \text{RO}(u_{k^*})^{k_1k_2} \) replaced by \( g_{k^*}^{k_1} \) in Party 2’s message in Step 2 of Round 2 and
- \( \text{RO}(v_{k^*})^{k_2} \) 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 \( \text{SIM}_1 \).

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

We first observe that Hyb\(_0\) and Hyb\(_{1,0}\) are indistinguishable by the CPA security of the Paillier encryption scheme. We also observe that the pairs of hybrids (Hyb\(_{1,m-|J|}\), Hyb\(_{2,0}\)), (Hyb\(_{2,n-|J|}\), Hyb\(_{3,0}\)) and (Hyb\(_{3,|J|}\), Hyb\(_{4}\)) are identical.

It remains to show that hybrids of the form Hyb\(_{1,i-1}\), Hyb\(_{1,i}\), Hyb\(_{2,j-1}\), Hyb\(_{2,j}\) and Hyb\(_{3,k-1}\), Hyb\(_{3,k}\) are indistinguishable. We will argue that Hyb\(_{1,i-1}\) and Hyb\(_{1,i}\) are indistinguishable for all \( i \in [m - |J|] \), based on the hardness. We note that hybrids of the form Hyb\(_{2,j-1}\), Hyb\(_{2,j}\) and Hyb\(_{3,k-1}\), 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 Hyb\(_{1,i}\):
Algorithm 2: Simulator for $Hyb_{1,i}$

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

**Output:** $SimView(P_i)$ in $Hyb_{1,i}$

$Sim_{Hyb_{1,i}}(\lambda, i, (g, g^a, g^b, g^c), \{u_j\}_{i \in [m]}, \{v_j\}_{j \in [n]})$

1. Reprogram the Random Oracle using the DDH tuple as follows:
   
   $$\begin{align*}
   RO(u_i) &= g^{r_i} & \text{for } u_i \neq u^*, \\
   &= g^a & \text{for } u_i = u^*, \\
   RO(v_j) &= g^{s_j} & \forall j \in [n]
   \end{align*}$$

   where each $r_i$ and $s_j$ is randomly chosen in the range $[1, |G|)$, and $u^*$ is the newest element replaced with a random one in $Hyb_{1,i}$.

2. Generate key $k_i \in G$, and Paillier key-pair $(pk, sk)$.

3. Send $\{RO(u_i)^{k_i}\}_{i \in [m]}$ 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:
   
   $g_i = g^c$
   
   $= \text{random element of } G$
   
   $= (g^b)^{s_i}$

   where each $g_i$ is randomly chosen in the range $[1, |G|)$, and $g^c$ is the newest element in $Hyb_{1,i}$.

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

5. Send $\{(g^b)^{s_j}, \text{Pai}(0)\}_{j \in [n]}$ in shuffled order as Party 2’s message in Step 4 of Round 2, where each $\text{Pai}(0)$ 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 $Hyb_{1,i}$. 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 $Hyb_{1,i}$: un-replaced messages in Round 2 Step 2 have the form $RO(u_i)^{k_i k_2}$, and messages sent in Round 2 Step 4 have the form $RO(v_j)^{k_2}$, $\text{Pai}(0)$.

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 $g^c$, we have $g^c = g^{ab} = RO(u^*)^b$. From our earlier interpretation of $b$ as $k_2$, this means $g_i^{k_i} = RO(u_i)^{k_i k_2}$. This change is exactly the difference between $Hyb_{1,i-1}$ and $Hyb_{1,i}$. Thus, the output of Algorithm 2 on inputs $i$ and $(g, g^a, g^b, g^{ab})$ is identical to $Hyb_{1,i-1}$.

From the preceding argument, we can infer that if any adversary can distinguish between $Hyb_{1,i-1}$ and $Hyb_{1,i}$, 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, $\text{Hyb}_{i,j-1}$ and $\text{Hyb}_{i,j}$ 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 $\text{SIM}$ such that for all security parameters $\lambda$ and inputs $\{u_i\}_{i \in \mathbb{Z}^m}$, $\{(v_j,t_j)\}_{j \in \mathbb{Z}^n}$,

$$\text{REAL}^{2\lambda}(\{u_i\}_{i \in \mathbb{Z}^m}, \{(v_j,t_j)\}_{j \in \mathbb{Z}^n}) \approx \text{SIM}_2(1^{\lambda}, \{(v_j,t_j)\}_{j \in \mathbb{Z}^n}, m, S_J)$$

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

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

- In Round 1, instead of sending $\{\text{RO}(u_i)^{k_1}\}_{i \in \mathbb{Z}^m}$ as Party 1’s message, $\text{SIM}_2$ instead sends $m$ randomly selected elements of $G$.

- In Round 3, instead of performing the intersection and computing the intersection-sum, $\text{SIM}_2$ 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 $\text{SIM}_2$ and the view of Party 2 in a real execution is in the Round 1 messages. However, the Round 1 messages output by $\text{SIM}_2$ 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, $\text{SIM}_2$ replaces one additional “real” Round 1 message of the form $\text{RO}(u_i)^{k_1}$ with a random element of $G$. 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 $G$.

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 and would affect any intersection-sum protocol. One strategy for resolving this issue would be to compose differential privacy techniques 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

\[ \text{Compute } \{\text{RO}(u_x)\}_j^{k_1} \text{ intersection } J, \]

\[ \text{Paib}(S_j) = \text{Paib}(\sum_{j} t_j + \tau_j) \]

\[ S_j, \text{ Indices } J \]

\[ \text{Output } S_j - \sum_{j} t_j, \text{ the intersection-sum} \]
Reverse Intersection-Sum Protocol

- **Setup:**
  - Both parties agree on a security parameter $\lambda$ and a $G \in \mathcal{G}(\lambda)$, and a user identifier space $\mathcal{U} = \mathcal{U}(\lambda)$.
  - Both parties have access to a Random Oracle $RO : \mathcal{U} \rightarrow G$ that maps user identifiers to random elements of $G$.
  - 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 $G$.
  - Party 2 generates a fresh key-pair $(pk, sk) \leftarrow \text{Pai.Gen}(\lambda)$ 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 $RO(v_j)^{k_2}$.
  2. Party 2 sends $\{(RO(v_j)^{k_2}, \text{Pai}(t_j))\}_{j \in [n]}$ to Party 1 in shuffled order.

- **Round 2 (Party 1):**
  1. For each element $(RO(v_j)^{k_2}, \text{Pai}(t_j))$ 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 $(RO(v_j)^{k_1 k_2}, \text{Pai}(t_j + r_j))$.
  2. Party 1 sends $\{(RO(v_j)^{k_1 k_2}, \text{Pai}(t_j + r_j))\}_{j \in [n]}$ to Party 2 in shuffled order. The (shuffled $j \rightarrow r_j$) 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, $RO(u_i)^{k_1}$.
  4. Party 1 sends the set $\{RO(u_i)^{k_1}\}_{i \in [m]}$ to Party 2 in shuffled order.

- **Round 3 (Party 2):**
  1. For each item $RO(u_i)^{k_1}$ received from Party 1 in Round 2 Step 4, Party 1 double-encrypts the using $k_2$, thus computing $RO(u_i)^{k_1 k_2}$.
  2. Party 2 computes the intersection set $J$:

  \[ J = \{ j : RO(v_j)^{k_1 k_2} \in \{RO(u_i)^{k_1 k_2}\}_{i \in [m]} \} \]

  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 $\text{SIM}_1$ 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


