2017

Many-to-one private set intersection

Keji Hu
Iowa State University

Many-to-one private set intersection

by

Keji Hu

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Wensheng Zhang, Major Professor

Ying Cai

Daji Qiao

Iowa State University

Ames, Iowa

2017

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In this thesis, we first define a new security problem, named mPSI (many-to-one private set interaction), which can find applications in many scenarios where the host of a big database may be queried by a large number of clients who have small-size queries and want to prevent both the intentions and results of their queries from being exposed to others. We also propose a new scheme to solve the mPSI problem. The scheme extends the state-of-the-art oblivious transfer-based one-to-one PSI schemes, but also embeds the innovative ideas of (1) leveraging the collaborations between clients to achieve high computational and communication efficiency, and (2) relying on server-aided secret encryption to hide each client's private information from being exposed to either the server or any other client. Extensive theoretical analysis and experiments have been conducted to evaluate the proposed scheme and compare the scheme with the state-of-the-art, and the results verify the security and efficiency of our proposed scheme.

CHAPTER 1. INTRODUCTION

The problem of private set intersection, or PSI, has been intensively studied since the introduction of secured computation by Yao [14]. The purpose of PSI is to have two parties P_1 and P_2 compute the intersection of their private databases X and Y, without revealing the private information beyond the result of PSI, which is $X \cap Y$. This problem has surged over the last decade when privacy-preserving data query becomes popular.

One example of typical application scenarios of PSI is as follows. The government may establish and maintain a database of criminal records, and allows organizations or individuals to query the database when there is justifiable need. For example, when a company decides to hire some employees, the company can query the database to find out if the candidates have criminal records. To protect the privacy of employees, it is desired that the identities of the candidates are not exposed to the government agency which hosts the database. If the PSI solution is in place, the company, which possesses a list of candidates, and the host of the criminal database can figure out their intersection without exposing any other information.

1.1 Motivation

All of the existing PSI schemes [1, 4, 2, 12, 11, 8, 6] are designed for the two-party scenario, and thus can only support one client to query the database at a time in the afore-described typical application scenarios. Hence, if a large number of clients need to make query to the single database, even that the size of each query (e.g., the list of candidates checked by a company) is typically much smaller than the size of the database, the host of the database (called *server* hereafter) has to go through the whole PSI protocol with each of the users respectively. This could impose heavy, unscalable workload on server. Taking the state-of-the-art oblivious transfer-based PSI schemes [2, 12, 11] as example, the heavy workload is caused by the following reasons.

Communication-wise, the server needs to transfer all the data records, in the forms of random label, to each client respectively. Note that, the labels cannot be broadcast as they are different for different clients. This way, to serve n queries from n clients, even that each query could be very small, the server needs to transfer the labels of all its data records for n times. Furthermore, according to Pinkas et al. [12], for a server with database size of 2^{n_1} and a client with query size of 2^{n_2}, each random label must have at least $n_1 + n_2 + \lambda$ bits, such that the false positive rate of PSI can be lower than $2^{-\lambda}$. In particular, let us consider a scenario where the server has 2^{24} data records (e.g., identities of criminals) in its database, and 100 clients, each having a query size of $16 = 2^4$ (e.g., lists each of 16 candidates). If the security parameter λ is set to 80, each data record needs to be represented by a label of 100 bits. As we have computed, the server will need to transfer 216 megabytes of data for each query, and 21 gigabytes in total for all the 100 queries, which is barely practical.

Computation-wise, the server needs to conduct bit-wise exclusive OR operations to compute each label for a data record.

Considering the same application scenario as above, for each client, 1.8 billion XOR operations must be carried out by the server. Thus, 180 billion XOR operations are needed for all the 100 clients. Although each XOR operation is considered as a cheap operation, a huge number of such operations could be expensive, and labeling the same set of data records once for each client is redundant.

The above observations have motivated us to design a new type of PSI scheme, which ideally can enable one server to process the queries from multiple clients at a time, and thus can reduce the redundancy and hence improve the efficiency in terms of both communication and computation.

More specifically, an experiment is conducted to illustrate how much redundant computation and communication cost we can save by combining multiple queries into one. Let's fix the length of RSA encryption key to be $\kappa = 1024$. We assume the security parameter $\lambda = 80$. For ease of discussion, we compare the computation and communication cost under the assumption that,

Table 1.1 Communication cost comparison at server side

	One-to-one PSI schemes [11]			Many-to-one PSI schemes		
	$n_1 = 24$	$n_1 = 32$	$n_1 = 40$	$n_1 = 24$	$n_1 = 32$	$n_1 = 40$
$n = 64, n_2 = 4$	13.5GB	3.6TB	992TB	216MB	58GB	15.5TB
$n = 16, n_2 = 6$	3.44GB	944GB	252TB	220MB	59GB	15.75TB
$n = 4, n_2 = 8$	896MB	240GB	64TB	224MB	60GB	16TB

Table 1.2 Computation cost comparison at server side

	One-to-one PSI schemes [11]			Many-to-one PSI protocols		
	$n_1 = 24$	$n_1 = 32$	$n_1 = 40$	$n_1 = 24$	$n_1 = 32$	$n_1 = 40$
$n = 64, n_2 = 4$	9.2 seconds	40 minutes	169 hours			
$n = 16, n_2 = 6$	2.4 seconds	10 minutes	41 hours	0.16 second	39.5 seconds	2.6 hours
$n = 4, n_2 = 8$	0.6 second	2.5 minutes	10.6 hours			

there are a single server and n clients in the system. Server has 2^{n_1} in its database. Each client C_i has 2^{n_2} querying records. Table 1.1 and 1.2 shows the communication and computation cost at the server side to answer a single query.

As we may find from the communication cost and communication cost comparison, when number of clients increases, the redundancy in transferring server's database and computation on server side increases. By combining multiple queries from n clients, we expect to save communication cost and computation cost dramatically.

1.2 Our Contribution

In this thesis, first we define a new problem called many-to-one private set intersection (i.e., mPSI problem), which models the scenarios that a server (i.e., the host of one big database) needs to interact with multiple clients (i.e., owners of small databases) to find out the common subsets of data records shared by the server itself and each of the clients, respectively.

Second, we propose an efficient scheme to solve the afore-defined mPSI problem, named mPSI problem. The proposed scheme extends the idea of the state-of-the-art oblivious transfer-based one-to-one PSI schemes [2, 12, 11], to ensure that the server cannot know the secrets of clients from their interactions. Meanwhile, the scheme adopts several novel ideas to make itself more efficient than the one-to-one PSI schemes. In particular, the scheme requires clients to form a group to act like a single virtual client to interact with the server following a protocol similar to an one-to-one PSI protocol. This way, the afore-discussed heavy communication and computational workloads on the server can be significantly reduced; hence high efficiency is accomplished. In addition, a new encryption technique has been proposed in the scheme to prevent curious clients from knowing other clients' query intentions in the collaboration.

Third, we conduct theoretical analysis and experiments to evaluate the performance of mPSI scheme, and compare the performance with that of a state-of-the-art one-to-one PSI scheme [12]. The results verify the improved efficiency of our designed scheme over the one-to-one PSI scheme, in solving the mPSI problem.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2, we briefly review the existing approaches to solve the private set intersection(PSI) problem, and elaborate some state-of-art PSI scheme in details. In Chapter 3, we define the many-to-one PSI problem; Specifically, we give a formal description of the system model, threat model and the design goal of the problem. In chapter 4, we describe the proposed mPSI scheme in details. Chapter 5 demonstrates the result of theoretical and experiment based on performance studies. Chapter 6 gives the formal security analysis of the mPSI scheme. Chapter 7 summarizes the thesis with a discussion of future work.

CHAPTER 2. LITERATURE REVIEW

2.1 Overview of Private Set Intersection Solutions

The private set intersection(PSI) has been studied intensively since the introduction of secured computation by Yao [14]. The approaches to this problem can be mainly classified into the following categories: naive hashing approach, public key encryption based approach, circuit based approach, and OT based approach.

Naive hashing compares data items of P_1 and P_2 in their hash values. It is the most efficient approach but insecure when database is small or the entropy of database is low. Public key encryption based approach [1, 4] applies public key cryptosystem to compare data items in encrypted form. Each party encrypts its own database using a separate public key. Ciphertext from both parties is exchanged, encrypted again and compared. Public key encryption based approach in general requires $O(n)$ asymmetric encryption operations. Circuit based approach [8, 6] builds a function specific garbled circuit that is optimized for PSI operations. For instance, The Sort-Compare-Shuffle circuit [8] requires each party to sort its local database before making comparisons. A function specific garbled circuit was built to merge to sorted list and grab the adjacent elements that are the same. The comparison cost is reduced from $O(n^2)$ to $O(n \log n)$ in general. OT based PSI approach [2, 12, 11] is the state-of-art which is introduced in section 2.3 in details.

2.2 Oblivious Transfer

Oblivious Transfer(OT) protocols [13, 9] allow a sender to send its data to a receiver and the receiver only receives a part of sender's data based on its selection. Both receiver's selection and sender's rest of the data are protected as private information in OT protocols. In general,

OT_l^m denotes an activity of a sender to obliviously transfer m strings, each with a length of l bits. Sender holds m pairs of l-bit strings (L_i^0, L_i^1) as input to OT, while receiver holds an $m-bit$ string $r = (b_1 b_2 \dots b_n)$. As a result of OT_l^m, receiver receives only the set $\{L_i^{b_i}\}_{1 \leq i \leq n}$ while sender not knowing any information about receiver's input r.

2.3 OT-based PSI Schemes

The concept of OT-based PSI is first raised by Dong et al. in the work of [2], and improved in follow-up works [12, 11]. The key abstract of OT-based PSI protocol is to use random labels to anonymize the identity of the data and enable comparison under random labels. Here, we review [12] in detailed steps.

During the execution of OT-based protocol, P_1 and P_2 follows the steps below to compute the set intersection:

- P_1 and P_2 agree on a set of hash functions $H = \{H_1, H_2, \dots, H_h\}$. P_2 creates a hash table T_2 of $R = h(1+\epsilon)n_2$ bins and let P_1 know the value of R. Due to the security parameter ϵ, P_1 does not know the exact size of P_2's database.

- P_1 create a hash table T_1 of R bins. For each word w_i in P_1's database, compute $ind_{(i,k)} = H_k(w_i)\%R$. Append $id(w_i)$ to bin $ind_{(i,k)}$.

- P_2, on the other hand, maps its elements using a Cuckoo hashing with h hash functions. More specifically, For each word w_j in P_2's database, compute $ind_{(j,k)} = H_k(w_j)\%R$. Place $id(w_j)$ at the first empty bin along the index list. In the case when none of those h positions is available, place w_i in a data structure called stash.

- P_1 pads each bin of T_1 to max_β items with a dummy element d_1 and P_2 fills an empty bin with a dummy d_2 in $T_2(d_1 \neq d_2)$. For each bin in the hash table, P_1 and P_2 runs a OT_l^β protocol where P_1 act as a sender and P_2 act as a receiver. P_1 generates β pairs of l-bit random strings, and P_2 takes the identity of words in its database $id(w)$ as the input to OT_l^β.

- P_2 obliviously obtains the labels corresponding to its input in each bin via OT and computes the exclusive-or of the labels. Denote an item in P_2's hash table by $w = b_1 b_2 \dots b_\beta$. P_2 obtains the set of label $\{L_i^{b_i}\}_{1 \le i \le \beta}$ and computes $\oplus_{i=1}^{\beta} L_i^{b_i}$ as the label for w. At the meanwhile, P_1 XORs the labels corresponding to its input in each bin. All non-dummy labels are permuted before sending to P_2.

- P_2 receives the labels from P_2 and computes the intersection of XORed labels in plaintext. The elements in the stash are compared in a separate round. We refer readers to [12] for more details.

A succeeding work [11] exploits the permutation based hashing to further reduce the length of $id(w)$. It encodes the index of a hash table where a word w is placed as a part of identity of w. With the length of $id(w)$ shorten, the communication overhead is reduced.

CHAPTER 3. PROBLEM STATEMENT

In this chapter, we define the many-to-one private set intersection (mPSI) problem in terms of system model, threat model, and design goals.

3.1 System Model

We consider a system that consists of a central server denoted as S, and a large set of clients (say, m clients). Each client in the system is denoted as C_i, where the client's identity i is a distinct positive integer. For the purpose of readability, the clients are labelled as $C_1, \cdots, C_m$.

The server S owns a dataset $\mathcal{D}_0$ that stores s data records denoted as $\{d_{0,i} | i = 0, \cdots, s-1\}$. Each client C_i has a dataset $\mathcal{D}_i$ of c_i data records denoted as $\{d_{i,j} | j = 0, \cdots, c_i - 1\}$, and desires to find out the intersection $\mathcal{D}_0 \cap \mathcal{D}_i$ in a secured manner; that is, no one other than C_i is allowed to know the intersection. Here, we assume that s is much greater than each c_i. Let p be a large prime number, and G_p be a cyclic group over $\mathcal{Z}_p^*$, with $g \in \mathcal{Z}_p^*$ as the generator. We assume each data record can be represented, using some coding function, as an element of G_p, and let β denote the maximum number of binary bits needed to represent any element of G_p. Further, we assume for any two clients C_i and C_j, their data sets do not intersect; that is, $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$.

3.2 Threat Model

We assume that server S and clients $C_1, \cdots, C_m$ are semi-honest. That is, each of them honestly runs the protocol assigned for it to execute, but it may be curious and may take extra actions (called *privacy attacks*) attempting to reveal private information of others. In the privacy attacks, clients may collude, but we assume that the server does not collude with any client.

3.3 Design Goals

For a scheme we design to solve the mPSI problem, we aim to achieve two-fold goals:

- *Privacy Preservation*, which has the following implications:
  - The content of the dataset owned by each of the server and clients can not be exposed to other parties in the system, except for the intersection.
  - The intersection $\mathcal{D}_0 \cap \mathcal{D}_i$ is known only by client C_i, but not the server or the other clients C_j.
- *Efficiency*, which has the following implications:
  - As the server is the one who interacts with all clients and handles queries, which is considered as the bottleneck of the system. Hence, the communication and computational costs of the server should be as low as possible. This is the major performance goal.
  - The communication cost has a higher priority than the computation cost, since the network bandwidth is less expandable than the computational power on server side.
  - It is also desired that the system-wide overall communication and computational overheads, which are imposed on the server or clients, are as low as possible.

CHAPTER 4. mPSI SCHEME: A SOLUTION TO THE mPSI PROBLEM

In this chapter, we present our proposed mPSI scheme to the mPSI problem. Our scheme is run by a server and a group of clients, and facilitate each of the clients to secretly find out the intersection between data sets of itself with the server.

The scheme consists of four phases:

- *Phase I - System Initialization.* During this phase, each of the server and clients conducts initialization for the data set it possesses.

- *Phase II - Encryption of Clients' Data Records.* During this phase, each client encrypts its data records. The encryption process is aided by the server, but neither the plaintext nor the ciphertext of a data record is exposed to the server.

- *Phase III - Clients' Collaborative Formation of Hash Table.* During this phase, each group of clients collaborate to construct a hash table, which is then presented to the server. Like in an oblivious transfer-based one-to-one PSI scheme [12, 11], the hash table is used as an important vehicle to facilitate the oblivious communication between the clients and the server.

- *Phase IV - Client-Server Interaction for PSI Discovery.* This phase follows an interactive framework similar to that used in an oblivious transfer-based one-to-one PSI scheme. In this phase, the server first creates per-bit labels for each entry of the hash table, and obliviously transfers these labels to the clients. Second, each of the server and the clients uses the per-labels to label their data records. Then, the server sends a randomly-permuted set of the labels of all its data records to the clients. Finally, each client scans the labels received from the server to discover the intersection set with its own labels, which exactly corresponds to the intersection set between its own and the server's sets of data records.

The details of these phases are elaborated in the following.

4.1 Phase I - System Initialization

In this phase, the server encrypts its data records based on a secret (i.e., key) randomly selected by itself. Specifically, the server first randomly picks an element e from G_p, and keeps it secret. Then, it encrypts each data record $d_{0,i}$ (where $i = 0, \cdots, s-1$) that it possesses, into $\hat{d}_{0,i} = d_{0,i}^e$, and keeps the encrypted data records.

Each client C_i works as follows to initialize each of its data record $d_{i,j}$. Specifically, it randomly constructs β pairs of RSA public and private keys for the data item, and we denote the key pairs as

$$(k_{i,j,0}^+, k_{i,j,0}^-), \cdots, (k_{i,j,\beta-1}^+, k_{i,j,\beta-1}^-),$$

where each $k_{i,j,t}^+$ is a public key and each $k_{i,j,t}^-$ is a private key for $t = 0, \cdots, \beta - 1$.

The parties (i.e., the clients and the server) need to have secure peer-to-peer communication in mPSI. We assume that, when a pair of parties need to communicate securely, they can set up a secure communication channel, e.g., a TLS/SSL [5] connection.

The clients also need to collaborate by having some information passed along and processed by all the them sequentially. To facilitate such collaboration, the clients shall agree on a sequence following which information will flow through them. In practice, the sequence could be determined based on the order of the users' IP addresses. For simplicity, we denote the sequence as $C_1 \rightarrow C_2 \rightarrow \cdots \rightarrow C_m$.

4.2 Phase II - Encryption of Clients' Data Records

Each client C_i encrypts its data records $\{d_{i,j} | j = 0, \cdots, c_i - 1\}$ as follows:

- First, it randomly picks numbers $\{r_{i,j} | j = 0, \cdots, c_i - 1\}$ from G_p.

- Second, it computes $d'_{i,j} = d_{i,j}^{r_{i,j}}$ for each $d_{i,j}$, and then sends the set $\{d'_{i,j} | j = 0, \cdots, c_i - 1\}$ to the server,
- Upon receiving the set of data, the server computes $d''_{i,j} = (d'_{i,j})^e$ for each $d'_{i,j}$ and returns $\{d''_{i,j} | j = 0, \cdots, c_i - 1\}$ to C_i.
- Finally, C_i computes $\hat{d}_{i,j} = (d''_{i,j})^{1/r_{i,j}}$ for each received $d''_{i,j}$, and records $\{\hat{d}_{i,j} | j = 0, \cdots, c_i - 1\}$.

Note that, the encryption process is aided by the server; however, due to the randomness of number $r_{i,j}$ $(j = 0, \cdots, c_i - 1)$ selected by client C_i, neither the original data record $d_{i,j}$ nor the encrypted data record $\hat{d}_{i,j}$ is exposed to the server.

4.3 Phase III - Clients' Collaborative Formation of Hash Table

In this phase, clients first form groups and then each group can act like a single virtual client to interact with the server following a protocol similar to an oblivious transfer-based one-to-one PSI scheme. Various procedures could be used for clients to find their group mates, and they may utilize the trust built among them during the past experience in the group formation. Since this is not our focus in this thesis, we skip the group formation detail, and assume a group has been formed to include clients C_1, C_2, $\cdots$, C_m, without loss of generality.

Once a group has been formed, the participating clients in the group collaborate as follows to form a hash table.

First, the clients of the group should agree on a pair of public random hash functions $H_0(.)$ and $H_1(.)$. In practice, this may be accomplished in the following manner: each client randomly generates a pair of random strings; all these pairs are concatenated to form a pair of long random strings denoted as IV_0 and IV_1; then, $H_b(x)$ for $b \in \{0, 1\}$ is defined as $SHA3(IV_b || x)$.

Second, each client C_i reports the number of data records that it possesses to every other clients. Based on the reports, each client gets to know the total number of data records $\sum_{i=1}^{m} c_i$ and then agrees on the size of a hash table to be $l = (1 + \rho) \sum_{i=1}^{m} c_i$, where ρ is a system parameter of the scheme.

Third, each client applies the hash functions on its encrypted data records $\{\hat{d}_{i,j}|j = 0, \cdots, c_i - 1\}$ to obtain $\{(h_{i,j,0} = H_0(\hat{d}_{i,j}) \bmod l,\ h_{i,j,1} = H_1(\hat{d}_{i,j}) \bmod l)|j = 0, \cdots, c_i - 1\}$, and distributes this set to every other clients. This way, each client gets to know the hash pairs of all the data records owned by the clients. Based on these hash values, one specific client (assumed to be C_1 without loss of generality) can apply the Cuckoo hashing strategy [10] to map each data record to one entry of the hash table, such that: (1) for each data record $d_{i,j}$, it is mapped to either entry $h_{i,j,0}$ or $h_{i,j,1}$, and (2) no two data records are mapped to the same entry. In the case that mapping conflicts cannot be avoided, some data records will be temporarily skipped to avoid the conflicts, and the skipped records will be processed in an extra round. Then, C_1 broadcasts the final mapping strategy to all the clients in the group.

Fourth, the hash table is passed from C_1 to C_m one by one, to allow each client C_i to fill in its information to the hash table. Specifically, supposing client C_i's data record d is mapped to entry k of the hash table, C_i acts as follows to fill in entry k: Let the binary representation of d be

$$b_0 b_1 \cdot b_{\beta-1},$$

where $b_t \in \{0, 1\}$ for $t = 0, \cdots, \beta - 1$. Let the sequence of RSA public keys picked by C_i for d be

$$k_0^+, \cdots, k_{\beta-1}^+;$$

recall that these keys are selected during the system initialization phase. C_i fills β pairs of public keys, denoted as

$$(PU_{0,0}, PU_{0,1}), \cdots, (PU_{\beta-1,0}, PU_{\beta-1,1})$$

to entry k. In particular, each pair of $PU_{t,0}$ and $PU_{t,1}$ is computed according to the following rules:

- Case I: $b_t = 0$. $PU_{t,0} = k_t^+$. Supposing $PU_{t,0} = (n, x)$, $PU_{t,1} = (n,\ x^{-1} \bmod n)$.
- Case II: $b_t = 1$. $PU_{t,1} = k_t^+$. Supposing $PU_{t,1} = (n, x)$, $PU_{t,0} = (n,\ x^{-1} \bmod n)$.

The key idea of this step is to have the public keys generated during the system initialization phase to be mapped to the bit value of data in each bit position, while the other one must be a random key that the client does not have the corresponding private key. Hence later, when per-bit labels are generated and encrypted by the server the client can only decrypt and get the label corresponding to its bit value at all bit positions.

After C_m has filled in public key pairs for each of its data record, the client also fills every empty entry. Specifically, each empty entry k should be filled with a sequence of β pairs of public keys denoted as $(PU'_{0,0}, PU'_{0,1}), \cdots, (PU'_{\beta-1,0}, PU'_{\beta-1,1})$ that are constructed according to the following rule: For each $t = 0, \cdots, \beta - 1$,

- randomly selects an RSA public key (n', y) and assigns it to $PU'_{t,0}$;
- $PU'_{t,1} = (n',\ y^{-1} \mod n')$.

Finally, the fully-filled hash table, together with hash functions $H_0(.)$ and $H_1(.)$, are sent to the server.

4.4 Phase IV - Client-Server Interactions for PSI Discovery

This phase includes five steps:

- The server randomly creates per-bit labels for every entry of the hash table.
- The server obliviously transfer the per-bit labels to the group of clients.
- The server labels its data records using the previously-created per-bit labels, and sends the data record labels to the clients.
- Each client scans the labels to identify the intersection between its own data records and the server's data records.

The details of the these step are presented in the following.

4.4.1 Server's Random Creation of Per-bit Labels

For each entry j $(j = 0, \cdots, l - 1)$ of the hash table, the server randomly picks β pairs of per-bit labels, denoted as

$$(L^0_{j,0}, L^1_{j,0}); (L^0_{j,1}, L^1_{j,1}); \cdots ; (L^0_{j,\beta-1}, L^1_{j,\beta-1}).$$

Here, each $L^b_{j,t}$ ($b \in \{0,1\}$, $j \in \{0,\cdots,l-1\}$ and $t \in \{0,\cdots,\beta-1\}$) is a γ-bit number, where γ is a security parameter.

4.4.2 Server's Oblivious Transferring of Per-bit Labels to the Clients

For each entry j ($j = 0,\cdots,l-1$) of the hash table, the server encrypts the per-bit labels it picks for the entry and encrypts them with the RSA public keys carried in the entry.

Specifically, each per-bit label pair $(L^0_{j,t}, L^1_{j,t})$ is encrypted to $(enc(PU^0_{j,t}, L^0_{j,t}), enc(PU^1_{j,t}, L^1_{j,t}))$, where $enc(PU, L)$ represents the RSA encryption of L using public key PU. Hence, each entry j of the hash table is changed from

$$(PU^0_{j,0}, PU^1_{j,0}); (PU^0_{j,1}, PU^1_{j,1}); \cdots; (PU^0_{j,\beta-1}, PU^1_{j,\beta-1})$$

to

$$(enc(PU^0_{j,0}, L^0_{j,0}), enc(PU^1_{j,0}, L^1_{j,0}));$$
$$(enc(PU^0_{j,1}, L^0_{j,1}), enc(PU^1_{j,1}, L^1_{j,1}));$$
$$\cdots;$$
$$(enc(PU^0_{j,\beta-1}, L^0_{j,\beta-1}), enc(PU^1_{j,\beta-1}, L^1_{j,\beta-1})).$$

After the replacement, server sends the hash table to C_1, and the hash table is passed along the group of clients one by one. When a client C_i receives the table, it works on each entry j where it has filled RSA public keys. Without loss of generality, suppose C_i filled this entry because $H_0(d) \bmod l = j$, where d is its data record, and C_i desires to find out if d is in the data set of the server. Also, assume d can be represented in a string of binary bits as $b_0 b_1 \cdots b_{\beta-1}$.

More specifically, C_i works on entry j in the following two steps:

- *Obtaining Labels.* According to the algorithm by which C_i computed the RSA public keys filled to entry j, C_i should have known the public keys

  $$PU^{b_0}_{j,0}, PU^{b_1}_{j,1}, \cdots, PU^{b_{\beta-1}}_{j,\beta-1},$$

  and their corresponding RSA private keys the we denote as

  $$PR^{b_0}_{j,0}, PR^{b_1}_{j,1}, \cdots, PR^{b_{\beta-1}}_{j,\beta-1}.$$

Hence, C_i can decrypt the entry to obtain the following labels:

$$L_{j,t}^{b_t} = dec(PR_{j,t}^{b_t}, enc(PU_{j,t}^{b_t})),$$

where $t = 0, \cdots, \beta - 1$ and $dec(PR, x)$ denotes the decryption of x using RSA private key PR.

- *Labeling Data Records.* Upon obtaining the labels, C_i further labels its data record d to

$$L(d) = \oplus_{t=0}^{\beta-1} L_{j,t}^{b_t}.$$

4.4.3 Server's Labeling of Data Records and Sending of Labels

After randomly generating labels, the server also labels all its data records.

For each data record d, which is $b_0 b_1 \cdots, b_{\beta-1}$ if represented as a binary string, it is mapped to entries $H_0(d) \mod l$ and $H_1(d) \mod l$ through using hash functions $H_0(.)$ and $H_1(.)$. For each entry $j \in \{H_0(d) \bmod l, H_1(d) \bmod l\}$, one label of d is computed as

$$L(d) = \oplus_{t=0}^{\beta-1} L_{j,t}^{b_t},$$

similar to the way in which the clients label their data records.

Once every data record has been labeled, all these labels of data records within the same entry are randomly shuffled, and then broadcast to the group of clients in the block of entries. Random dummy labels are generated to make the number of data items in each entry uniform.

4.4.4 Client's Discovering of Set Intersection

Upon receiving the broadcast labels, each client C_i in the group scan the labels to find the overlapping with the labels of its own data records. The set of data records whose labels overlap is then identified as the intersection between the data sets of this client and the server.

Note that, since the labels are grouped in the unit of entries, each client only look for labels at the entries where the client has placed its queries to. For instance, without loss of generality, suppose C_i has a data item $d_{i,j}$ mapped to entry k, i.e., $H_0(d_{i,j}) = k$ (if C_i uses $H_0(.)$ for $d_{i,j}$), the overlapping of the label about $d_{i,j}$ will only happen in the k^{th} entry of the labels broadcast by the server. That saves a great amount of communication overhead on the client side.

CHAPTER 5. PERFORMANCE EVALUATION

We conduct both the theoretical analysis and the experiment to evaluate the performance of our proposed mPSI scheme. In this chapter, we present the result of both asymptotic computation and communication cost, along with the experiment result. We also simulate one of the state-of-art one-to-one PSI protocol [12], and make comparison between mPSI and the work of [12] under the same experimental setting. Our results indicate a significant advantage of using many-to-one PSI scheme over one-to-one PSI scheme in the multi-client setting.

5.1 Asymptotic Analysis

First, we conduct theoretical analysis to obtain the asymptotic communication and computational costs of the proposed mPSI scheme, and compare with the state-of-the-art one-to-one PSI scheme [12]. For the simplicity of discussion, we assume all clients have the equal database size, denoted by c. Table 5.1, 5.2 and 5.3 demonstrate the result of comparison. As we can find from the tables, mPSI incurs lower server-side communication and computational costs than the one-to-one PSI scheme, due to the integration of clients' datasets; the improvement becomes more significant when s (i.e., the size of the server's dataset) or m (i.e., the number of clients in a group) increases. Meanwhile, mPSI incurs slightly higher communication and computational costs at the client side, than the one-to-one PSI scheme, due to the overheads caused

Table 5.1 Bandwidth Consumption by The Server and Each Client.

	Server	Each Client
1-to-1 PSI	$O(m \cdot s \cdot \gamma + m \cdot c \cdot \beta \cdot \gamma)$	$O(s \cdot \gamma + c \cdot \beta \cdot \gamma)$
mPSI	$O(s \cdot \gamma + m \cdot c \cdot \beta \cdot \gamma)$	$O(s \cdot \gamma + m \cdot c \cdot \beta \cdot \gamma)$

Table 5.2 One-to-One PSI Computational Cost.

	Server	Each Client
RSA Key Generation	–	$O(c \cdot \beta)$
RSA Encryption	$O(m \cdot c \cdot \beta)$	–
RSA Decryption	–	$O(c \cdot \beta)$
XOR	$O(s \cdot \beta \cdot \gamma)$	$O(c \cdot \beta \cdot \gamma)$

Table 5.3 mPSI Computational Cost.

	Server	Each Client
RSA Key Generation	–	$O(c \cdot \beta)$
RSA Encryption	$O(m \cdot c \cdot \beta)$	–
RSA Decryption	–	$O(c \cdot \beta)$
XOR	$O(s \cdot \beta \cdot \gamma)$	$O(c \cdot \beta \cdot \gamma)$
Modulus EXP	$O(m \cdot c + s)$	$O(c)$

by the collaborations among the clients. We argue that, the tradeoff is beneficial, because it is the server, not the clients, that is the bottleneck of the system. More significantly, when the size of databases are asymmetric, i.e., when $c \ll s$(the application scenarios we mentioned in introduction), the saving in server side over-weighs the overhead on each client.

5.2 Experiment Results

First, we carried out a benchmark for basic cryptographic operations used in the mPSI scheme. This includes the RSA key generation($KeyGen$), encryption(Enc), decryption(Dec).

Table 5.4 shows the benchmark results of the three basic crypto operations used in mPSI scheme on the test machine. From the results, we can find that RSA key generation is computationally more intensive than RSA encryption and decryption. Since RSA key generation is largely used during the system initialization, we predict that phase I of mPSI scheme will consume most of the time in our simulation.

To evaluate the performance of our proposed scheme in more practical settings, we also implement the both mPSI and the one-to-one PSI scheme [12] in Java, and conduct experiments

Table 5.4 Time to perform repeated operations for 1000 times (unit: second)

Key Length	$KeyGen$	Enc	Dec
512 bits	9.5	0.024	0.22
1024 bits	47.2	0.068	1.13
2048 bits	371.1	0.181	6.1

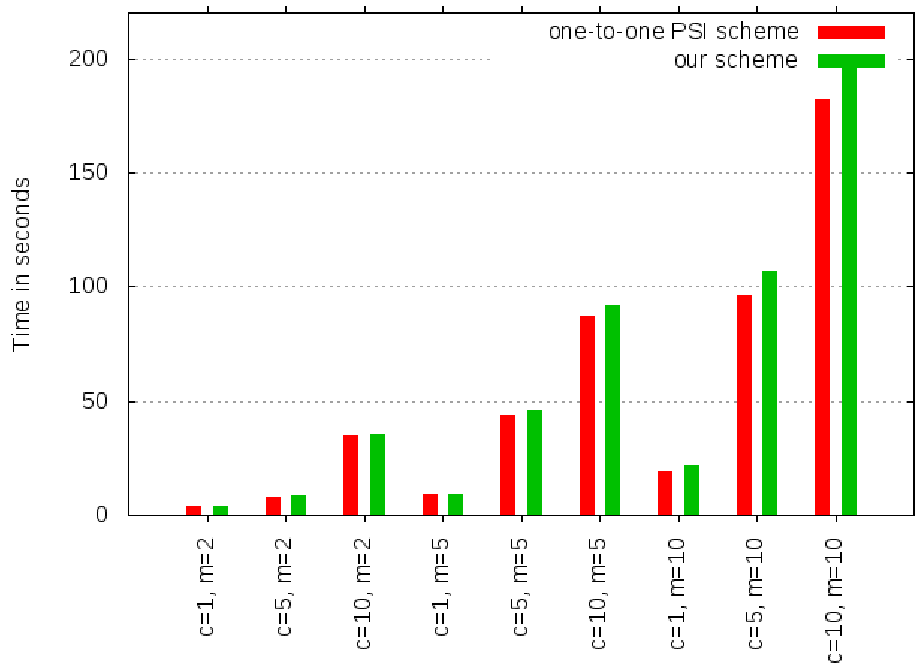

Figure 5.1 Client side computation comparison between one-to-one PSI scheme and our scheme.

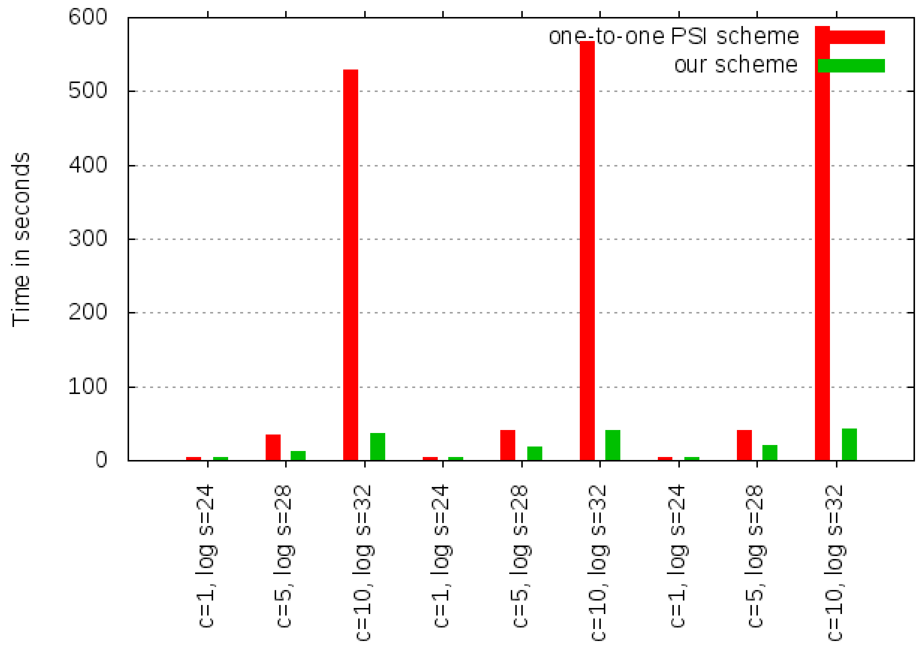

Figure 5.2 Server side computation comparison between one-to-one PSI scheme and our scheme.

to compare their computational costs. In the experiments, we use the computes with the same configurations for the server and each client: the computer has a CPU clocking @2.5GHz, with 8 gigabyte of RAM and solid state as the hard drive.

For the purpose of easy comparison, we fix the length of RSA keys to be 1024 and the data length $\beta = 40$, and make client's database size c, number of users in the system, m, and the server's database size s, to be variables. We assume the all clients have the same database size, which is c.

The results are reported in Figures 5.1, 5.2. As we can see, the experimental results are consistent with the above theoretical analysis results.

CHAPTER 6. SECURITY ANALYSIS

In this chapter, we conduct the security analysis of our proposed mPSI scheme. The mPSI scheme is secure if it has the following three properties

- Security Property I: Server cannot infer any information about the data item of client C_i's database except for the intersection.
- Security Property II: Any client C_i cannot infer any information about data items of server's database except for the intersection.
- Security Property III: Any client C_i cannot infer any information about data items of C_j's database for $i \neq j$.

The first two security properties are the same as the security properties of one-to-one PSI schemes. The third security property is unique for many-to-one PSI protocols because our mPSI involves multiple clients in the system, and each client C_i's database information is kept secret from other clients C_j. In this chapter, we discuss the security proofs for each of the security property mPSI is intended to achieve.

6.1 Security Property I

To prove that server cannot infer any information about client C_i's database, we divide the scheme into two stages:

- The first stage includes the phases of scheme before the submission of hash table to server, i.e., phase I, II, and III.
- The second stage includes the phase of server-client interaction to find out the intersection, i.e, phase IV.

During the first stage, the view from a server about clients' database, is a collection of data records encrypted with clients' random secret. That is, $View_S = \{d'_{i,j} = d_{i,j}^{r_{i,j}} | i = 1, 2, \cdots, m, j = 0, 1, \cdots, c_i - 1\}$. The secret $r_{i,j}$ is uniformly selected from the finite field $\mathcal{Z}_p^*$. Then, the inverse $[d'_{i,j}]^{1/r_{i,j}}$ is uniformly distributed over $\mathcal{Z}_p^*$. Hence, the probability for the adversary's algorithm A to correctly guess $d_{i,j}$ based on $View_S$ is $1/p$, that is, $Prob[A(View_S) = d_{i,j}] = 1/p$.

In the second stage, clients in the system act like a virtual client and interact with the server in the way defined by one-to-one OT-based PSI schemes. The security proof of this part is exactly the same as that of [12].

Combining the proof of stage one and two, we conclude that server cannot infer information about client's database, except for the intersection.

6.2 Security Property II

To prove that client cannot infer and information about server's database, we first take a look at the interactions between clients and server in the mPSI scheme. When interacting with the server, all clients in the mPSI scheme act like a virtual client and send queries together in a hash table. This makes the proof of the second security goal to be exactly the same as OT-based one-to-one PSI schemes. Here, we refer readers to [12] for the security proof of the second security goal.

6.3 Security Property III

To prove that mPSI scheme achieves the third security goal, we first give an intuition about our overall approach, and then we define the behavior of an adversary client. after which we give a formal security definition with a proof.

6.3.1 Intuition for the Security Analysis

Intuitively, the view of an adversary client is a set of plaintext-ciphertext pairs that it can query the server for intersection. Let's denote the set as $Q = \{\langle x_i, x_i^e \rangle | i = 0, \cdots, c - 1\}$. The purpose of an adversary client is to figure out that, if an arbitrary data item y is in an innocent client C_i's database.

An adversary algorithm A is defined in section 6.3.2 based on the view of an adversary client to determine if $H(y^e) = z$, where z is an index of the hash table that an innocent client has query in.

Given this defined algorithm A, we formally layout a theorem and a proof to reduce adversary's algorithm A to matching Deffie-Hellman problem(MDHP) [3, 7].

6.3.2 Behavior of An Adversary Client

Here, we model the behavior of the adversary as algorithm $A(\{\langle x_i, x_i^e\rangle | i = 0, \cdots, c-1\}, H(\cdot), y, l, z)$, where

- s is randomly picked from $\mathcal{Z}_p^*$, which is the secret kept by the server;
- each x_i is randomly picked from G_p, which reprents a data item queried by the adversary client or its colluding coalition;
- y is randomly picked from G_p, which represents the data item that the adversary wants to figure out if is queried by an innocent client;
- l is an even number that represents the number of entries in the hash table;
- $H(\cdot) : G_p \to \{0, \cdots, l-1\}$ is a random hash function used to create the hash table; and
- $z \in \{0, \cdots, l-1\}$ represents an entry index of the hash table.

If the attack succeeds, the output of algorithm A is:

- *false* iff $H(y^e) \in \{z + 1 \bmod l, \cdots, z + l/2 \bmod l\}$; or
- *true* otherwise.

6.3.3 Theorem and Proof

In this section, we prove that our mPSI scheme is secure from an adversary client by reducing the adversary algorithm A to the MDHP problem.

First, we restate a formal definition of MDHP problem [3, 7].

Definition. *In G_p, given $(g^{a_0}, g^{a_0 b_0}, g^{a_1}, g^{a_1 b_1})$ and $(g^{b_r}, g^{b_{1-r}})$, where $a_0, b_0 \in \mathcal{Z}_p^*$ and $r \in \{0, 1\}$, find r.*

Theorem. *Suppose the adversary client succeeds with an advantage of ϵ. That is, $Adv(A) = \epsilon$, or more specifically,*

$$Pr[A(\{\langle x_i, x_i^e\rangle | i = 0, \cdots, c-1\}, H(\cdot), y, l, z) \equiv (H(y^e) \notin \{z+1 \; mod\, l, \cdots, z+l/2 \; mod\, l\})] = 0.5+\epsilon,$$

with time complexity $\Theta(t)$. Then, the MDHP problem can be solved with Algorithm $\hat{A}$, which is defined as follows, with advantage 0.5ϵ and time complexity $\Theta(t)$.

In order to prove the theorem, we need to construct an algorithm $\hat{A}$ based on the adversary algorithm A to solve the MDHP problem. In the following, we first describe $\hat{A}$, and then present the complete proof.

6.3.3.1 Construction of Algorithm $\hat{A}$

First, based on g^{a_0} and g^{b_r}, the input to one running of algorithm A can be constructed as in the following.

Numbers $r_{0,0}, \cdots, r_{0,c-1}$ are randomly picked from $\mathcal{Z}_p^*$. Let $e_0 = a_0$. For each $i \in \{0, \cdots, c-1\}$, $x_i = g^{r_{0,i}}$ and $x_i^{e_0} = (g^{a_0})^{r_{0,i}}$. Let $y = g^{b_r}$.

Let the output of this running of A be ϕ_0.

Second, based on g^{a_1} and g^{b_r}, the input to another running of algorithm A can be constructed as in the following.

Numbers $r_{1,0}, \cdots, r_{1,c-1}$ are randomly picked from $\mathcal{Z}_p^*$. Let $e_1 = a_1$. For each $i \in \{0, \cdots, c-1\}$, $x_i = g^{r_{1,i}}$ and $x_i^{e_1} = (g^{a_0})^{r_{1,i}}$. Let $y = g^{b_r}$.

Let the output of this running of A be ϕ_1.

Further, let us define a boolean function $\tau(x, y)$ with $x \in G_p$ and $y \in \{0, \cdots, l-1\}$ as

$$\tau(x, y) \equiv H(x) \in \{y - (l/2 - 1) \text{ mod } l, y - (l/2 - 2) \text{ mod } l, \cdots, y\}.$$

Third, based on the output of the two runs of algorithm A, the value of r (i.e., the solution to the MDHP problem) is determined according to the rules represented by the table 6.1.

Table 6.1 Rules to determine r

$\phi_0 = \tau(g^{a_0 b_r}, z)$	$\phi_1 = \tau(g^{a_1 b_r}, z)$	r
true	false	0
false	true	1
false	false	0 or 1 equally possible
true	true	0 or 1 equally possible

6.3.3.2 Proof

First, construct algorithm $\hat{A}$ as described in section 6.3.3.1. Next, let us analyze the probability that the value of r can be corrected obtained using the above rule.

- *Case I: r=0, which occurs with probablity 0.5.* In this case, if the first running of A produces correct output (which occurs with probability $0.5+\epsilon$ according to the assumption of A's advantage), $\phi_0 = \tau(g^{a_0 b_r}, y)$; otherwise (which occurs with probability $0.5 - \epsilon$), $\phi_0 \neq \tau(g^{a_0 b_r})$. Meanwhile, ϕ_1 could be equal to or not equal to $\tau(g^{a_1 b_r}, y)$ with the same probability of 0.5. Hence, according to Table 6.1, we have the following two tiers of subcases:

  - *Subcase I-1: the first running of A produces correct output.*

    - *Subcase I-1-a: $\phi_1 = \tau(g^{a_1 b_r}, y)$.* In this subcase, $\hat{A}$ outputs 0 or 1 with equal probability.

    - *Subcase I-1-b: $\phi_1 \neq \tau(g^{a_1 b_r}, y)$.* In this subcase, $\hat{A}$ outputs 0.

    Note that, either of the above subcases (i.e., I-1-a and I-1-b) occurs with the probability of $0.5 \cdot (0.5 + \epsilon) \cdot 0.5$.

  - *Subcase I-2: the first running of A produces incorrect output.*

    - *Subcase I-2-a: $\phi_1 = \tau(g^{a_1 b_r}, y)$.* In this subcase, $\hat{A}$ outputs 1.

    - *Subcase I-2-b: $\phi_1 \neq \tau(g^{a_1 b_r}, y)$.* In this subcase, $\hat{A}$ outputs 0 or 1 with equal probability.

    Note that, either of the above subcases (i.e., I-2-a and I-2-b) occurs with the probability of $0.5 \cdot (0.5 - \epsilon) \cdot 0.5$.

- *Case II: r=1, which occurs with probability 0.5.* The analysis in this case is similar to Case I. If the second running of A produces correct output (which occurs with probability $0.5+\epsilon$ according to the assumption of A's advantage), $\phi_1 = \tau(g^{a_1 b_r}, z)$; otherwise (which occurs with probability $0.5-\epsilon$), $\phi_1 \neq \tau(g^{a_1 b_r}, z)$. Meanwhile, ϕ_0 could be equal to or not equal to $\tau(g^{a_0 b_r}, z)$ with the same probability of 0.5. Hence, according to Table 6.1, we have the following two tiers of subcases:

  – *Subcase II-1: the second running of A produces correct output.*

    ∗ *Subcase II-1-a:* $\phi_0 = \tau(g^{a_0 b_r}, z)$. In this subcase, $\hat{A}$ outputs 0 or 1 with equal probability.

    ∗ *Subcase II-1-b:* $\phi_0 \neq \tau(g^{a_0 b_r}, z)$. In this subcase, $\hat{A}$ outputs 1.

    Note that, either of the above subcases (i.e., II-1-a and II-1-b) occurs with the probability of $0.5 \cdot (0.5+\epsilon) \cdot 0.5$.

  – *Subcase II-2: the second running of A produces incorrect output.*

    ∗ *Subcase II-2-a:* $\phi_0 = \tau(g^{a_0 b_r}, z)$. In this subcase, $\hat{A}$ outputs 0.

    ∗ *Subcase II-2-b:* $\phi_1 \neq \tau(g^{a_0 b_r}, z)$. In this subcase, $\hat{A}$ outputs 0 or 1 with equal probability.

    Note that, either of the above subcases (i.e., II-2-a and II-2-b) occurs with the probability of $0.5 \cdot (0.5-\epsilon) \cdot 0.5$.

To summarize the above analysis, Algorithm $\hat{A}$ outputs a correct r occurs with the probability of $0.5 + 0.5\epsilon$. Hence, the advantage for $\hat{A}$ to solve the MDHP problem is 0.5ϵ.

Therefore, the theorem is proved.

CHAPTER 7. CONCLUSION

7.1 Summary

In this thesis, we have defined a new security problem, named mPSI problem, to model the application scenarios where the host of a big database may be queried by a large number of clients who have small-size queries and want to protect both the intentions and results of their queries. We have also proposed a new scheme to solve the mPSI problem. The scheme extends the state-of-the-art oblivious transfer-based one-to-one PSI schemes, but also embeds the innovative ideas of (1) leveraging the collaborations between clients to achieve high computational and communication efficiency, and (2) relying on server-aided secret encryption to hide each client's private information from being exposed to either the server or any other client. Extensive theoretical analysis and experiments have been conducted to evaluate the performance of the proposed scheme and compare the scheme with the state-of-the-art, and the results verify the efficiency of our proposed scheme.

7.2 Future Works

More research efforts are demanded for this new research problem. For example, our proposed scheme relies on the assumption that the server does not collude with any client. In the future work, we plan to develop more secure schemes without such assumption. In addition, our mPSI scheme requires server to pre-compute the encryption of its database per-round of communication. As the server's database size gets larger, server side cost(even if it is pre-computation) will be high. More researches are demanded to look for a replacement of database encryption on server's side.

BIBLIOGRAPHY

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97. ACM, 2003.

[2] C. Dong, L. Chen, and Z. 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.

[3] Y. Frankel, Y. Tsiounis, and M. Yung. indirect discourse proofs: Achieving efficient fair off-line e-cash. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 286–300. Springer, 1996.

[4] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-EUROCRYPT 2004*, pages 1–19. Springer, 2004.

[5] A. Freier, P. Karlton, and P. Kocher. The secure sockets layer (ssl) protocol version 3.0. 2011.

[6] S. Goldwasser, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with an honest majority. In *Proc. of the Nienteenth Annual ACM STOC*, volume 87, pages 218–229, 1987.

[7] H. Handschuh, Y. Tsiounis, and M. Yung. Decision oracles are equivalent to matching oracles. In *International Workshop on Public Key Cryptography*, pages 276–289. Springer, 1999.

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

[9] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

[10] R. Pagh and F. F. Rodler. Cuckoo hashing. In *European Symposium on Algorithms*, pages 121–133. Springer, 2001.

[11] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *Proceedings of the 24th USENIX Security Symposium*, 2015.

[12] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on ot extension. In *Usenix Security*, volume 14, pages 797–812, 2014.

[13] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[14] A. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.