Secure Mining of Association Rules in Horizontally Distributed Databases

Tamir Tassa

Abstract—We propose a protocol for secure mining of association rules in horizontally distributed databases. The current leading protocol is that of Kantarcioglu and Clifton [18]. Our protocol, like theirs, is based on the Fast Distributed Mining (FDM) algorithm of Cheung et al. [8], which is an unsecured distributed version of the Apriori algorithm. The main ingredients in our protocol are two novel secure multi-party algorithms — one that computes the union of private subsets that each of the interacting players hold, and another that tests the inclusion of an element held by one player in a subset held by another. Our protocol offers enhanced privacy with respect to the protocol in [18]. In addition, it is simpler and is significantly more efficient in terms of communication rounds, communication cost and computational cost.

Index Terms—Privacy Preserving Data Mining; Distributed Computation; Frequent Itemsets; Association Rules.

1 Introduction

We study here the problem of secure mining of association rules in horizontally partitioned databases. In that setting, there are several sites (or players) that hold homogeneous databases, i.e., databases that share the same schema but hold information on different entities. The goal is to find all association rules with support at least s and confidence at least c, for some given minimal support size s and confidence level c, that hold in the unified database, while minimizing the information disclosed about the private databases held by those players. The information that we would like to protect in this context is not only individual transactions in the different databases, but also more global information such as what association rules are supported locally in each of those databases.

That goal defines a problem of secure multi-party computation. In such problems, there are M players that hold private inputs, $x_1, \ldots, x_M$, and they wish to securely compute $y = f(x_1, \ldots, x_M)$ for some public function f. If there existed a trusted third party, the players could surrender to him their inputs and he would perform the function evaluation and send to them the resulting output. In the absence of such a trusted third party, it is needed to devise a protocol that the players can run on their own in order to arrive at the required output y. Such a protocol is considered perfectly secure if no player can learn from his view of the protocol more than what he would have learnt in the idealized setting where the computation is carried out by a trusted third party. Yao [32] was the first to propose a generic solution for this problem in the case of two players. Other generic solutions, for the multi-party case, were later proposed in [3], [5], [15].

In our problem, the inputs are the partial databases, and the required output is the list of association rules that hold in the unified database with support and confidence no smaller than the given thresholds s and c, respectively. As the above mentioned generic solutions rely upon a description of the function f as a Boolean circuit, they can be applied only to small inputs and functions which are realizable by simple circuits. In more complex settings, such as ours, other methods are required for carrying out this computation. In such cases, some relaxations of the notion of perfect security might be inevitable when looking for practical protocols, provided that the excess information is deemed benign (see examples of such protocols in e.g. [18], [28], [29], [31], [34]).

Kantarcioglu and Clifton studied that problem in [18] and devised a protocol for its solution. The main part of the protocol is a sub-protocol for the secure computation of the union of private subsets that are held by the different players. (The private subset of a given player, as we explain below, includes the itemsets that are s-frequent in his partial database.) That is the most costly part of the protocol and its implementation relies upon cryptographic primitives such as commutative encryption, oblivious transfer, and hash functions. This is also the only part in the protocol in which the players may extract from their view of the protocol information on other databases, beyond what is implied by the final output and their own input. While such leakage of information renders the protocol not perfectly secure, the perimeter of the excess information is explicitly bounded in [18] and it is argued there that such information leakage is innocuous, whence acceptable from a practical point of view.

Herein we propose an alternative protocol for the secure computation of the union of private subsets. The proposed protocol improves upon that in [18] in terms of simplicity and efficiency as well as privacy. In particular, our protocol does not depend on commutative encryption and oblivious transfer (what simplifies it significantly and contributes towards much reduced communication and computational costs). While our solution is still not perfectly secure, it leaks excess information only to a small number (three) of possible coalitions, unlike the protocol of [18] that discloses information also to some single players. In addition, we claim that the excess information

- *T. Tassa is with the Department of Mathematics and Computer Science, The Open University, Ra'anana, Israel.*

that our protocol may leak is less sensitive than the excess information leaked by the protocol of [18].

The protocol that we propose here computes a parameterized family of functions, which we call threshold functions, in which the two extreme cases correspond to the problems of computing the union and intersection of private subsets. Those are in fact general-purpose protocols that can be used in other contexts as well. Another problem of secure multi-party computation that we solve here as part of our discussion is the set inclusion problem; namely, the problem where Alice holds a private subset of some ground set, and Bob holds an element in the ground set, and they wish to determine whether Bob's element is within Alice's subset, without revealing to either of them information about the other party's input beyond the above described inclusion.

1.1 Preliminaries

1.1.1 *Definitions and notations*

Let D be a transaction database. As in [18], we view D as a binary matrix of N rows and L columns, where each row is a transaction over some set of items, $A = \{a_1, \ldots, a_L\}$, and each column represents one of the items in A. (In other words, the (i, j)th entry of D equals 1 if the ith transaction includes the item a_j, and 0 otherwise.) The database D is partitioned horizontally between M players, denoted $P_1, \ldots, P_M$. Player P_m holds the partial database D_m that contains $N_m = |D_m|$ of the transactions in D, $1 \le m \le M$. The unified database is $D = D_1 \cup \cdots \cup D_M$, and it includes $N := \sum_{m=1}^{M} N_m$ transactions.

An itemset X is a subset of A. Its global support, $supp(X)$, is the number of transactions in D that contain it. Its local support, $supp_m(X)$, is the number of transactions in D_m that contain it. Clearly, $supp(X) = \sum_{m=1}^{M} supp_m(X)$. Let s be a real number between 0 and 1 that stands for a required support threshold. An itemset X is called s-frequent if $supp(X) \ge sN$. It is called locally s-frequent at D_m if $supp_m(X) \ge sN_m$.

For each $1 \le k \le L$, let F_s^k denote the set of all k-itemsets (namely, itemsets of size k) that are s-frequent, and $F_s^{k,m}$ be the set of all k-itemsets that are locally s-frequent at D_m, $1 \le m \le M$. Our main computational goal is to find, for a given threshold support $0 < s \le 1$, the set of all s-frequent itemsets, $F_s := \bigcup_{k=1}^{L} F_s^k$. We may then continue to find all (s, c)-association rules, i.e., all association rules of support at least sN and confidence at least c. (Recall that if X and Y are two disjoint subsets of A, the support of the corresponding association rule $X \Rightarrow Y$ is $supp(X \cup Y)$ and its confidence is $supp(X \cup Y)/supp(X)$.)

1.1.2 *The Fast Distributed Mining algorithm*

The protocol of [18], as well as ours, are based on the Fast Distributed Mining (FDM) algorithm of Cheung et al. [8], which is an unsecured distributed version of the Apriori algorithm. Its main idea is that any s-frequent itemset must be also locally s-frequent in at least one of the sites. Hence, in order to find all globally s-frequent itemsets, each player reveals his locally s-frequent itemsets and then the players check each of them to see if they are s-frequent also globally. The FDM algorithm proceeds as follows:

(1) **Initialization:** It is assumed that the players have already jointly calculated F_s^{k-1}. The goal is to proceed and calculate F_s^k.
(2) **Candidate Sets Generation:** Each player P_m computes the set of all $(k-1)$-itemsets that are locally frequent in his site and also globally frequent; namely, P_m computes the set $F_s^{k-1,m} \cap F_s^{k-1}$. He then applies on that set the Apriori algorithm in order to generate the set $B_s^{k,m}$ of candidate k-itemsets.
(3) **Local Pruning:** For each $X \in B_s^{k,m}$, P_m computes $supp_m(X)$. He then retains only those itemsets that are locally s-frequent. We denote this collection of itemsets by $C_s^{k,m}$.
(4) **Unifying the candidate itemsets:** Each player broadcasts his $C_s^{k,m}$ and then all players compute $C_s^k := \bigcup_{m=1}^{M} C_s^{k,m}$.
(5) **Computing local supports.** All players compute the local supports of all itemsets in C_s^k.
(6) **Broadcast Mining Results:** Each player broadcasts the local supports that he computed. From that, everyone can compute the global support of every itemset in C_s^k. Finally, F_s^k is the subset of C_s^k that consists of all globally s-frequent k-itemsets.

In the first iteration, when $k = 1$, the set $C_s^{1,m}$ that the mth player computes (Steps 2-3) is just $F_s^{1,m}$, namely, the set of single items that are s-frequent in D_m. The complete FDM algorithm starts by finding all single items that are globally s-frequent. It then proceeds to find all 2-itemsets that are globally s-frequent, and so forth, until it finds the longest globally s-frequent itemsets. If the length of such itemsets is K, then in the $(K+1)$th iteration of the FDM it will find no $(K+1)$-itemsets that are globally s-frequent, in which case it terminates.

1.1.3 *A running example*

Let D be a database of $N = 18$ itemsets over a set of $L = 5$ items, $A = \{1, 2, 3, 4, 5\}$. It is partitioned between $M = 3$ players, and the corresponding partial databases are:

$$\begin{aligned} D_1 &= \{12, 12345, 124, 1245, 14, 145, 235, 24, 24\} \\ D_2 &= \{1234, 134, 23, 234, 2345\} \\ D_3 &= \{1234, 124, 134, 23\}\,. \end{aligned}$$

For example, D_1 includes $N_1 = 9$ transactions, the third of which (in lexicographic order) consists of 3 items — 1, 2 and 4.

Setting $s = 1/3$, an itemset is s-frequent in D if it is supported by at least $6 = sN$ of its transactions. In this case,

$$\begin{aligned} F_s^1 &= \{1, 2, 3, 4\} \\ F_s^2 &= \{12, 14, 23, 24, 34\} \\ F_s^3 &= \{124\} \\ F_s^4 &= F_s^5 = \emptyset\,, \end{aligned}$$

and $F_s = F_s^1 \cup F_s^2 \cup F_s^3$. For example, the itemset 34 is indeed globally s-frequent since it is contained in 7 transactions of D. However, it is locally s-frequent only in D_2 and D_3.

In the first round of the FDM algorithm, the three players compute the sets $C_s^{1,m}$ of all 1-itemsets that are locally frequent at their partial databases:

$$C_s^{1,1} = \{1,2,4,5\}\,,\ C_s^{1,2} = \{1,2,3,4\}\,,\ C_s^{1,3} = \{1,2,3,4\}\,.$$

Hence, $C_s^1 = \{1,2,3,4,5\}$. Consequently, all 1-itemsets have to be checked for being globally frequent; that check reveals that the subset of globally s-frequent 1-itemsets is $F_s^1 = \{1,2,3,4\}$.

In the second round, the candidate itemsets are:

$$\begin{array}{l} C_s^{2,1} = \{12,14,24\} \\ C_s^{2,2} = \{13,14,23,24,34\} \\ C_s^{2,3} = \{12,13,14,23,24,34\}\,. \end{array}$$

(Note that $15, 25, 45$ are locally s-frequent at D_1 but they are not included in $C_s^{2,1}$ since 5 was already found to be globally infrequent.) Hence, $C_s^2 = \{12,13,14,23,24,34\}$. Then, after veryfing global frequency, we are left with $F_s^2 = \{12,14,23,24,34\}$.

In the third round, the candidate itemsets are:

$$C_s^{3,1} = \{124\}\,,\ C_s^{3,2} = \{234\}\,,\ C_s^{3,3} = \{124\}\,.$$

So, $C_s^3 = \{124, 234\}$ and, then, $F_s^3 = \{124\}$. There are no more frequent itemsets.

1.2 Overview and organization of the paper

The FDM algorithm violates privacy in two stages: In Step 4, where the players broadcast the itemsets that are locally frequent in their private databases, and in Step 6, where they broadcast the sizes of the local supports of candidate itemsets. Kantarcioglu and Clifton [18] proposed secure implementations of those two steps. Our improvement is with regard to the secure implementation of Step 4, which is the more costly stage of the protocol, and the one in which the protocol of [18] leaks excess information. In Section 2 we describe Kantarcioglu and Clifton's secure implementation of Step 4. We then describe our alternative implementation and proceed to analyze the two implementations in terms of privacy and efficiency and compare them. We show that our protocol offers better privacy and that it is simpler and is significantly more efficient in terms of communication rounds, communication cost and computational cost.

In Sections 3 and 4 we discuss the implementation of the two remaining steps of the distributed protocol: The identification of those candidate itemsets that are globally s-frequent, and then the derivation of all (s,c)-association rules. In Section 5 we describe shortly an alternative protocol, that was already considered in [9], [18], which offers full security at enhanced costs. Section 6 describes our experimental evaluation which illustrates the significant advantages of our protocol in terms of communication and computational costs. Section 7 includes a review of related work. We conclude the paper in Section 8.

Like in [18], we assume that the players are semi-honest; namely, they follow the protocol but try to extract as much information as possible from their own view. (See [17], [26], [34] for a discussion and justification of that assumption.) We too, like [18], assume that $M > 2$. (The case $M = 2$ is discussed in [18, Section 5]; the conclusion is that the problem of secure computation of frequent itemsets and association rules in the two-party case is unlikely to be of any use.)

2 Secure computation of all locally frequent itemsets

Here we discuss the secure implementation of Step 4 in the FDM algorithm, namely, the secure computation of the union $C_s^k = \bigcup_{m=1}^M C_s^{k,m}$. We describe the protocol of [18] (Section 2.1) and then our protocol (Sections 2.2–2.3). We analyze the privacy of the two protocols in Section 2.4, their communication cost in Section 2.5, and their computational cost in Section 2.6.

2.1 The protocol of Kantarcioglu and Clifton for the secure computation of all locally frequent itemsets

2.1.1 Overview

Protocol 1 is the protocol that was suggested by Kantarcioglu and Clifton [18] for computing the unified list of all locally frequent itemsets, $C_s^k = \bigcup_{m=1}^M C_s^{k,m}$, without disclosing the sizes of the subsets $C_s^{k,m}$ nor their contents. The protocol is applied when the players already know F_s^{k-1} — the set of all $(k-1)$-itemsets that are globally s-frequent, and they wish to proceed and compute F_s^k. We refer to it hereinafter as Protocol UNIFI-KC (Unifying lists of locally Frequent Itemsets — Kantarcioglu and Clifton).

The input that each player P_m has at the beginning of Protocol UNIFI-KC is the collection $C_s^{k,m}$, as defined in Steps 2-3 of the FDM algorithm. Let $Ap(F_s^{k-1})$ denote the set of all candidate k-itemsets that the Apriori algorithm generates from F_s^{k-1}. Then, as implied by the definition of $C_s^{k,m}$ (see Section 1.1.2), $C_s^{k,m}$, $1 \le m \le M$, are all subsets of $Ap(F_s^{k-1})$. The output of the protocol is the union $C_s^k = \bigcup_{m=1}^M C_s^{k,m}$. In the first iteration of this computation $k = 1$, and the players compute all s-frequent 1-itemsets (here $F_s^0 = \{\emptyset\}$). In the next iteration they compute all s-frequent 2-itemsets, and so forth, until the first $k \le L$ in which they find no s-frequent k-itemsets.

After computing that union, the players proceed to extract from C_s^k the subset F_s^k that consists of all k-itemsets that are globally s-frequent; this is done using the protocol that we describe later on in Section 3. Finally, by applying the above described procedure from $k = 1$ until the first value of $k \le L$ for which the resulting set F_s^k is empty, the players may recover the full set $F_s := \bigcup_{k=1}^L F_s^k$ of all globally s-frequent itemsets.

Protocol UNIFI-KC works as follows: First, each player adds to his private subset $C_s^{k,m}$ fake itemsets, in order to hide its size. Then, the players jointly compute the encryption of their private subsets by applying on those subsets a commutative encryption[1], where each player adds, in his turn, his own layer of encryption using his private secret key. At the end of that stage, every itemset in each subset is encrypted by all

1. An encryption algorithm is called commutative if $E_{K_1} \circ E_{K_2} = E_{K_2} \circ E_{K_1}$ for any pair of keys K_1 and K_2.

of the players; the usage of a commutative encryption scheme ensures that all itemsets are, eventually, encrypted in the same manner. Then, they compute the union of those subsets in their encrypted form. Finally, they decrypt the union set and remove from it itemsets which are identified as fake. We now proceed to describe the protocol in detail.

(Notation agreement: Since all protocols that we present herein involve cyclic communication rounds, the index $M+1$ always means 1, while the index 0 always means M.)

2.1.2 Detailed description

• **In Phase 0** (Steps 2-4), the players select the needed cryptographic primitives: They jointly select a commutative cipher, and each player selects a corresponding private random key. In addition, they select a hash function h to apply on all itemsets prior to encryption. It is essential that h will not experience collisions on $Ap(F_s^{k-1})$ in order to make it invertible on $Ap(F_s^{k-1})$. Hence, if such collusions occur (an event of a very small probability), a different hash function must be selected. At the end, the players compute a lookup table with the hash values of all candidate itemsets in $Ap(F_s^{k-1})$; that table will be used later on to find the preimage of a given hash value.

• **In Phase 1** (Steps 6-19), all players compute a composite encryption of the hashed sets $C_s^{k,m}$, $1 \le m \le M$. First (Steps 6-12), each player P_m hashes all itemsets in $C_s^{k,m}$ and then encrypts them using the key K_m. (Hashing is needed in order to prevent leakage of algebraic relations between itemsets, see [18, Appendix].) Then, he adds to the resulting set faked itemsets until its size becomes $|Ap(F_s^{k-1})|$, in order to hide the number of locally frequent itemsets that he has. (Since $C_s^{k,m} \subseteq Ap(F_s^{k-1})$, the size of $C_s^{k,m}$ is bounded by $|Ap(F_s^{k-1})|$, for all $1 \le m \le M$.) We denote the resulting set by X_m. Then (Steps 13-19), the players start a loop of $M-1$ cycles, where in each cycle they perform the following operation: Player P_m sends a permutation of X_m to the next player P_{m+1}; Player P_m receives from P_{m-1} a permutation of the set X_{m-1} and then computes a new X_m as $X_m = E_{K_m}(X_{m-1})$. At the end of this loop, P_m holds an encryption of the hashed $C_s^{k,m+1}$ using all M keys. Due to the commutative property of the selected cipher, Player P_m holds the set $\{E_M(\cdots(E_2(E_1(h(x))))\cdots) : x \in C_s^{k,m+1}\}$.

• **In Phase 2** (Steps 21-26), the players merge the lists of encrypted itemsets. At the completion of this stage P_1 holds the union set $C_s^k = \bigcup_{m=1}^M C_s^{k,m}$ hashed and then encrypted by all encryption keys, together with some fake itemsets that were used for the sake of hiding the sizes of the sets $C_s^{k,m}$; those fake itemsets are not needed anymore and will be removed after decryption in the next phase.

The merging is done in two stages, where in the first stage the odd and even lists are merged separately. As explained in [18, Section 3.2.1], not all lists are merged at once since if they were, then the player who did the merging (say P_1) would be able to identify all of his own encrypted itemsets (as he would get them from P_M) and then learn in which of the other sites they are also locally frequent.

Protocol 1 (UNIFI-KC) Unifying lists of locally Frequent Itemsets — Kantarcioglu and Clifton

Input: Each player P_m has an input set $C_s^{k,m} \subseteq Ap(F_s^{k-1})$, $1 \le m \le M$.
Output: $C_s^k = \bigcup_{m=1}^M C_s^{k,m}$.

```
1:  Phase 0: Getting started
2:  The players decide on a commutative cipher and each player $P_m$, $1 \le m \le M$, selects a random secret encryption key $K_m$.
3:  The players select a hash function $h$ and compute $h(x)$ for all $x \in Ap(F_s^{k-1})$.
4:  Build a lookup table $T = \{(x, h(x)) : x \in Ap(F_s^{k-1})\}$.
5:  Phase 1: Encryption of all itemsets
6:  for all Player $P_m$, $1 \le m \le M$, do
7:      Set $X_m = \emptyset$.
8:      for all $x \in C_s^{k,m}$ do
9:          Player $P_m$ computes $E_{K_m}(h(x))$ and adds it to $X_m$.
10:     end for
11:     Player $P_m$ adds to $X_m$ faked itemsets until its size becomes $|Ap(F_s^{k-1})|$.
12: end for
13: for $i = 2$ to $M$ do
14:     for all $1 \le m \le M$ do
15:         $P_m$ sends a permutation of $X_m$ to $P_{m+1}$.
16:         $P_m$ receives from $P_{m-1}$ the permuted $X_{m-1}$.
17:         $P_m$ computes a new $X_m$ as the encryption of the permuted $X_{m-1}$ using the key $K_m$.
18:     end for
19: end for
20: Phase 2: Merging itemsets
21: Each odd player sends his encrypted set to player $P_1$.
22: Each even player sends his encrypted set to player $P_2$.
23: $P_1$ unifies all sets that were sent by the odd players and removes duplicates.
24: $P_2$ unifies all sets that were sent by the even players and removes duplicates.
25: $P_2$ sends his permuted list of itemsets to $P_1$.
26: $P_1$ unifies his list of itemsets and the list received from $P_2$ and then removes duplicates from the unified list. Denote the final list by $EC_s^k$.
27: Phase 3: Decryption
28: for $m$ = 1 to $M-1$ do
29:     $P_m$ decrypts all itemsets in $EC_s^k$ using $K_m$.
30:     $P_m$ sends the permuted (and $K_m$-decrypted) $EC_s^k$ to $P_{m+1}$.
31: end for
32: $P_M$ decrypts all itemsets in $EC_s^k$ using $K_M$; denote the resulting set by $C_s^k$.
33: $P_M$ uses the lookup table $T$ to replace hashed values with the actual itemsets, and to identify and remove faked itemsets.
34: $P_M$ broadcasts $C_s^k$.
```

• **In Phase 3** (Steps 28-34), a similar round of decryptions is initiated. At the end, the last player who performs the last decryption uses the lookup table T that was constructed in

Step 4 in order to identify and remove the fake itemsets and then to recover C_s^k. Finally, he broadcasts C_s^k to all his peers.

Going back to the running example in Section 1.1.3, the set F_s^2, consisting of all 2-itemsets that are globally s-frequent, includes the itemsets $\{12, 14, 23, 24, 34\}$. Applying on it the Apriori algorithm, we find that $Ap(F_s^2) = \{124, 234\}$. Therefore, each of the three players proceed to look for 3-itemsets from $Ap(F_s^2)$ that are locally s-frequent in his partial database. Since $C_s^{3,1} = \{124\}$, P_1 will hash and encrypt the itemset 124 and will add to it one fake itemset, since $|Ap(F_s^2)| = 2$. As $C_s^{3,2} = \{234\}$ and $C_s^{3,3} = \{124\}$, also P_2 and P_3 will each use one fake itemset. At the completion of the protocol, the three players will conclude that $C_s^3 = \{124, 234\}$. Then, by applying the protocol in Section 3, they will find out that only the first of these two candidate itemsets is globally frequent, whence $F_s^3 = \{124\}$.

2.2 A secure multiparty protocol for computing the OR of private binary vectors

Protocol UNIFI-KC securely computes of the union of private subsets of some publicly known ground set ($Ap(F_s^{k-1})$). Such a problem is equivalent to the problem of computing the OR of private vectors. Indeed, if the ground set is $\Omega = \{\omega_1, \ldots, \omega_n\}$, then any subset B of Ω may be described by the characteristic binary vector $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{Z}_2^n$ where $b_i = 1$ if and only if $\omega_i \in B$. Let $\mathbf{b}_m$ be the binary vector that characterizes the private subset held by player P_m, $1 \le m \le M$. Then the union of the private subsets is described by the OR of those private vectors, $\mathbf{b} := \bigvee_{m=1}^{M} \mathbf{b}_m$.

Such a simple function can be evaluated securely by the generic solutions suggested in [3], [5], [15]. We present here a protocol for computing that function which is much simpler to understand and program and much more efficient than those generic solutions. It is also much simpler than Protocol UNIFI-KC and employs less cryptographic primitives. Our protocol (Protocol 2) computes a wider range of functions, which we call threshold functions.

Definition 2.1. *Let $b_1, \ldots, b_M$ be M bits and $1 \le t \le M$ be an integer. Then*

$$T_t(b_1, \ldots, b_M) = \begin{cases} 1 & \text{if } \sum_{m=1}^{M} b_m \ge t \\ 0 & \text{if } \sum_{m=1}^{M} b_m < t \end{cases} \tag{1}$$

is called the t-threshold function. Given binary vectors $\mathbf{b}_m = (b_m(1), \ldots, b_m(n)) \in \mathbb{Z}_2^n$, $1 \le m \le M$, we let $T_t(\mathbf{b}_1, \ldots, \mathbf{b}_M)$ denote the binary vector in which the ith component equals $T_t(b_1(i), \ldots, b_M(i))$, $1 \le i \le n$.

The OR and AND functions are the 1- and M-threshold functions, respectively; i.e.,

$$\bigvee_{m=1}^{M} \mathbf{b}_m = T_1(\mathbf{b}_1, \ldots, \mathbf{b}_M)\,, \qquad \bigwedge_{m=1}^{M} \mathbf{b}_m = T_M(\mathbf{b}_1, \ldots, \mathbf{b}_M)\,.$$

Those special cases may be used, as we show in Section 2.3, to compute in a privacy-preserving manner unions and intersections of private subsets.

Let $P_1, \ldots, P_M$ be M players where P_m has an input binary vector $\mathbf{b}_m \in \mathbb{Z}_2^n$, $1 \le m \le M$. Protocol 2 (to which we refer as THRESHOLD henceforth) computes, in a secure manner, the output vector $\mathbf{b} := T_t(\mathbf{b}_1, \ldots, \mathbf{b}_M)$, for some $1 \le t \le M$. Let $\mathbf{a} = (a(1), \ldots, a(M)) := \sum_{m=1}^{M} \mathbf{b}_m$ be the sum of the input binary vectors. Since $a(m) \in \mathbb{Z}_{M+1} = \{0, 1, \ldots, M\}$, for all $1 \le m \le M$, the sum vector $\mathbf{a}$ may be seen as a vector in $\mathbb{Z}_{M+1}^n$. The main idea behind the protocol is to use the secure summation protocol of [6] in order to compute shares of the sum vector $\mathbf{a}$ and then use those shares to securely verify the threshold conditions in each component.

Since $\mathbf{a} \in \mathbb{Z}_{M+1}^n$, each player starts by creating random shares in $\mathbb{Z}_{M+1}^n$ of his input vector (Step 1); namely, P_m selects M random vectors in $\mathbb{Z}_{M+1}^n$ that add up to $\mathbf{b}_m$, $1 \le m \le M$. In Step 2, all players send to all other players the corresponding shares in their input vector. Then (Step 3), player P_ℓ, $1 \le \ell \le M$, adds the shares that he got and arrives at his share, $\mathbf{s}_\ell$, in the sum vector $\mathbf{a} := \sum_{m=1}^{M} \mathbf{b}_m$. Namely, $\mathbf{a} = \sum_{\ell=1}^{M} \mathbf{s}_\ell \bmod (M+1)$ and, furthermore, any $M-1$ vectors out of $\{\mathbf{s}_1, \ldots, \mathbf{s}_M\}$ do not reveal any information on the sum $\mathbf{a}$. In Steps 4-5, all players, apart from the last one, send their shares to P_1 who adds them up to get the share $\mathbf{s}$. Now, players P_1 and P_M hold additive shares of the sum vector $\mathbf{a}$: P_1 has $\mathbf{s}$, P_M has $\mathbf{s}_M$, and $\mathbf{a} = (\mathbf{s} + \mathbf{s}_M) \bmod (M+1)$. It is now needed to check for each component $1 \le i \le n$ whether

$$(s(i) + s_M(i)) \bmod (M+1) < t\,. \tag{2}$$

Whenever inequality (2) holds, we set $b(i) = 0$; otherwise, we set $b(i) = 1$ (Steps 6-8).

We proceed now to discuss the secure verification of inequality (2). That inequality is equivalent to the following set inclusion:

$$(s(i) + s_M(i)) \bmod (M+1) \in \{j : 0 \le j \le t-1\}\,. \tag{3}$$

The inclusion in (3) is equivalent to

$$s(i) \in \Theta(i) := \{(j - s_M(i)) \bmod (M+1) : 0 \le j \le t-1\}\,. \tag{4}$$

The value of $s(i)$ is known only to P_1 while the set $\Theta(i)$ is known only to P_M. The problem of verifying the set inclusion in Eq. (4) can be seen as a simplified version of the *privacy-preserving keyword search*, which was solved by Freedman et. al. [13]. In the case of the OR function, $t = 1$, which is the case relevant for us, the set $\Theta(i)$ is of size 1, and therefore it is the problem of oblivious string comparison, a problem that was solved in e.g. [12]. However, we claim that, since $M > 2$, there is no need to invoke neither of the secure protocols of [13] or [12]. Indeed, as $M > 2$, the existence of other semi-honest players can be used to verify the inclusion in Eq. (4) much more easily. This is done in Protocol 3 (SETINC) which we proceed to describe next.

Protocol SETINC involves three players: P_1 has a vector $\mathbf{s} = (s(1), \ldots, s(n))$ of elements in some ground set Ω; P_M, on the other hand, has a vector $\boldsymbol{\Theta} = (\Theta(1), \ldots, \Theta(n))$ of subsets of that ground set. The required output is a vector $\mathbf{b} = (b(1), \ldots, b(n))$ that describes the corresponding set inclusions in the following manner: $b(i) = 0$ if $s(i) \in \Theta(i)$

Protocol 2 (THRESHOLD) Secure computation of the t-threshold function

Input: Each player P_m has an input binary vector $\mathbf{b}_m \in \mathbb{Z}_2^n$, $1 \le m \le M$.
Output: $\mathbf{b} := T_t(\mathbf{b}_1, \dots, \mathbf{b}_M)$.

```
1: Each $P_m$ selects $M$ random share vectors $\mathbf{b}_{m,\ell} \in \mathbb{Z}_{M+1}^n$, $1 \le \ell \le M$, such that $\sum_{\ell=1}^M \mathbf{b}_{m,\ell} = \mathbf{b}_m \bmod (M+1)$.
2: Each $P_m$ sends $\mathbf{b}_{m,\ell}$ to $P_\ell$ for all $1 \le \ell \ne m \le M$.
3: Each $P_\ell$ computes $\mathbf{s}_\ell = (s_\ell(1), \dots, s_\ell(n)) := \sum_{m=1}^M \mathbf{b}_{m,\ell} \bmod (M+1)$.
4: Players $P_\ell$, $2 \le \ell \le M-1$, send $\mathbf{s}_\ell$ to $P_1$.
5: $P_1$ computes $\mathbf{s} = (s(1), \dots, s(n)) := \sum_{\ell=1}^{M-1} \mathbf{s}_\ell \bmod (M+1)$.
6: for $i = 1, \dots, n$ do
7:     If $(s(i) + s_M(i)) \bmod (M+1) < t$ set $b(i) = 0$ otherwise set $b(i) = 1$.
8: end for
9: Output $\mathbf{b} = (b(1), \dots, b(n))$.
```

and $b(i) = 1$ if $s(i) \notin \Theta(i)$, $1 \le i \le n$. The computation in the protocol involves a third player P_2. (When Protocol SETINC is called from Protocol THRESHOLD, the ground set is $\Omega = \mathbb{Z}_{M+1}$ and the inputs $s(i)$ and $\Theta(i)$ of the two players are as in Eq. (4), $1 \le i \le n$.)

The protocol starts with players P_1 and P_M agreeing on a keyed hash function $h_K(\cdot)$ (e.g., HMAC [4]), and a corresponding secret key K (Step 1). Consequently (Steps 2-3), P_1 converts his sequence of elements $\mathbf{s} = (s(1), \dots, s(n))$ into a sequence of corresponding "signatures" $\mathbf{s}' = (s'(1), \dots, s'(n))$, where $s'(i) = h_K(i, s(i))$ and P_M does a similar conversions to the subsets that he holds. Then, in Steps 4-5, P_1 sends $\mathbf{s}'$ to P_2, and P_M sends to P_2 the subsets $\Theta'(i)$, $1 \le i \le n$, where the elements within each subset are randomly permuted. Finally (Steps 6-7), P_2 performs the relevant inclusion verifications on the signature values. If he finds out that for a given $1 \le i \le n$, $s'(i) \in \Theta'(i)$, he may infer, with high probability, that $s(i) \in \Theta(i)$ (see more on that below), whence he sets $b(i) = 0$. If, on the other hand, $s'(i) \notin \Theta'(i)$, then, with certainty, $s(i) \notin \Theta(i)$, and thus he sets $b(i) = 1$.

Two comments are in order:

(1) If the index i had not been part of the input to the hash function (Steps 2-3), then two equal components in P_1's input vector, say $s(i) = s(j)$, would have been mapped to two equal signatures, $s'(i) = s'(j)$. Hence, in that case player P_2 would have learnt that in P_1's input vector the ith and jth components are equal. To prevent such leakage of information, we include the index i in the input to the hash function.

(2) An event in which $s'(i) \in \Theta'(i)$ while $s(i) \notin \Theta(i)$ indicates a collusion; specifically, it implies that there exist $\theta' \in \Theta(i)$ and $\theta'' \in \Omega \setminus \Theta(i)$ for which $h_K(i, \theta') = h_K(i, \theta'')$. Hash functions are designed so that the probability of such collusions is negligible, whence the risk of a collusion can be ignored. However, it is possible for player P_M to check upfront the selected random key K in order to verify that for all $1 \le i \le n$, the sets $\Theta'(i) = \{h_K(i, \theta) : \theta \in \Theta(i)\}$ and $\Theta''(i) = \{h_K(i, \theta) : \theta \in \Omega \setminus \Theta(i)\}$ are disjoint.

Protocol 3 (SETINC) Set Inclusion computation

Input: P_1 has a vector $\mathbf{s} = (s(1), \dots, s(n))$ and P_M has a vector $\mathbf{\Theta} = (\Theta(1), \dots, \Theta(n))$, where for all $1 \le i \le n$, $s(i) \in \Omega$ and $\Theta(i) \subseteq \Omega$ for some ground set Ω.
Output: The vector $\mathbf{b} = (b(1), \dots, b(n))$ where $b(i) = 0$ if $s(i) \in \Theta(i)$ and $b(i) = 1$ otherwise, $1 \le i \le n$.

```
1: $P_1$ and $P_M$ agree on a keyed-hash function $h_K(\cdot)$ and on a secret key $K$.
2: $P_1$ computes $\mathbf{s}' = (s'(1), \dots, s'(n))$, where $s'(i) = h_K(i, s(i))$, $1 \le i \le n$.
3: $P_M$ computes $\mathbf{\Theta}' = (\Theta'(1), \dots, \Theta'(n))$, where $\Theta'(i) = \{h_K(i, \theta) : \theta \in \Theta(i)\}$, $1 \le i \le n$.
4: $P_1$ sends to $P_2$ the vector $\mathbf{s}'$.
5: $P_M$ sends to $P_2$ the vector $\mathbf{\Theta}'$ in which each $\Theta(i)$ is randomly permuted.
6: For all $1 \le i \le n$, $P_2$ sets $b(i) = 0$ if $s'(i) \in \Theta'(i)$ and $b(i) = 1$ otherwise.
7: $P_2$ broadcasts the vector $\mathbf{b} = (b(1), \dots, b(n))$.
```

We refer hereinafter to the combination of Protocols THRESHOLD and SETINC as Protocol THRESHOLD-C; namely, it is Protocol THRESHOLD where the verifications of the inequalities in Steps 6-8, which are equivalent to the verification of the set inclusions in Eq. (4), are carried out by Protocol SETINC. Then our claims are as follows:

Theorem 2.2. *Protocol* THRESHOLD-C *is correct, i.e., it computes the threshold function.*

Proof. Protocol THRESHOLD operates correctly if the inequality verifications in Step 7 are carried out correctly, since $(s(i) + s_M(i)) \bmod (M+1)$ equals the ith component $a(i)$ in the sum vector $\mathbf{a} = \sum_{m=1}^M \mathbf{b}_m$. The inequality verification is correct if Protocol SETINC is correct. The latter protocol is indeed correct if the randomly selected key K is such that for all $1 \le i \le n$, the sets $\Theta'(i) = \{h_K(i, \theta) : \theta \in \Theta(i)\}$ and $\Theta''(i) = \{h_K(i, \theta) : \theta \in \Omega \setminus \Theta(i)\}$ are disjoint. As discussed earlier, such a verification can be carried out upfront, and most all selections of K are expected to pass that test. □

Theorem 2.3. *Assume that the $M > 2$ players are semi-honest. Let $C \subset \{P_1, P_2, \dots, P_M\}$ be a coalition of players.*

(a) If $P_2 \notin C$ and at least one of P_1 and P_M is not in C either, then Protocol THRESHOLD-C *is perfectly private with respect to C.*

(b) If $P_2 \in C$ but $P_1, P_M \notin C$, the protocol is computationally private with respect to C.

(c) Otherwise, the coalition C includes at least two of the three players P_1, P_2, P_M. In such cases, it may learn the sum $\mathbf{a} = \sum_{m=1}^M \mathbf{b}_m$, but no information on the private vectors $\mathbf{b}_m$, $1 \le m \le M$, beyond what is implied by that sum and the coalition's input vectors.

(A multiparty computation is perfectly private with respect to a subset of players if it does not enable those players to learn information on the inputs of other players beyond what is implied by the final output and their own inputs, even if they are computationally unbounded. Such a computation is computationally private if it achieves the same goal when the players are polynomially-bounded.)

Proof. The view of each single player P_ℓ consists of $\mathbf{b}_{m,\ell}$ for all $1 \leq m \neq \ell \leq M$, where $\mathbf{b}_{m,\ell}$ is P_ℓ's additive share in an M-out-of-M secret sharing scheme for P_m's private input $\mathbf{b}_m$. In addition, P_1's view includes $\mathbf{s}_2, \ldots, \mathbf{s}_{M-1}$ (Step 4 in Protocol THRESHOLD), which are additive shares in an M-out-of-M secret sharing for the sum $\mathbf{a}$; and P_2's view includes the signatures $\mathbf{s}'$ and $\mathbf{\Theta}'$ (Steps 4 and 5 in Protocol SETINC).

(a) If the coalition C does not include P_2 and, in addition, it does not include at least one of P_1 and P_M, then C's collaborative view consists of *incomplete* sets of additive shares in $\mathbf{b}_1, \ldots, \mathbf{b}_M$ and $\mathbf{a}$. As the M-out-of-M secret sharing scheme is perfect, those additive shares are independent and each one is a uniformly distributed random vector in $\mathbb{Z}_{M+1}^n$. Hence, the coalition C may simulate its view during the protocol, whence the protocol is perfectly private with respect to C.

(b) If $P_2 \in C$ and $P_1, P_M \notin C$, we may repeat the same arguments as before, since P_2's additional view of $\mathbf{s}'$ and $\mathbf{\Theta}'$ can also be simulated by independent and uniformly distributed random vectors; indeed, assuming that the hash function h is secure and that the key K is chosen uniformly at random, then $\mathbf{s}'$ and $\mathbf{\Theta}'$ are indistinguishable from vectors chosen uniformly at random from H^n and $(H^t)^n$, respectively, where H is the hash range.

However, while the coalitions discussed above in (a) cannot learn any information about the inputs of other players, even if their members are computationally unbounded, here the privacy guarantee assumes that P_2 is polynomially bounded. If P_2 is computationally unbounded, he may scan the exponential number of possible keys K in order to find what key P_1 and P_M used. To do so, he will compute for each possible K the hashed values $h_K(i,\theta)$ for all $1 \leq i \leq n$ and $\theta \in \Omega = \{0,1,\ldots,M\}$. Then, he will check whether the signature values that he got from P_1 and P_M (namely, $\mathbf{s}'$ and $\mathbf{\Theta}'$) are consistent with the values which he computed. After finding the true K (assuming that it is the only one that will pass the check), P_2 will be able to recover $s(i)$ from $s'(i)$, and $\Theta(i)$ from $\Theta'(i)$, for all $1 \leq i \leq n$. Since $\Theta(i)$ reveals $s_M(i)$ (see Eq. (4)), P_2 may proceed to compute $a(i) = s(i) + s_M(i)$, $1 \leq i \leq n$. Hence, if P_2 is computationally unbounded, he may be able to deduce the value of the sum $\mathbf{a}$. (If during his check, P_2 finds more than one possible K, he will compute the vector $\mathbf{a}$ that corresponds to each of them, and then infer that the true $\mathbf{a}$ is one of those vectors.)

(c) If $P_1, P_M \in C$, then by adding $\mathbf{s}$ (known to P_1) and $\mathbf{s}_M$ (known to P_M), they will get the sum $\mathbf{a}$. No further information on the input vectors $\mathbf{b}_1, \ldots, \mathbf{b}_M$ may be deduced from the inputs of the players in such a coalition; specifically, every set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_M$ that is consistent with the sum $\mathbf{a}$ is equally likely. (Put differently, such a coalition may simulate its view by selecting at random any set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_M$ whose sum equals $\mathbf{a}$ and then generate for each such vector random additive shares.)

Coalitions C that include either P_1 and P_2 or P_2 and P_M can also recover the vector $\mathbf{a}$. Indeed, P_2 knows $\mathbf{s}'$ and $\mathbf{\Theta}'$, while P_1 or P_M knows h_K, and K. Hence, if P_2 colludes with either P_1 or P_M, he may recover from $\mathbf{s}'$ and $\mathbf{\Theta}'$ the preimages $\mathbf{s}$ and $\mathbf{\Theta}$. Thus, such a coalition can recover $\mathbf{s}$ and $\mathbf{s}_M$, and consequently, it can recover $\mathbf{a}$. As argued before, the shares available for such coalitions do not reveal any further information about the input vectors $\mathbf{b}_1, \ldots, \mathbf{b}_M$. □

The susceptibility of Protocol THRESHOLD-C to coalitions is not very significant because of two reasons:

- The entries of the sum vector $\mathbf{a}$ do not reveal information about specific input vectors. Namely, knowing that $a(i) = p$ only indicates that p out of the M bits $b_m(i)$, $1 \leq m \leq M$, equal 1, but it reveals no information regarding which of the M bits are those.
- There are only three players that can collude in order to learn information beyond the intention of the protocol. Such a situation is far less severe than a situation in which any player may participate in a coalition, since if it is revealed that a collusion took place, there is a small set of suspects.

2.3 An improved protocol for the secure computation of all locally frequent itemsets

As before, we denote by F_s^{k-1} the set of all globally frequent $(k-1)$-itemsets, and by $Ap(F_s^{k-1})$ the set of k-itemsets that the Apriori algorithm generates when applied on F_s^{k-1}. All players can compute the set $Ap(F_s^{k-1})$ and decide on an ordering of it. (Since all itemsets are subsets of $A = \{a_1, \ldots, a_L\}$, they may be viewed as binary vectors in $\{0,1\}^L$ and, as such, they may be ordered lexicographically.) Then, since the sets of locally frequent k-itemsets, $C_s^{k,m}$, $1 \leq m \leq M$, are subsets of $Ap(F_s^{k-1})$, they may be encoded as binary vectors of length $n_k := |Ap(F_s^{k-1})|$. The binary vector that encodes the union $C_s^k := \bigcup_{m=1}^M C_s^{k,m}$ is the OR of the vectors that encode the sets $C_s^{k,m}$, $1 \leq m \leq M$. Hence, the players can compute the union by invoking Protocol THRESHOLD-C on their binary input vectors. This approach is summarized in Protocol 4 (UNIFI).

Protocol 4 (UNIFI) Unifying lists of locally Frequent Itemsets

```
Input: Each player $P_m$ has an input subset $C_s^{k,m} \subseteq Ap(F_s^{k-1})$, $1 \leq m \leq M$.
Output: $C_s^k = \bigcup_{m=1}^M C_s^{k,m}$.
1: Each player $P_m$ encodes his subset $C_s^{k,m}$ as a binary vector $\mathbf{b}_m$ of length $n_k = |Ap(F_s^{k-1})|$, in accord with the agreed ordering of $Ap(F_s^{k-1})$.
2: The players invoke Protocol THRESHOLD-C to compute $\mathbf{b} = T_1(\mathbf{b}_1, \ldots, \mathbf{b}_M) = \bigvee_{m=1}^M \mathbf{b}_m$.
3: $C_s^k$ is the subset of $Ap(F_s^{k-1})$ that is described by $\mathbf{b}$.
```

In the running example in Section 1.1.3, $F_s^2 = \{12, 14, 23, 24, 34\}$ and $Ap(F_s^2) = \{124, 234\}$. The private sets of locally frequent itemsets are $C_s^{3,1} = \{124\}$, $C_s^{3,2} = \{234\}$, and $C_s^{3,3} = \{124\}$. Those private sets will be encoded as $\mathbf{b}_1 = (1,0)$, $\mathbf{b}_2 = (0,1)$, and $\mathbf{b}_3 = (1,0)$. The OR of these vectors is $\mathbf{b} = (1,1)$ and, therefore, $C_s^3 = \{124, 234\}$.

Comment. Replacing T_1 with T_M in Step 2 of Protocol 4 will result in computing the intersection of the private subsets rather than their union.

2.4 Privacy

We begin by analyzing the privacy offered by Protocol UNIFI-KC. That protocol does not respect perfect privacy since it reveals to the players information that is not implied by their own input and the final output. In Step 11 of Phase 1 of the protocol, each player augments the set X_m by fake itemsets. To avoid unnecessary hash and encryption computations, those fake itemsets are random strings in the ciphertext domain of the chosen commutative cipher. The probability of two players selecting random strings that will become equal at the end of Phase 1 is negligible; so is the probability of Player P_m to select a random string that equals $E_{K_m}(h(x))$ for a true itemset $x \in Ap(F_s^{k-1})$. Hence, every encrypted itemset that appears in two different lists indicates with high probability a true itemset that is locally s-frequent in both of the corresponding sites. Therefore, Protocol UNIFI-KC reveals the following excess information:

(1) P_1 may deduce for any subset of the odd players, the number of itemsets that are locally supported by all of them.
(2) P_2 may deduce for any subset of the even players, the number of itemsets that are locally supported by all of them.
(3) P_1 may deduce the number of itemsets that are supported by at least one odd player and at least one even player.
(4) If P_1 and P_2 collude, they reveal for any subset of the players the number of itemsets that are locally supported by all of them.

As for the privacy offered by Protocol UNIFI, we consider two cases: If there are no collusions, then, by Theorem 2.3, Protocol UNIFI offers perfect privacy with respect to all players P_m, $m \neq 2$, and computational privacy with respect to P_2. This is a privacy guarantee better than that offered by Protocol UNIFI-KC, since the latter protocol does reveal information to P_1 and P_2 even if they do not collude with any other player.

If there are collusions, both Protocols UNIFI-KC and UNIFI allow the colluding parties to learn forbidden information. In both cases, the number of "suspects" is small — in Protocol UNIFI-KC only P_1 and P_2 may benefit from a collusion while in Protocol UNIFI only P_1, P_2 and P_M can extract additional information if two of them collude (see Theorem 2.3). In Protocol UNIFI-KC, the excess information which may be extracted by P_1 and P_2 is about the number of common frequent itemsets among any subset of the players. Namely, they may learn that, say, P_2 and P_3 have many itemsets that are frequent in both of their databases (but not which itemsets), while P_2 and P_4 have very few itemsets that are frequent in their corresponding databases. The excess information in Protocol UNIFI is different: If any two out of P_1, P_2 and P_M collude, they can learn the sum of all private vectors. That sum reveals for each specific itemset in $Ap(F_s^{k-1})$ the number of sites in which it is frequent, but not which sites. Hence, while the colluding players in Protocol UNIFI-KC can distinguish between the different players and learn about the similarity or dissimilarity between them, Protocol UNIFI leaves the partial databases totally indistinguishable, as the excess information that it leaks is with regard to the itemsets only.

To summarize, given that Protocol UNIFI reveals no excess information when there are no collusions, and, in addition, when there are collusions, the excess information still leaves the partial databases indistinguishable, it offers enhanced privacy preservation in comparison to Protocol UNIFI-KC.

2.5 Communication cost

Here and in the next section we analyze the communication and computational costs of Protocols UNIFI-KC and UNIFI. In doing so, we use the following notations and terms:

- $K = K_s$ is the size of the longest s-frequent itemset in D.
- The kth iteration refers to the iteration in which F_s^k is computed from F_s^{k-1}. Both protocols have $K+1$ iterations (where in the last iteration, the players find that $F_s^{K+1} = \emptyset$ and consequently terminate the protocol).
- n_k is the number of candidate itemsets of size k in the kth iteration. $n_1 = L$ and $n_k := |Ap(F_s^{k-1})|$ for all $1 < k \leq K+1$.
- $n := \sum_{k=1}^{K+1} n_k$.
- ℓ_k is the number of k-itemsets that were s-frequent in at least one of the sites.
- $\ell := \sum_{k=1}^{K+1} \ell_k$.
- B is the size in bits of representing one frequent itemset. (We use the same notation for frequent itemsets of any length for simplicity. In practice, the length of frequent itemsets is typically a small number.)

In evaluating the communication cost, we consider three parameters: Total number of communication rounds, total number of messages sent, and the overall size of the messages sent. For example, in Step 15 of Protocol UNIFI-KC, every player P_m sends a message to P_{m+1}. Those M messages are sent simultaneously. Hence, each time this step is executed, the counter of communication rounds is increased by 1, the number of messages sent is increased by M, and the total message size is increased by the sum of sizes of those messages.

2.5.1 *Communication cost of Protocol* UNIFI-KC

Let t denote the number of bits required to represent an itemset. Clearly, t must be at least $\log_2 n_k$ for all $1 \leq k \leq L$. However, as Protocol UNIFI-KC hashes the itemsets and then encrypts them, t should be at least the recommended ciphertext length in commutative ciphers. RSA [25], Pohlig-Hellman [24] and ElGamal [10] ciphers are examples of commutative encryption schemes. As the recommended length of the modulus in all of them is at least 1024 bits, we take $t = 1024$.

We begin by analyzing the communication costs of Protocol UNIFI-KC in each of the $K+1$ iterations separately. Each iteration, consists of four phases.

During Phase 1 of Protocol UNIFI-KC, there are $M-1$ rounds of communication. In each such round, each of the M players sends to the next player a message; the length of that message in the kth iteration is tn_k. Hence, the communication cost of this phase in the kth iteration is $(M-1)M$ messages of total size of $(M-1)Mtn_k$ bits.

During Phase 2 of the protocol all odd players send their encrypted itemsets to P_1 and all even players send their encrypted itemsets to P_2. Then P_2 unifies the itemsets he got and sends them to P_1. Hence, this phase takes two more rounds. The communication cost, in the kth iteration, of the first of those two rounds is $M-2$ messages of total size of $(M-2)tn_k$. The communication cost of the second round is a single message whose size is $\mu_1 tn_k$ where $\mu_1 \in [1, M/2]$. (The size of the unified list will equal the lower bound in that range, i.e. tn_k bits, if all lists of all even players coincide; it will equal the upper bound of $Mtn_k/2$ if all those lists are pairwise disjoint.)

In Phase 3, a similar round of decryptions is initiated. The unified list of all encrypted true and fake itemsets may contain in the kth iteration at least n_k itemsets but no more than Mn_k itemsets. Hence, that phase involves $M-1$ rounds with communication cost of $M-1$ messages with a total size of $\mu_2(M-1)tn_k$, where $\mu_2 \in [1, M]$.

Finally, in Step 34, P_M broadcasts C_s^k to all other players. That step adds one more communication round with $M-1$ messages of total size of $(M-1)\ell_k B$, where $\ell_k = |C_s^k|$.

Adding up the above costs over all iterations, $1 \le k \le K+1$, we find that Protocol UNIFI-KC entails:

- $(2M+1)(K+1)$ communication rounds.
- $(M^2+2M-3)(K+1)$ messages.
- $g(M)tn + (M-1)B\ell$ bits of communication, where

$$M^2 + M - 2 \le g(M) \le 2M^2 - \frac{M}{2} - 2\,. \qquad (5)$$

2.5.2 *Communication cost of Protocol* UNIFI

Protocol UNIFI consists of four communication rounds (in each of the iterations): One for Step 2 of Protocol THRESHOLD that it invokes; one for Step 4 of that protocol; one for Steps 4-5 in Protocol SETINC which is used for the inequality verifications in Protocol THRESHOLD; and one for Step 7 in Protocol SETINC.

In the kth iteration, the length of the vectors in Protocol THRESHOLD is n_k; each entry in those vectors represents a number between 0 and $M-1$, whence it may be encoded by $\log_2 M$ bits. Therefore:

- The communication cost of Step 2 in Protocol THRESHOLD is $(M-1)M$ messages of total size of $(M-1)M(\log_2 M)n_k$ bits. (Since each of the M players sends a vector of size $(\log_2 M)n_k$ bits to each of the other $M-1$ players.)
- The communication cost of Step 4 in Protocol THRESHOLD is $M-2$ messages of total size of $(M-2)(\log_2 M)n_k$ bits.
- The communication cost of Step 4 in Protocol SETINC is a single message of size $|h|n_k$, where $|h|$ is the size in bits of the hash function's output. The communication cost of Step 5 in that protocol is also a single message of size $|h|n_k$; indeed, when Protocol SETINC is called from Protocol THRESHOLD-C, the size of the sets $\Theta(i)$ and $\Theta'(i)$ is $t=1$ (see Eq. (4)), because the OR function corresponds to $t=1$.
- The communication cost of Step 7 in Protocol SETINC is $M-1$ messages of total size of $(M-1)\ell_k B$, since P_2 can send to all his peers the actual ℓ_k k-itemsets that were s-frequent in at least one of the sites.

Adding up the above costs over all iterations, $1 \le k \le K+1$, we find that Protocol UNIFI entails:

- $4(K+1)$ communication rounds.
- $(M^2+M-1)(K+1)$ messages.
- $(M^2-2)(\log_2 M)n + 2n|h| + (M-1)B\ell$ bits of communication.

2.5.3 *Comparison*

Comparing the costs of the two protocols as derived in Sections 2.5.1 and 2.5.2 we find that Protocol UNIFI reduces the number of rounds by a factor of $(2M+1)/4$ with respect to Protocol UNIFI-KC. The number of messages in the two protocols is roughly the same. As for the bit communication cost, Protocol UNIFI offers a significant improvement. The improvement factor in the bit communication cost, as offered by Protocol UNIFI with respect to Protocol UNIFI-KC, is

$$\frac{g(M)tn + (M-1)B\ell}{(M^2-2)(\log_2 M)n + 2n|h| + (M-1)B\ell} \qquad (6)$$

where the range of possible values of $g(M)$ is given in Eq. (5). The communication cost of the fourth phase, which is $(M-1)B\ell$ in both protocols, may be neglected, as validated in our experimental evaluation. The reason for this is that it depends on ℓ (the overall number of itemsets that were s-frequent in at least one site), which is much smaller than n (the overall number of Apriori-generated candidate itemsets), and, in addition, it depends on $M-1$ rather than $\Theta(M^2)$ as the other costs. Therefore, the ratio in Eq. (6) may be approximated by

$$\frac{g(M)t}{(M^2-2)(\log_2 M) + 2|h|}$$

As discussed earlier, a plausible setting of t would be $t = 1024$. A typical value of $|h|$ is 160. Hence, For $M=4$ we get an improvement factor that ranges between 53 and 82, while for $M=8$ we get an improvement factor that ranges between 142 and 247.

2.6 Computational cost

In Protocol UNIFI-KC each of the players needs to perform hash evaluations as well as encryptions and decryptions. As the cost of hash evaluations is significantly smaller than the cost of commutative encryption, we focus on the cost of the latter operations. In Steps 8-10 of the protocol, player P_m performs $|C_s^{k,m}| \le n_k = |Ap(F_s^{k-1})|$ encryptions (in the kth iteration). Then, in Steps 13-19, each player performs $M-1$ encryptions of sets that include n_k items. Hence, in Phase 1 in the kth iteration, each player performs between $(M-1)n_k$

and Mn_k encryptions. In Phase 3, each player decrypts the set of items EC_s^k. EC_s^k is the union of the encrypted sets from all M players, where each of those sets has n_k items — true and fake ones. Clearly, the size of EC_s^k is at least n_k. On the other hand, since most of the items in the M sets are expected to be fake ones, and the probability of collusions between fake items is negligible, it is expected that the size of EC_s^k would be close to Mn_k. So, in all its iterations ($1 \le k \le K+1$), Protocol UNIFI-KC requires each player to perform an overall number of close to $2Mn$ (but no less than Mn) encryptions or decryptions, where, as before $n = \sum_{k=1}^{K+1} n_k$. Since commutative encryption is typically based on modular exponentiation, the overall computational cost of the protocol is $\Theta(Mt^3 n)$ bit operations per player.

In Protocol THRESHOLD, which Protocol UNIFI calls, each player needs to generate $(M-1)n$ (pseudo)random $(\log_2 M)$-bit numbers (Step 1). Then, each player performs $(M-1)n$ additions of such numbers in Step 1 as well as in Step 3. Player P_1 has to perform also $(M-2)n$ additions in Step 5. Therefore, the computational cost for each player is $\Theta(Mn \log_2 M)$ bit operations. In addition, Players 1 and M need to perform n hash evaluations. Compared to a computational cost of $\Theta(Mt^3 n)$ bit operations per player, we see that Protocol UNIFI offers a significant improvement with respect to Protocol UNIFI-KC also in terms of computational cost.

3 IDENTIFYING THE GLOBALLY s-FREQUENT ITEMSETS

Protocols UNIFI-KC and UNIFI yield the set C_s^k that consists of all itemsets that are locally s-frequent in at least one site. Those are the k-itemsets that have potential to be also globally s-frequent. In order to reveal which of those itemsets is globally s-frequent there is a need to securely compute the support of each of those itemsets. That computation must not reveal the local support in any of the sites. Let x be one of the candidate itemsets in C_s^k. Then x is globally s-frequent if and only if

$$\Delta(x) := supp(x) - sN = \sum_{m=1}^{M} (supp_m(x) - sN_m) \ge 0\,. \quad (7)$$

We describe here the solution that was proposed by Kantarcioglu and Clifton. They considered two possible settings. If the required output includes all globally s-frequent itemsets, as well as the sizes of their supports, then the values of $\Delta(x)$ can be revealed for all $x \in C_s^k$. In such a case, those values may be computed using a secure summation protocol (e.g. [6]), where the private addend of P_m is $supp_m(x) - sN_m$. The more interesting setting, however, is the one where the support sizes are not part of the required output. We proceed to discuss it.

As $|\Delta(x)| \le N$, an itemset $x \in C_s^k$ is s-frequent if and only if $\Delta(x) \bmod q \le N$, for $q = 2N+1$. The idea is to verify that inequality by starting an implementation of the secure summation protocol of [6] on the private inputs $\Delta_m(x) := supp_m(x) - sN_m$, modulo q. In that protocol, all players jointly compute random additive shares of the required sum $\Delta(x)$ and then, by sending all shares to, say, P_1, he may add them and reveal the sum. If, however, P_M withholds his share of the sum, then P_1 will have one random share, $s_1(x)$, of $\Delta(x)$, and P_M will have a corresponding share, $s_M(x)$; namely, $s_1(x)+s_M(x) = \Delta(x) \bmod q$. It is then proposed that the two players execute the generic secure circuit evaluation of [32] in order to verify whether

$$(s_1(x) + s_M(x)) \bmod q \le N\,. \quad (8)$$

Those circuit evaluations may be parallelized for all $x \in C_s^k$.

We observe that inequality (8) holds if and only if

$$s_1(x) \in \Theta(x) := \{(j - s_M(x)) \bmod q : 0 \le j \le N\}\,. \quad (9)$$

As $s_1(x)$ is known only to P_1 while $\Theta(x)$ is known only to P_M, the verification of the set inclusion in (9) can also be carried out by means of Protocol SETINC. However, the ground set Ω in this case is $\mathbb{Z}_{q=2N+1}$, which is typically a large set. (Recall that when Protocol SETINC is invoked from UNIFI, the ground set Ω is $\mathbb{Z}_{M+1}$, which is usually a small set.) Hence, Protocol SETINC is not useful in this case, and, consequently, Yao's generic protocol remains, for the moment, the protocol of choice to securely verify inequality (8). Yao's protocol is designed for the two-party case. In our setting, as $M > 2$, there exist additional semi-honest players. An interesting question which arises in this context is whether the existence of such additional semi-honest players may be used to verify inequalities like (8), even when the modulus is large, without resorting to costly protocols such as oblivious transfer.

4 IDENTIFYING ALL (s,c)-ASSOCIATION RULES

Once the set F_s of all s-frequent itemsets is found, we may proceed to look for all (s,c)-association rules (rules with support at least sN and confidence at least c), as described in [18]. For $X, Y \in F_s$, where $X \cap Y = \emptyset$, the corresponding association rule $X \Rightarrow Y$ has confidence at least c if and only if $supp(X \cup Y)/supp(X) \ge c$, or, equivalently,

$$C_{X,Y} := \sum_{m=1}^{M} (supp_m(X \cup Y) - c \cdot supp_m(X)) \ge 0\,. \quad (10)$$

Since $|C_{X,Y}| \le N$, then by taking $q = 2N+1$, the players can verify inequality (10), in parallel, for all candidate association rules, as described in Section 3.

In order to derive from F_s all (s,c)-association rules in an efficient manner we rely upon the following straightforward lemma.

Lemma 4.1. *If $X \Rightarrow Y$ is an (s,c)-rule and $Y' \subset Y$, then $X \Rightarrow Y'$ is also an (s,c)-rule.*

Proof: The rule $X \Rightarrow Y'$ has the required support count since $supp(X \cup Y') \ge supp(X \cup Y) \ge sN$. It is also c-confident since $\frac{supp(X \cup Y')}{supp(X)} \ge \frac{supp(X \cup Y)}{supp(X)} \ge c$. Hence, it is an (s,c)-rule too. □

We first find all (s,c)-rules with 1-consequents; namely, all (s,c)-rules $X \Rightarrow Y$ with a consequent (right hand side) Y

of size 1. To that end, we scan all itemsets $Z \in F_s$ of size $|Z| \geq 2$, and for each such itemset we scan all $|Z|$ partitions $Z = X \cup Y$ where $|Y| = 1$ and $X = Z \setminus Y$. The association rule $X \Rightarrow Y$ that corresponds to such a given partition $Z = X \cup Y$ is tested to see whether it satisfies inequality (10). We may test all those candidate rules in parallel and at the end we get the full list of all (s, c)-rules with 1-consequents.

We then proceed by induction; assume that we found all (s, c)-rules with j-consequents for all $1 \leq j \leq \ell - 1$. To find all (s, c)-rules with ℓ-consequents, we rely upon Lemma 4.1. Namely, if $Z \in F_s$ and $Z = X \cup Y$ where $X \cap Y = \emptyset$ and $|Y| = \ell$, then $X \Rightarrow Y$ is an (s, c)-rule only if $X \Rightarrow Y'$ were found to be (s, c)-rules for all $Y' \subset Y$. Hence, we may create all candidate rules with ℓ-consequents and test them against inequality (10) in parallel.

It should be noted that in practice, one usually aims at finding association rules of the form $X \Rightarrow Y$ where $|Y| = 1$, or at least $|Y| \leq \ell$ for some small constant ℓ. However, the above procedure may be continued until all candidate association rules, with no upper bounds on the consequent size, are found.

5 A FULLY SECURE PROTOCOL

As noted in [18, Section 6], the players may dispense the local pruning and union computation in the FDM algorithm (Steps 2-4) and, instead, test all candidate itemsets in $Ap(F_s^{k-1})$ to see which of them are globally s-frequent. Such a protocol is fully secure, as it reveals only the set of globally s-frequent itemsets but no further information about the partial databases. However, as discussed in [18], such a protocol would be much more costly since it requires each player to compute the local support of $|Ap(F_s^{k-1})|$ itemsets (in the kth round) instead of only $|C_s^k|$ itemsets (where $C_s^k = \bigcup_{m=1}^{M} C_s^{k,m}$). In addition, the players will have to execute the secure comparison protocol of [32] to verify inequality (8) for $|Ap(F_s^{k-1})|$ rather than only $|C_s^k|$ itemsets. Both types of added operations are very costly: the time to compute the support size depends linearly on the size of the database, while the secure comparison protocol entails a costly oblivious transfer sub-protocol. Since, as shown in [9], $|Ap(F_s^{k-1})|$ is much larger than $|C_s^k|$, the added computing time in such a protocol is expected to dominate the cost of the secure computation of the union of all locally s-frequent itemsets. Hence, the enhanced security offered by such a protocol is accompanied by increased implementation costs.

6 EXPERIMENTAL EVALUATION

In Section 6.1 we describe the synthetic database that we used for our experimentation. In Section 6.2 we explain how the database was split horizontally into partial databases. In Section 6.3 we describe the experiments that we conducted. The results are given in Section 6.4.

6.1 Synthetic database generation

The databases that we used in our experimental evaluation are synthetic databases that were generated using the same techniques that were introduced in [1] and then used also in subsequent studies such as [8], [18], [23]. Table 1 gives the parameter values that were used in generating the synthetic database. The reader is referred to [8], [18], [23] for a description of the synthetic generation method and the meaning of each of those parameters. The parameter values that we used here are similar to those used in [8], [18], [23].

Parameter	Interpretation	Value
N	Number of transactions in the whole database	500,000
L	Number of items	1000
A_t	Transaction average size	10
A_f	Average size of maximal potentially large itemsets	4
N_f	Number of maximal potentially large itemsets	2000
CS	Clustering size	5
PS	Pool size	60
Cor	Correlation level	0.5
MF	Multiplying factor	1800

TABLE 1
Parameters for generating the synthetic database

6.2 Distributing the database

Given a generated synthetic database D of N transactions and a number of players M, we create an artificial split of D into M partial databases, D_m, $1 \leq m \leq M$, in the following manner: For each $1 \leq m \leq M$ we draw a random number w_m from a normal distribution with mean 1 and variance 0.1, where numbers outside the interval $[0.1, 1.9]$ are ignored. Then, we normalize those numbers so that $\sum_{m=1}^{M} w_m = 1$. Finally, we randomly split D into m partial databases of expected sizes of $w_m N$, $1 \leq m \leq M$, as follows: Each transaction $t \in D$ is assigned at random to one of the partial databases, so that $\Pr(t \in D_m) = w_m$, $1 \leq m \leq M$.

6.3 Experimental settup

We compared the performance of two secure implementations of the FDM algorithm (Section 1.1.2). In the first implementation (denoted FDM-KC), we executed the unification step (Step 4 in FDM) using Protocol UNIFI-KC, where the commutative cipher was 1024-bit RSA [25]; in the second implementation (denoted FDM) we used our Protocol UNIFI, where the keyed-hash function was HMAC [4]. In both implementations, we implemented Step 5 of the FDM algorithm in the secure manner that was described in Section 3. We tested the two implementations with respect to three measures:

1) Total computation time of the complete protocols (FDM-KC and FDM) over all players. That measure includes the Apriori computation time, and the time to identify the globally s-frequent itemsets, as described in Section 3. (The latter two procedures are implemented in the same way in both Protocols FDM-KC and FDM.)
2) Total computation time of the unification protocols only (UNIFI-KC and UNIFI) over all players.
3) Total message size.

We ran three experiment sets, where each set tested the dependence of the above measures on a different parameter:

- N — the number of transactions in the unified database,

- M — the number of players, and
- s — the threshold support size.

In our basic configuration, we took $N = 500,000$, $M = 10$, and $s = 0.1$. In the first experiment set, we kept M and s fixed and tested several values of N. In the second experiment set, we kept N and s fixed and varied M. In the third set, we kept N and M fixed and varied s. The results in each of those experiment sets are shown in Section 6.4.

All experiments were implemented in C# (.net 4) and were executed on an Intel(R) Core(TM)i7-2620M personal computer with a 2.7GHz CPU, 8 GB of RAM, and the 64-bit operating system Windows 7 Professional SP1.

6.4 Experimental results

Figure 1 shows the values of the three measures that were listed in Section 6.3 as a function of N. In all of those experiments, the value of M and s remained unchanged — $M = 10$ and $s = 0.1$. Figure 2 shows the values of the three measures as a function of M; here, $N = 500,000$ and $s = 0.1$. Figure 3 shows the values of the three measures as a function of s; here, $N = 500,000$ and $M = 10$.

From the first set of experiments, we can see that N has little effect on the runtime of the unification protocols, UNIFI-KC and UNIFI, nor on the bit communication cost. However, since the time to identify the globally s-frequent itemsets (see Section 3) does grow linearly with N, and that procedure is carried out in the same manner in FDM-KC and FDM, the advantage of Protocol FDM over FDM-KC in terms of runtime decreases with N. While for $N = 100,000$, Protocol FDM is 22 times faster than Protocol FDM-KC, for $N = 500,000$ it is five times faster. (The total computation times for larger values of N retain the same pattern that emerges from Figure 1; for example, with $N = 10^6$ the total computation times for FDM-KC and FDM were 744.1 and 238.5 seconds, respectively, which gives an improvement factor of 3.1.)

The second set of experiments shows how the computation and communication costs increase with M. In particular, the improvement factor in the bit communication cost, as offered by Protocol UNIFI with respect to Protocol UNIFI-KC, is in accord with our analysis in Section 2.5.3. Finally, the third set of experiments shows that higher support thresholds entail smaller computation and communication costs since the number of frequent itemsets decreases.

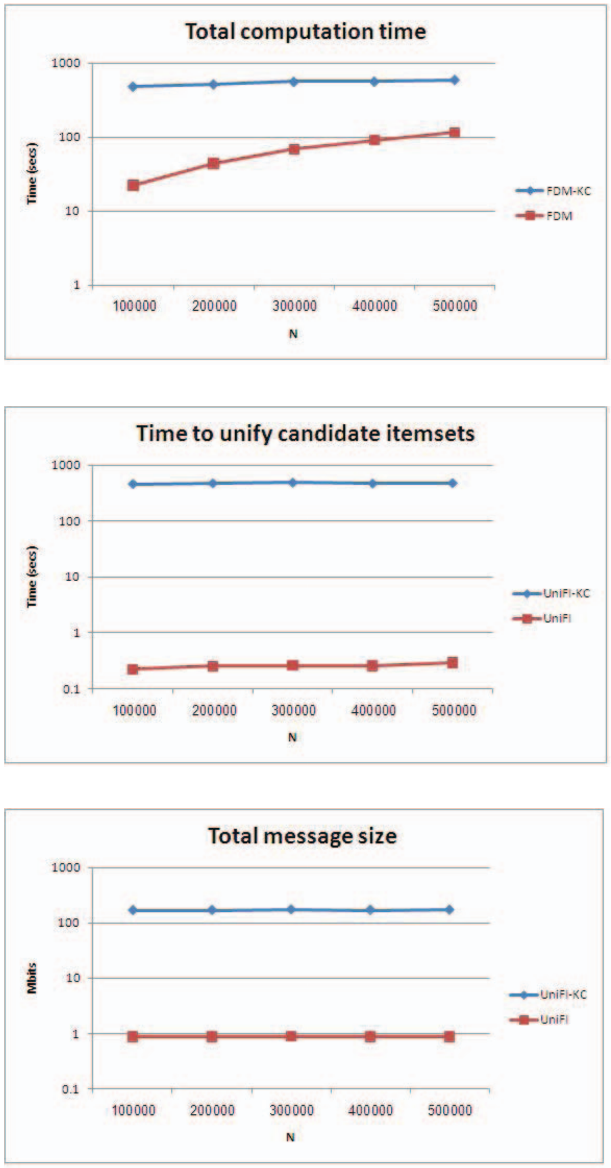

Fig. 1. Computation and communication costs versus the number of transactions N

7 RELATED WORK

Previous work in privacy preserving data mining has considered two related settings. One, in which the data owner and the data miner are two different entities, and another, in which the data is distributed among several parties who aim to jointly perform data mining on the unified corpus of data that they hold.

In the first setting, the goal is to protect the data records from the data miner. Hence, the data owner aims at anonymizing the data prior to its release. The main approach in this context is to apply data perturbation [2], [11]. The idea is that the perturbed data can be used to infer general trends in the data, without revealing original record information.

In the second setting, the goal is to perform data mining while protecting the data records of each of the data owners from the other data owners. This is a problem of secure multiparty computation. The usual approach here is cryptographic rather than probabilistic. Lindell and Pinkas [22] showed how to securely build an ID3 decision tree when the training set is distributed horizontally. Lin et al. [21] discussed secure clustering using the EM algorithm over horizontally distributed data. The problem of distributed association rule mining was studied in [19], [31], [33] in the vertical setting, where each party holds a different set of attributes, and in [18] in the horizontal setting. Also the work of [26] considered this problem in the horizontal setting, but they considered large-scale systems in which, on top of the parties that hold the data records (resources) there are also managers which

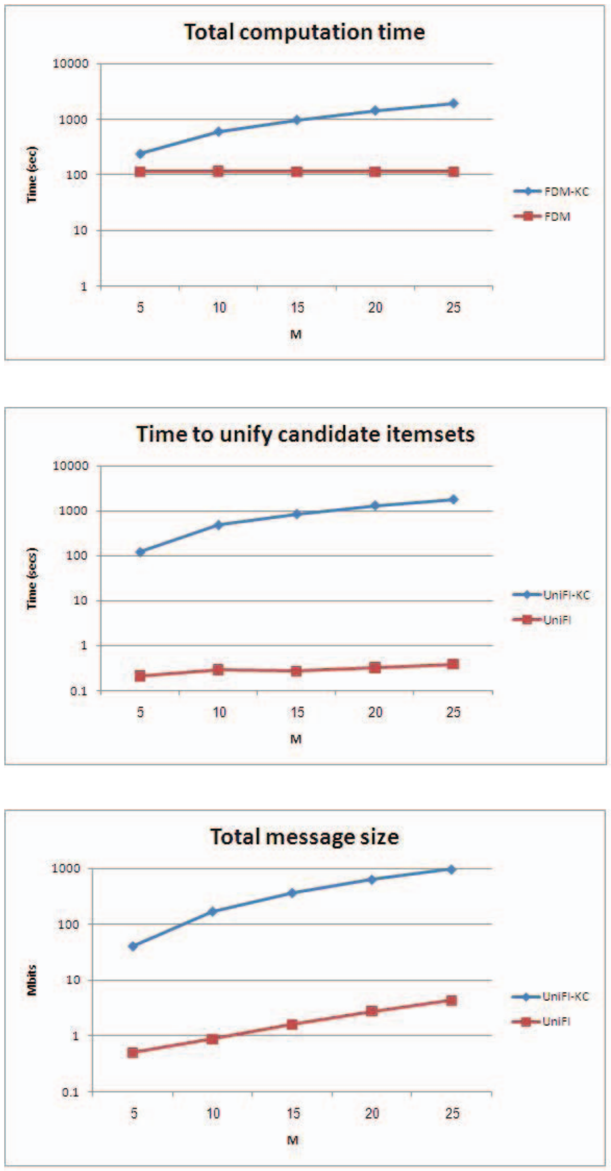

Fig. 2. Computation and communication costs versus the number of players M

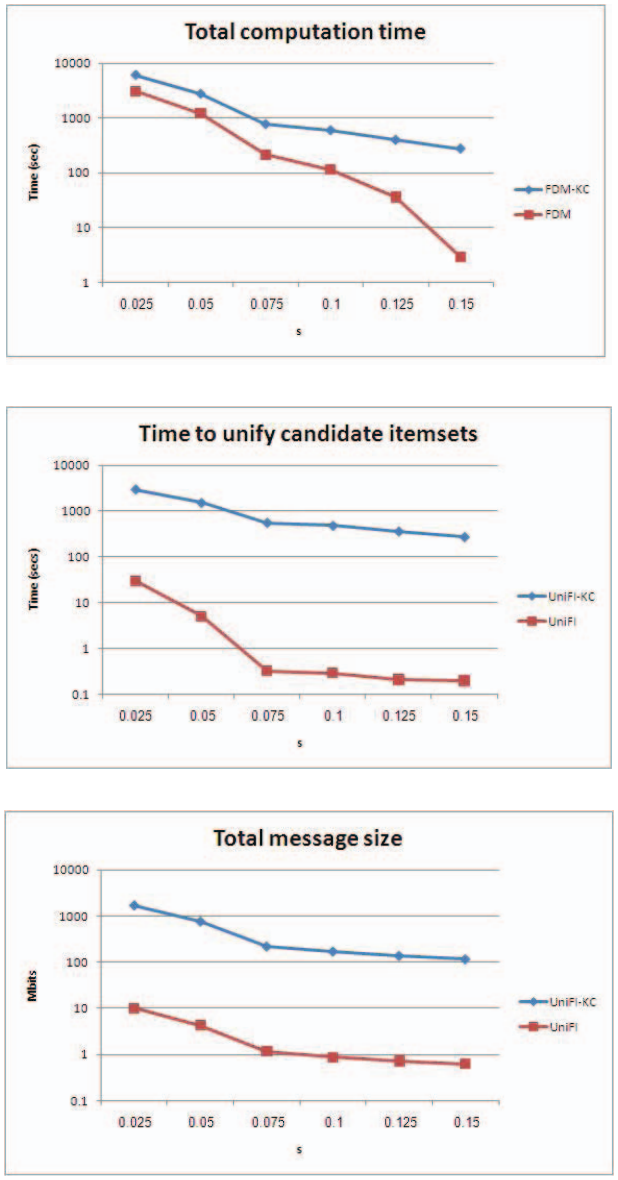

Fig. 3. Computation and communication costs versus the support threshold s

are computers that assist the resources to decrypt messages; another assumption made in [26] that distinguishes it from [18] and the present study is that no collusions occur between the different network nodes — resources or managers.

The problem of secure multiparty computation of the union of private sets was studied in [7], [14], [20], as well as in [18]. Freedman et al. [14] present a privacy-preserving protocol for set intersections. It may be used to compute also set unions through set complements, since $A \cup B = \overline{\overline{A} \cap \overline{B}}$. Kissner and Song [20] present a method for representing sets as polynomials, and give several privacy-preserving protocols for set operations using these representations. They consider the threshold set union problem, which is closely related to the threshold function (Definition 2.1). The communication overhead of the solutions in those two works, as well as in [18]'s and in our solutions, depends linearly on the size of the ground set. However, as the protocols in [14], [20] use homomorphic encryption, while that of [18] uses commutative encryption, their computational costs are significantly higher than ours. The work of Brickell and Shmatikov [7] is an exception, as their solution entails a communication overhead that is logarithmic in the size of the ground set. However, they considered only the case of two players, and the logarithmic communication overhead occurs only when the size of the intersection of the two sets is bounded by a constant.

The problem of set inclusion can be seen as a simplified version of the *privacy-preserving keyword search*. In that problem, the server holds a set of pairs $\{(x_i, p_i)\}_{i=1}^n$, where x_i are distinct "keywords", and the client holds a single value w. If w is one of the server's keywords, i.e., $w = x_i$ for some $1 \le i \le n$, the client should get the corresponding p_i. In case w differs from all x_i, the client should get notified of that. The privacy requirements are that the server gets no information about w and that the client gets no information about other pairs in the server's database. This problem was solved by Freedman et. al. [13]. If we take all p_i to be the empty string, then the only information the client gets is whether or not w is in the set $\{x_1, \ldots, x_n\}$. Hence, in that case the privacy-preserving keyword search problem reduces to the set inclusion problem. Another solution for the set inclusion problem was recently proposed in [30], using a protocol for oblivious polynomial evaluation.

8 Conclusion

We proposed a protocol for secure mining of association rules in horizontally distributed databases that improves significantly upon the current leading protocol [18] in terms of privacy and efficiency. One of the main ingredients in our proposed protocol is a novel secure multi-party protocol for computing the union (or intersection) of private subsets that each of the interacting players hold. Another ingredient is a protocol that tests the inclusion of an element held by one player in a subset held by another. Those protocols exploit the fact that the underlying problem is of interest only when the number of players is greater than two.

One research problem that this study suggests was described in Section 3; namely, to devise an efficient protocol for inequality verifications that uses the existence of a semi-honest third party. Such a protocol might enable to further improve upon the communication and computational costs of the second and third stages of the protocol of [18], as described in Sections 3 and 4. Other research problems that this study suggests is the implementation of the techniques presented here to the problem of distributed association rule mining in the vertical setting [31], [33], the problem of mining generalized association rules [27], and the problem of subgroup discovery in horizontally partitioned data [16].

Acknowledgement. The author thanks Keren Mendiuk for her help in conducting the experiments.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.
[2] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD Conference*, pages 439–450, 2000.
[3] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC*, pages 503–513, 1990.
[4] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Crypto*, pages 1–15, 1996.
[5] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP - A system for secure multi-party computation. In *CCS*, pages 257–266, 2008.
[6] J.C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Crypto*, pages 251–260, 1986.
[7] J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT*, pages 236–252, 2005.
[8] D.W.L. Cheung, J. Han, V.T.Y. Ng, A.W.C. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *PDIS*, pages 31–42, 1996.
[9] D.W.L Cheung, V.T.Y. Ng, A.W.C. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. *IEEE Trans. Knowl. Data Eng.*, 8(6):911–922, 1996.
[10] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985.
[11] A.V. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *KDD*, pages 217–228, 2002.
[12] R. Fagin, M. Naor, and P. Winkler. Comparing Information Without Leaking It. *Communications of the ACM*, 39:77–85, 1996.
[13] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.
[14] M.J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.
[15] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
[16] H. Grosskreutz, B. Lemmen, and S. Rüping. Secure distributed subgroup discovery in horizontally partitioned data. *Transactions on Data Privacy*, 4:147–165, 2011.
[17] W. Jiang and C. Clifton. A secure distributed framework for achieving k-anonymity. *The VLDB Journal*, 15:316–333, 2006.
[18] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering*, 16:1026–1037, 2004.
[19] M. Kantarcioglu, R. Nix, and J. Vaidya. An efficient approximate protocol for privacy-preserving association rule mining. In *PAKDD*, pages 515–524, 2009.
[20] L. Kissner and D.X. Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.
[21] X. Lin, C. Clifton, and M.Y. Zhu. Privacy-preserving clustering with distributed EM mixture modeling. *Knowl. Inf. Syst.*, 8:68–81, 2005.
[22] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Crypto*, pages 36–54, 2000.
[23] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash based algorithm for mining association rules. In *SIGMOD Conference*, pages 175–186, 1995.
[24] S.C. Pohlig and M.E. Hellman. An improved algorithm for computing logarithms over $gf(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.
[25] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.
[26] A. Schuster, R. Wolff, and B. Gilburd. Privacy-preserving association rule mining in large-scale distributed systems. In *CCGRID*, pages 411–418, 2004.
[27] R. Srikant and R. Agrawal. Mining generalized association rules. In *VLDB*, pages 407–419, 1995.
[28] T. Tassa and D. Cohen. Anonymization of centralized and distributed social networks by sequential clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2012.
[29] T. Tassa and E. Gudes. Secure distributed computation of anonymized views of shared databases. *Transactions on Database Systems*, 37, Article 11, 2012.
[30] T. Tassa, A. Jarrous, and J. Ben-Ya'akov. Oblivious evaluation of multivariate polynomials. *Submitted*.
[31] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, pages 639–644, 2002.
[32] A.C. Yao. Protocols for secure computation. In *FOCS*, pages 160–164, 1982.
[33] J. Zhan, S. Matwin, and L. Chang. Privacy preserving collaborative association rule mining. In *Data and Applications Security*, pages 153–165, 2005.
[34] S. Zhong, Z. Yang, and R.N. Wright. Privacy-enhancing k-anonymization of customer data. In *PODS*, pages 139–147, 2005.

PLACE
PHOTO
HERE

Tamir Tassa is an Associate Professor at the Department of Mathematics and Computer Science at The Open University of Israel. Previously, he served as a lecturer and researcher in the School of Mathematical Sciences at Tel Aviv University, and in the Department of Computer Science at Ben Gurion University. During the years 1993-1996 he served as an assistant professor of Computational and Applied Mathematics at University of California, Los Angeles. He earned his Ph.D. in applied mathematics from the Tel Aviv University in 1993. His research interests include cryptography, privacy-preserving data publishing and data mining.