A Scalable Multi-Party Protocol for Privacy-Preserving Equality Test

Maryam Sepehri, Stelvio Cimato, and Ernesto Damiani

Dipartimento di Informatica, Università Degli Studi di Milano
Crema, Italy
{firstname.lastname}@unimi.it

Abstract. Multi-party computation (MPC) is attractive for data owners who are interested in collaborating to execute queries without sharing their data. Since data owners in MPC do not trust each other, finding a secure protocol for *privacy-preserving* query processing is a major requirement for real world applications. This paper deals with *equality test* query among data of multiple data owners without revealing anyone's private data to others. In order to nicely scale with large size data, we show how communication and computation costs can be reduced via a *bucketization* technique. Our bucketization requires the use of a trusted third party (TTP) only at the beginning of the protocol execution. Experimental tests on horizontally distributed data show the effectiveness of our approach.

Keywords: secure multi-party computation, equality test query, privacy-preserving query processing and bucketization.

1 Introduction

In today's collaborative environments there is an increasing need of performing computations over data held by multiple owners without sharing the data themselves. This requirement is the subject of an important area of research, called *privacy-preserving* computation. An important selection of privacy-preserving computations consists in executing queries over partitioned databases while keeping privacy. For instance, *equality test* queries [1, 2] selecting items in a database that are equal to a given value, or *range* queries [1–3], selecting values in a fixed range. In partitioned databases, owners can be required to execute even more complex operations like *intersection*, which computes the set of items common to all the private databases. Secure multi-party functions are often represented as combinational circuits [4, 5]. This solution is quite efficient in two-party case but does not scale to the multi-party case. Goldreich et al. [5] showed that a multi-party version of Yao's protocol needs a quadratic communication complexity in the number of parties. Over the years, the research community has developed a wide range of privacy-preserving techniques to realize different functions across the shared databases based on *homomorphic encryption*, *oblivious polynomial evaluation* and *commutative encryption* methods [6–10]. Homomorphic encryption has been deployed in [10] where a solution for two-party equality has been presented. Assuming that the receiver and the sender have a private set of s elements, the computation and communication complexity amounts to $O(2s^2 lgN)$, where N is the RSA-modulus. The homomorphic encryption method is not straightforwardly extendable to multi-party case because if multi-party equality test runs in pairs, each party will learn the intersection with the remaining ones. In [11], the authors present a solution based on oblivious polynomial evaluation (OPE) protocol. The protocol requires each party to perform s oblivious evaluations of a polynomial of degree n with communication cost $O(n)$. For this reason, this protocol is considered too expensive to implement in the multi-party setting. Li and Wu [12] propose a protocol that involves a TTP to compare the values held by two other parties using homomorphic encryption. Although this protocol is faster than OPE, it has two main drawbacks: 1) the TTP should be trusted by all parties; 2) when the number of parties increases, the solution does not scale because of the communication and computation bottleneck created at TTP. The set intersection problem is to compute the intersection between these sets without revealing any other information. Freedman, Nissim and Pinkas [9] propose a protocol based on the Paillier's cryptosystem mentioned above. According to [13], the average computation cost for each party and the communication cost among all parties are $O(s(s + m)lgN)$ and $10s(m - 1)^2 lgN$, respectively. Following Freedman's protocol, Kissner and Song [8] designed a technique for set intersection problem using polynomial representations with $O(cs^2 lgN)$ computation complexity and $2cm(5s+2)lgN$ communication complexity, where c is a suitable constant. Sang et al. [13] adopt a distinct through related approach with lower computation and communication costs with respect to [9, 8] with $O(csmlgN)$ computation cost for each party and $2cm(4s + 5)lgN$ communication cost. A major step forward was taken by Agrawal et al. [7], who introduce a two-party protocol for set intersection based on commutative encryption with linear complexity. The protocol has been extended to multi-party case [14] with lower complexity than [9, 8, 13], i.e., $O(smlgN)$ computation and $((m - 1)s + (m - 2)s + 1 + (m - 1))lgN$ communication overhead. However, Vaidya's [14] extension only computes the size of the intersection, without identifying the intersection's elements. Most of these techniques offer strong privacy guarantees, but they do not scale well for large databases because of using heavy-weight cryptographic operations or several rounds of interactions among parties.

A central issue in this context is designing a protocol for the **S**ecure **M**ulti-party **E**quality test **P**roblem (SMEP), which is secure and efficient when the number of parties and the size of data increases. In this paper we present a new protocol, B-SMEQ (**B**ucketized **S**ecure **M**ulti-party protocol for **E**quality test **Q**ueries) to address the SMEP problem, which adopts a *bucketization* technique to reduce the time complexity. Our solution uses a commutative encryption scheme to avoid information being revealed among data owners. In order to make the protocol fast, we divide data into buckets so that work can be done considering only a subset of data. To realize the bucketization scheme, TTP is involved in an initial phase. However, the TTP does not participate in the query processing, avoiding the creation of computation and communication bottleneck.

2 Problem Statement

We consider a Multi-party System$MS = \langle D, m, U, q, R\rangle$ for equality test computation involving five different entities as illustrated in Fig. 1: a common database D with n records, which has been horizontally partitioned among a set of distrustful data owners O in which $|O| = m$; a set of users U allowed to search; an equality test query q and the result R of the query computation across multiple partitions. The data owners $O = \{O_1, ..., O_m\}$ hold data partitions $\{T_1, ..., T_m\}$, respectively and we assume that all partitions include a searchable attribute A with a set of values V_A, and another attribute B with a set of values V_B. Moreover, for every $i \in \{1, ..., m\}$ we denote with $T_{i,A}$ the column corresponding to attribute A, and for each $v \in T_{i,A}$, with $ext_i(v)$ each value occurring in V_B where $T_{i,A} = v$.

In SMEP, the m data owners must jointly compute the result R to the equality test query q and return it to an authorized user $u \in U$ without revealing their private data to each other, satisfying: *data privacy* property, since the user should learn just the result of the query; the *query privacy* property, since the data owners should not learn the query; and *query anonymous result,* since the user should not know whom the results belong to. In our model, the data owners are honest but curious, meaning that they follow the protocol as exactly specified, but they try to learn extra information by analyzing the transcript of messages received during the execution. Owners are arranged in a ring topology communicating each other via secure channels; one data owner is designated as the master site (*initiator*)[1].

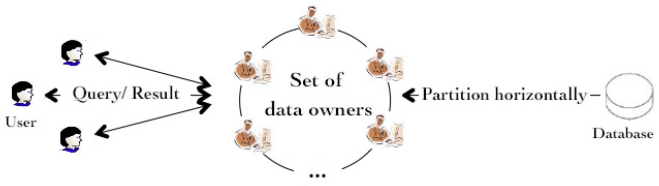

Fig. 1. Basic Scenario

2.1 Preliminaries

Commutative Encryption Scheme. Informally, a commutative encryption is a pair of encryption functions f and g such that $f(g(v))$=$g(f(v))$. By using the combination $f(g(v))$ to encrypt v, we can ensure that no data owner can perform encryption without the help of other data owners. For a more formal definition of commutative encryption scheme, we remand the reader to [7].

Set Intersection Protocol. The set intersection protocol proposed by Agrawal et al. [7] runs between two parties S(sender) and R(receiver), holding a set of values V_S, V_R and keys k_s, k_r, respectively.

[1] Since user authentication and access control are not the main focus in this paper, we assume the authorization between the data owners and users are appropriately managed.

Step 1. Both S and R apply hash function h to their own values and encrypt the result with their secret keys, $f_{k_s}(h(V_S))$ and $f_{k_r}(h(V_R))$.
Step 2. Sites S and R exchange $f_{k_s}(h(V_S))$ and $f_{k_r}(h(V_R))$ after randomly reordering their values to prevent possible inference attacks.
Step 3. Site S encrypts each value of the set $f_{k_r}(h(V_R))$ with k_s to obtain $Z_R = f_{k_s}(f_{k_r}(h(V_R)))$ and sends back the pairs ($f_{k_r}(h(V_R)), Z_R$) to R.
Step 4. Site R creates pairs (v,Z_R) by replacing $f_{k_r}(h(V_R))$ with the corresponding v where $v \in V_R$.
Step 5. Site R selects all v for which $Z_R \in f_{k_r}(f_{k_s}(h(V_S)))$ for $V_S \cap V_R$.

3 A Protocol for Equality Test

In this section, we extend Agrawal's protocol for solving the SMEP problem. Then, we introduce an improvement to ensure scalability. In the basic version, called SMEQ, we extend set intersection protocol from two-party to multi-party setting, where the role of sender is played by each of m data owners. SMEQ is simple to describe and implement but suffers from high communication and computation. We address these issues in an improved version protocol, B-SMEQ, which makes use of bucketization technique on the searchable attribute.

3.1 Protocol 1:SMEQ

Input. A multi-party system (see Section 2) with the data owner O_1 as initiator[2].
Output. The result of user query q to partitions $\{T_1, \ldots, T_m\}$ where $[(T_{1,A} = v) \vee \ldots \vee (T_{m,A} = v)]$.

Step 1. Both user (receiver side, R) and data owners O (sender side, S) apply a hash function h to the value v and to the set of their values of attribute A, respectively. Let $v' = h(v)$ be the result of hashing from the receiver side and let $T'_{i,A} = h(T_{i,A})$, for each $i \in \{1, \ldots, m\}$, be the hashing of the set values $T_{i,A}$. Sites S and R randomly choose a secret key, k_r for R and $\langle k_i, k'_i \rangle$ for data owner O_i.
Step 2. R spans the encrypted hash value y_R=$f_{k_r}(v')$ to all data owners at site S.
Step 3. At site S, each data owner O_i, $1 \le i \le m$, does the following:
3.1 Computes $f_{k_i}\left(T'_{i,A}\right) = Y_i = \{y_i = f_{k_i}(x) | x \in T'_{i,A}\}$
3.2 Generates a set of new keys, one for each value of attribute B, as K_i^B=$\{k_{ix} = f_{k'_i}(x) | x \in T'_{i,A}\}$
3.3 Encrypts each value x in $T_{i,B}$ with the corresponding key k_{ix} to obtain $Y_i^B = \{E_{k_{ix}}(u) | u \in T_{i,B}\}$ where E is an encryption function, which can be efficiently inverted given key (k_{ix})

[2] Any data owner (for instance, the one holding the largest data partition) can be chosen as initiator.

3.4 Computes $I_i = f_{k'_i}(y_R)$ for the purpose of decrypting the values of attribute B at site R

3.5 Owner O_i randomly reorders the tuples Y_i and Y_i^B and sends them along with I_i to the next owner $O_{(i \ mod \ m)+1}$

3.6 Data owner $O_{(i \ mod \ m)+1}$ encrypts only Y_i with the key $k_{(i \ mod \ m)+1}$ and sends the triple $\langle f_{k_{(i \ mod \ m)+1}}(Y_i), Y_i^B, I_i \rangle$ after reordering to the next participant in the ring.
This process is repeated until Y_i is encrypted by all keys of m data owners, obtaining $Z_i = f_{k_1}(f_{k_2}(...(f_{k_m}(Y_i))))$, up to a permutation of the encryption keys[3]

Step 4. Each data owner O_i sends $\langle Z_i, Y_i^B, I_i \rangle$ to owner O_1.

Step 5. Owner O_1 receives all tuples from Step 4 in order to initiate a two-party Agrawal's set intersection protocol (see Section 2.1) between the user as a receiver site and the initiator as a sender site.

Step 6. Owner O_1 passes y_R through the ring in order to have it encrypted by all keys $k_1, \ldots, k_m$ for obtaining $y'_R = f_{k_1}(\ldots(f_{k_m}(f_{k_r}(v'))))$, and then sends back y'_R together with $\langle Z_i, Y_i^B, I_i \rangle$ to the user, for all $i \in \{1, \ldots, m\}$.

Step 7. First, R decrypts y'_R with her decryption key to obtain $y''_R = f_{k_1}(\ldots(f_{k_m}(v')))$, and then for each i, $1 \le i \le m$: 1) Finds tuples in Z_i whose entry related to attribute A is equal to y''_R; 2) Considers the entry corresponding to attribute B of those tuples; 3) Decrypts I_i with k_r, obtaining $f_{k'_i}(v')$; 4) Uses $f_{k'_i}(v')$ to decrypt the corresponding entry in Y_i^B.

Although this protocol is simple and effective, it has high cost in terms of communication and computation over large data sets (see Section 5). Moreover, it suffers from high query computation workload at user side. In the next section, we use data bucketization to improve the protocol.

3.2 Protocol 2: B-SMEQ

In this Section, we describe the protocol B-SMEQ, which adopts a bucketization technique on searchable attribute to reduce the communication and computation costs, while preserving the user and owner privacy and the result anonymity. B-SMEQ makes use of a TTP that is not involved in the query processing, but only in the realization of the bucketization scheme. Before the protocol starts, the TTP builds an interchange matrix W(see Phase 1- Step 3) and sends the row vectors of W to the data owners. The owner that receives W_1 is called initiator. Then, data owners are arranged into a ring; each holds a permutation[4] of the bucket order $\{1, \ldots, s\}$ and a vector from matrix W. In the first part of the protocol, each owner sends her buckets in encrypted form to the next participant. When a user wants to submit an equality query, she gets from the TTP the bucket number of the query value according to the TTP's own permutation, $\bar{\Pi}$.

[3] The keys $k_1, k_2, \ldots, k_m$ represent a commutative set of keys.

[4] For the purpose of this paper, we assume that partitions are abundant meaning that the number of permutations is much greater than the number of participants.

This number is sent to the initiator who uses it to identify the bucket related to the user value from her predecessor in the ring she should overencrypt with her key. The procedure is repeated for each node of the ring. In the second phase of the protocol, selected buckets (i.e., the ones corresponding to the user value) are propagated along the ring until they have been encrypted by all keys. Once all data owners hold the encrypted buckets, a two-party Agrawal's protocol is executed between the initiator and the user. In the next sections we will provide a detailed description of the algorithm.

Defining Buckets. We divide the range of values, in the domain of each searchable attribute A, into buckets so that each bucket is assigned a unique label (ID). This label is then stored alongside the encrypted version of the searchable attribute. We adopt equi-width strategy for selecting buckets, which divides the range of possible values of searchable domain into s buckets of the same size l, i.e. $l = \frac{A_{max} - A_{min}}{s}$, where A_{max} and A_{min} are the maximum and minimum values in the domain of A, respectively. Thus, we define bucket boundaries of the searchable attribute as $BU = \{B_1 : [A_{min},\ l],\ \ldots,\ B_s : (l(s-1),\ A_{max}]\}$. BU is called *public bucketization scheme* and it is accessible to all data owners and authorized users as well.

The Protocol. B-SMEQ has two phases: *computation of matrix* W and *query protocol.* First, in order to preserve data privacy, each data owner O_i separately computes a local permutation $(B_{\pi_{i_1}},\ \ldots B_{\pi_{i_s}})$ of the public bucketization scheme for the searchable attribute. Each data owner i chooses a local permutation $\Pi_i = (\pi_{i_1}, \ldots, \pi_{i_s})$ of bucket indices $(1,\ 2,\ \ldots,\ s)$. Moreover, the TTP chooses her own permutation $\bar{\Pi}$ and sends it to the user posing the query.

Let's now assume that the user query value v falls into bucket B_i of public bucketization. Then, the user sends to a pre-set data owner (henceforth, the *initiator*) a query for the bucket $B_{\bar{\pi}_i}$. In this way, when data circulate along the ring, data owners will not know which bucket ID the user is looking for. We will now present our matrix-based mechanism, which allows each data owner to correctly select the local bucket corresponding to $B_{\bar{\pi}_i}$.

Phase 1. Computation of Matrix *W*

Step 1. Each data owner O_i sends her permutation Π_i to the TTP.

Step 2. The TTP builds the matrix Π containing the received permutation vectors and generates a $m \times s$ interchange matrix W, where the matrix elements are defined by Eq. (1).

$$\Pi = \begin{pmatrix} \Pi_1 \\ \Pi_2 \\ \vdots \\ \Pi_m \end{pmatrix} = \begin{pmatrix} \pi_{11} & \pi_{12} & \ldots & \pi_{1s} \\ \pi_{21} & \pi_{22} & \ldots & \pi_{2s} \\ \vdots & & & \vdots \\ \pi_{m1} & & & \pi_{ms} \end{pmatrix} W = \begin{pmatrix} W_1 \\ W_2 \\ \vdots \\ W_m \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1s} \\ w_{21} & w_{22} & \ldots & w_{2s} \\ \vdots & & & \vdots \\ w_{m1} & & & w_{ms} \end{pmatrix}$$

In the following equation, we denote by $\bar{\Pi}^{-1}(l)$ the position in vector $\bar{\Pi}$ that contains value l: $w_{ij} = \pi_{i\delta_i}$ $\forall i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, s\}$ where

$$
\begin{aligned}
&\delta_1 = \bar{\Pi}^{-1}(j) \\
&\delta_2 = \bar{\Pi}^{-1}(k_1) \quad \text{with } k_1 \text{ s.t. } w_{1k_1} = j \\
&\vdots \\
&\delta_i = \bar{\Pi}^{-1}(k_{i-1}) \\
&\qquad \text{with } k_{i-1} \text{ s.t. } w_{1k_{i-1}} = k_{i-2}, \\
&\qquad \text{with } k_{i-2} \text{ s.t. } w_{2k_{i-2}} = k_{i-3}, \\
&\qquad \vdots \\
&\qquad \text{with } k_1 \text{ s.t. } w_{i-1k_1} = j
\end{aligned} \tag{1}
$$

The rationale behind Eq. (1) is generating the entries of matrix W by identifying a corresponding entry of matrix Π. This entry, for the first row ($i = 1$) is obtained by looking for the position of index j in the TTP permutation. For the other rows , the entry is obtained by looking for the position of index j in the previous rows of matrix W itself. For instance to compute δ_2 for $w_{21} = \pi_{2\delta_2}$, TTP follows this procedure: (a) Find $j = 1$ in the previous line of matrix W, (b) Take the position of 1 and read the value in this position of the TTP permutation.
Step 3. The TTP sends the row vectors of W to the data owners and her permutation $\bar{\Pi}$ to the user.

Phase 2. Query Protocol

Steps 1-3 of B-SMEQ correspond to Steps 1-3 (excluding Step 3.5) of SMEQ
Step 4. The user sends $B_{\bar{\pi}_k}$ to the initiator, where k is the number of the bucket where the query value v falls.
Step 5. Let us recall that, at this step, each data owner O_i holds data Y_{i-1} (which corresponds to T_{i-1} encrypted with the key k_{i-1}) of O_{i-1}. With this arrangement, O_2 is the algorithm's initiator[5]. O_2 sets $h_2 = w_{1\bar{\pi}_k}$, selects $Y_1(h_2)$ i.e. the bucket in Y_1 whose ID is h_2, and encrypts it with her own key. Then O_2 sends h_2 to the next owner. When data owner O_i receives h_{i-1} from O_{i-1}, he sets $h_i = w_{(i-1)h_{i-1}}$, selects the corresponding bucket $Y_{i-1}(h_i)$ and sends the position h_i to the next owner. This step iterates until all data owners have selected their bucket.
Step 6. The data owners apply the procedure described in Step 3.5 of SMEQ just on the selected buckets obtained at Step 5.
Steps 7-10 of B-SMEQ correspond to the Steps 4-7 of SMEQ

4 Correctness and Privacy Issues

Correctness. In order to show that B-SMEQ computes query results correctly, we will sketch a simple proof showing that data owners always select the position corresponding to the bucket ID including the user value (Step 5).

[5] This is not a limitation, because any node can be selected as initiator by changing the order of permutations at the time of W generation by TTP.

- Owner O_2 who has the data Y_1 of O_1 receives $\bar{\pi}_a$, computes $h_2 = w_{1\bar{\pi}_a}$, selects $Y_1(h_2)$ and sends h_2 to the O_3. Observe that by definition $h_2 = w_{1\bar{\pi}_a} = \pi_{1\delta_1}$, where $\delta_1 = \bar{\Pi}^{-1}(\bar{\pi}_a)$. Hence $h_2 = \pi_{1a}$, which means O_2 chooses the correct bucket.
- Owner O_i who has data Y_{i-1} of O_{i-1} receives h_{i-1} from party $i-1$, computes $h_i = w_{(i-1)h_{i-1}}$, selects $Y_{i-1}(h_i)$, and sends h_i to next owner. Recall that by Eq. (1), we have $h_i = w_{(i-1)h_{i-1}} = \pi_{(i-1)\delta_{i-1}}$, where $\delta_{i-1} = \bar{\Pi}^{-1}(k_{i-2})$ and

$$\begin{aligned} &k_{i-2} \text{ s.t. } w_{1k_{i-2}} = k_{i-3} \\ &\vdots \\ &k_2 \text{ s.t. } w_{(i-3)k_2} = k_1 \\ &k_1 \text{ s.t. } w_{(i-2)k_1} = h_{i-1} \end{aligned} \tag{2}$$

Since by definition $h_{i-1} = w_{(i-2)h_{i-2}}$, we have that $w_{(i-2)k_1} = w_{(i-2)h_{i-2}}$, hence $k_1 = h_{i-2}$. Again by definition, $h_{i-2} = w_{(i-3)h_{i-3}}$ and then $w_{(i-3)k_2} = w_{(i-3)h_{i-3}}$ that implies $k_2 = h_{i-3}$. By iterating this procedure, we obtain $k_{i-3} = h_2$, that implies $w_{1k_{i-2}} = h_2 = w_{1\bar{\pi}_a}$. This means $k_{i-2} = \bar{\pi}_a$, since $\delta_{i-1} = a$, proving the correctness for data owner O_i. □

Privacy of B-SMEQ. Here, we provide an informal analysis of B-SMEQ privacy. Here, we just consider processing a single query. A more detailed analysis including multiple queries will be carried out as future work. We focus on the privacy of Steps 5 and 6 where the records of each data owner are selected according to the user bucket ID and circulated in order to be encrypted by all keys.

- *Indistinguishability of data distribution*: At each round of the query protocol, every data owner receives a new set from her predecessor via the ring. Since each value of the received set has been hashed and encrypted using commutative encryption function. The distribution of encrypted hash values are indistinguishable from the uniform random distribution.
- *Protection from relation inference*: In each round of Step 6, data owner i receives a data set encrypted with different keys, so that party i can not infer the relationship between received data sets.
- *Protection from bucket inference*: In addition to the size of the whole upstream data set (revealed in the set intersection protocol), in Step 5 each party can learn the size (number of tuples) of the upstream selected bucket. Nevertheless, when a data owner O_i overencrypts the encrypted tuples received from her predecessor, owner O_i does not know which bucket ID corresponds to the received tuples. The size of the selected position does not reveal any further information to the data owner.

According to the above analysis, our protocol is secure against semi-honest parties (see Section 2) as long as no two data owners collude.

5 Time Complexity Analysis

5.1 Theoretical Cost Analysis

Computation Cost. The main computation cost during the execution of both protocols belongs to hashing and encrypting the set of values V_A corresponding

Table 1. Computation and Communication costs of SMEQ and B-SMEQ with m=10 data owners, s=5 buckets and t equal to 10% of the number of records

Number of records	C_{SMEQ}	C_{B-SMEQ}	C'_{SMEQ}	C'_{B-SMEQ}
50000	$660.022 \cdot 10^3$	$300.022 \cdot 10^3$	$5450.645 \cdot 10^3$	$991.145 \cdot 10^3$
100000	$1320.022 \cdot 10^3$	$600.022 \cdot 10^3$	$10900.645 \cdot 10^3$	$1981.145 \cdot 10^3$
200000	$2640.022 \cdot 10^3$	$1200.022 \cdot 10^3$	$26160.774 \cdot 10^3$	$4753.375 \cdot 10^3$
300000	$3960.022 \cdot 10^3$	$1800.022 \cdot 10^3$	$39240.774 \cdot 10^3$	$7129.374 \cdot 10^3$
500000	$6600.022 \cdot 10^3$	$3000.022 \cdot 10^3$	$65400.774 \cdot 10^3$	$11881.374 \cdot 10^3$

to n distributed records among m data owners[6]. Let C_h be the cost of evaluating the hash function h, C_f be the cost of encryption/decryption by function f, C_E be the cost of encryption/decryption by function$E$, and let t be the number of tuples satisfying the equality match[7]. Following the SMEQ protocol step-by-step, it is possible to quantify the complexity of each step and compute the overall complexity of the protocol. For the sake of conciseness, we omit the details, but report the total computation cost of SMEQ :

$$\begin{aligned} C_{SMEQ} &= (n+1) + 1 + (nm+n) + (n+m) + m + 2t \\ &= nm + 3n + 2m + 2 + 2t \in O(mn) \end{aligned} \tag{3}$$

As regards the B-SMEQ protocol, a reduction factor s where s is the number of buckets, is to be considered affecting the evaluation of the computation cost. Table 1 shows the computation costs C_{SMEQ} and C_{B-SMEQ} for the protocols SMEQ and B-SMEQ, respectively. We supposed t to be equal to 10 percent of the number of records. B-SMEQ has significantly lower computation cost for the record cardinality values of interest for practical applications.

Communication Cost. Communication cost can be computed as the total number of bits transmitted during the protocol execution. We compute the total communication cost of our protocols under the assumption that each data owner has $\frac{n}{m}$ expected records (uniform distribution). We consider k as the length of each encrypted codeword of the domain of encryption function f (see Section 2) and k' the length of the encryption function E on other attribute. The communication cost of SMEQ can be computed by looking each step of the procedure, for the sake of conciseness, here we only give the final result that is $O(mn)$:

$$C'_{SMEQ} = (m^2 + nm - \tfrac{n}{m} + 3m + n - 1) \cdot k + (nm - \tfrac{n}{m} + n) \cdot k' \tag{4}$$

The total communication cost of B-SMEQ is:

$$\begin{aligned} C'_{B-SMEQ} &= (m^2 + 3m + \tfrac{1}{s}(mn - \tfrac{n}{m}) + 2ms + \tfrac{n}{m} - 1) \cdot k \\ &+ (\tfrac{n}{m} + \tfrac{1}{s}(nm - n - \tfrac{n}{m})) \cdot k' \end{aligned} \tag{5}$$

Table 1 presents the communication costs C'_{SMEQ} and C'_{B-SMEQ} of SMEQ and B-SMEQ. We assumed k and k' to be equal to the smallest integer greater than

[6] We compute the total computation cost of the two mentioned protocols with the assumption that each data owner has$\frac{n}{m}$ expected records (uniform distribution).

[7] For the sake of simplicity in the calculations below, we do not consider the cost of reordering encrypted values and assume a unitary cost when applying C_h, C_f and C_E to a single record.

$\lg(n)$. As Table 1 shows, the B-SMEQ protocol has lower communication cost; also, in this range, the difference between the two protocol increases with the number of records.

5.2 Practical Cost Analysis

In this section, we verify the scalability of our protocol via some experimental tests. We report the results in terms of communication time on a Linux machine with dual Intel CPU running at 2.26 GHz. and 2GB Ram. We use Castalia[8] simulator for Wireless Sensor Networks. In particular, for our simulation, we deploy 4 and 5 nodes, respectively for SMEQ and B-SMEQ. For both protocols, we create a ring of three nodes with numbers from 0 to 2 for the three data owners and one node (number 3) for the querier[9]. To encrypt the set of searchable attribute of each data owner's table, we implement a simple *commutative encryption* protocol based on *exponentiation modulo p*. Since the most communication time of the two protocols is devoted to exchanging data of each data owner along the ring in order to be encrypted by all keys, we only focus on Steps 3.4 and 3.5 of SMEQ, and Steps 5 and 6 of B-SMEQ. We run two different experiments: in the first one, we compare the two protocols; in the second one, we evaluate the effect the number of buckets has on the communication time for B-SMEQ.

In the first experiment, we set five different triples $(\theta 1, \ldots, \theta 5)$ of number of records as $\theta 1 = \{5, 6, 7\}$, $\theta 2 = \{50, 60, 70\}$, $\theta 3 = \{500, 600, 700\}$, $\theta 4 = \{5000, 6000, 7000\}$ and $\theta 5 = \{50000, 60000, 70000\}$, where the position j in each triple is the number of records for data owner $j \in \{1, 2, 3\}$. Note that for B-SMEQ, we divide the searchable attribute domain $[1 \ldots 100]$ into s=5 number of buckets of the same size $l = 20$. Figure 2(a) shows the result of our simulation, the solid line displays the result from SMEQ, whereas the dotted line displays the result from B-SMEQ. The difference in communication time between SMEQ and B-SMEQ increases fairly slowly when the number of records of data owners is relatively small, but it grows much faster as the number of records increases. The results come from the fact that for each query, in SMEQ all records of data owners must pass through the ring, while in B-SMEQ only records corresponding to the bucket ID of user's query are taken into consideration. In the second simulation, we study the effect of increasing the number of buckets on communication time for B-SMEQ. We fix the number of records given by $\theta 5$ and searchable attribute with range $[1 \ldots 100]$ and repeat the experiment with s=5k, where $k \in \{1, \ldots, 13\}$. In Figure 2(b), the x axis shows the number of buckets and y axis shows the total communication time. Interestingly, with respect to varying the number of buckets in ascending order, we can see a progressive communication time decreasing. For instance, when the number of buckets is $s = 5$, B-SMEQ provides about 2 times improvement over SMEQ (i.e. s=1). The reason is that, increasing the number of buckets, the expected number of records "falling" in each bucket decreases. Moreover, the improvement due to bucketization is higher for $1 \leq s \leq 10$, since when $s > 10$

[8] `http://www.omnetpp.org/component/content/article/8-news/3478`

[9] It should be noted that for B-SMEQ we need to deploy an initiator node for the role of TTP.

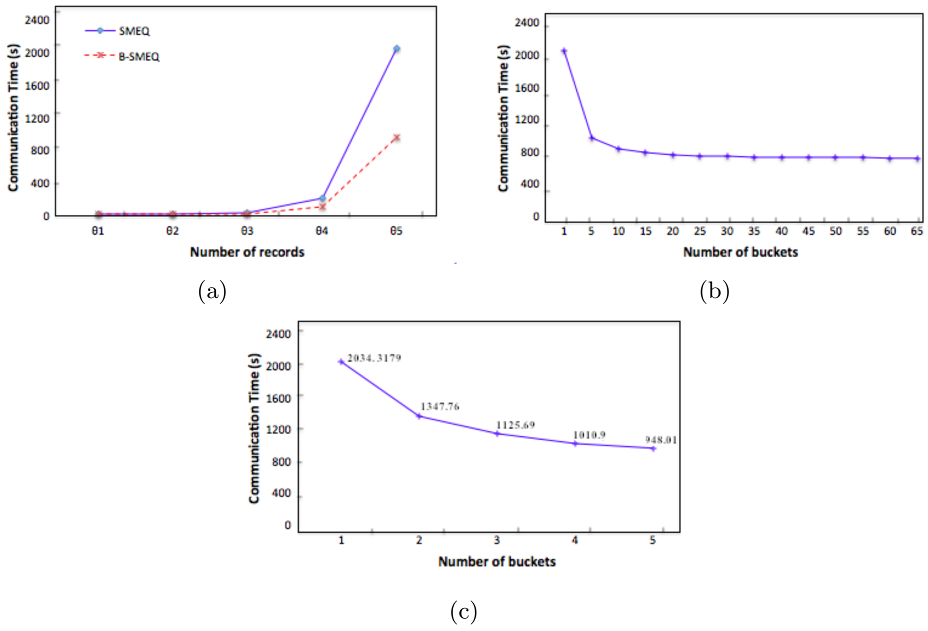

Fig. 2. (a) Comparison of SMEQ and B-SMEQ Protocols based on communication time (m=3, s=5), (b) Effect of increasing number of buckets on communication time (θ_5, m=3, $1 \le s \le 65$) and (c) Effect of increasing number of buckets on communication time (θ_5, m=3, $1 \le s \le 5$)

the expected number of records in each bucket and for each data owner does not considerably change. In order to further clarify this concept, in Figure 3(c) we show the results of the same experiment when $s = \{1, 2, 3, 4, 5\}$. Our results show that bucketization decreases communication time dramatically at first; then, the marginal contribution of additional buckets to speed-up tends to decrease. This behavior suggests finding the optimal number of buckets, i.e., the number of buckets where the marginal contribution speed of the additional bucket is negligible. This behavior happens regardless of data distribution (uniform and normal).

6 Conclusions

In this paper, we proposed the B-SMEQ protocol to compute equality test queries in multi-party setting. Unlike existing approaches, our protocol scales well to large size data and it is designed to work with more than two parties. The protocol adopts a bucketization technique to reduce the workload and make the algorithm fast even when the number of records in the private databases considerably increases. Experimental tests on randomly generated databases with around $2{\cdot}10^5$ records confirm the efficiency on our protocol. Since each range query can be translated into a sequence of equality queries [15], our protocol can be straightforwardly extended to manage *range* queries. Moreover, an extension to join queries is underway, taking into account table fragmentation along multiple data owners.

References

1. Damiani, E., Vimercati, S.D.C., Jajodia, S., Paraboschi, S., Samarati, P.: Balancing confidentiality and efficiency in untrusted relational dbmss. In: Proceedings of the 10th ACM conference on Computer and communications security, CCS 2003, pp. 93–102. ACM, New York (2003)
2. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing sql over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, SIGMOD 2002, pp. 216–227. ACM, New York (2002)
3. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
4. Yao, A.C.C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, SFCS 1986, pp. 162–167. IEEE Computer Society, Washington, DC (1986)
5. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC 1987, pp. 218–229. ACM, New York (1987)
6. Agrawal, R., Asonov, D., Li, M.K.: Sovereign joins. In: Proceedings of the 22nd International Conference on Data Engineering (2006)
7. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD 2003, pp. 86–97. ACM, New York (2003)
8. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
9. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection, pp. 1–19. Springer (2004)
10. Lindell, Y., Pinkas, B.: Privacy preserving data mining. J. Cryptology 15(3), 177–206 (2002)
11. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM J. Comput. 35(5), 1254–1281 (2006)
12. Li, R., Wu, C.: Co-operative private equality test. I. J. Network Security 1(3), 149–153 (2005)
13. Sang, Y., Shen, H., Tan, Y., Xiong, N.: Efficient protocols for privacy preserving matching against distributed datasets. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 210–227. Springer, Heidelberg (2006)
14. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security 13(4), 593–622 (2005)
15. Damiani, E., De, S., Vimercati, C., Paraboschi, S., Samarati, P.: Computing range queries on obfuscated data. In: Proc. of the Information Processing and Management of Uncertainty in Knowledge-Based Systems (2004)