Secure Construction of Contingency Tables from Distributed Data

Haibing Lu   Xiaoyun He   Jaideep Vaidya   Nabil Adam

MSIS Department and CIMIC, Rutgers University, USA
{haibing, xiaoyun, jsvaidya, adam}@cimic.rutgers.edu

Abstract. Contingency tables are widely used in many fields to analyze the relationship or infer the association between two or more variables. Indeed, due to their simplicity and ease, they are one of the first methods used to analyze gathered data. Typically, the construction of contingency tables from source data is considered straightforward since all data is supposed to be aggregated at a single party. However, in many cases, the collected data may actually be federated among different parties. Privacy and security concerns may restrict the data owners from free sharing of the raw data. However, construction of the global contingency tables would still be of immense interest. In this paper, we propose techniques for enabling secure construction of contingency tables from both horizontally and vertically partitioned data. Our methods are efficient and secure. We also examine cases where the constructed contingency table may itself leak too much information and discuss potential solutions.

1 Introduction

Contingency tables have been widely used in a number of application domains, including social science [11] , epidemiology [8] , experimental studies of economics [9], etc. Simply put, a contingency table is a table of frequency counts (i.e., Figure 2), which is often used to analyze the relationship or infer the association between two or more variables. The construction of contingency tables from a source data is considered straightforward - i.e., in the two variable case, listing all the levels of one variable as rows and the levels of the other variables as columns in a table, then finding the joint frequency for each cell. The underlying assumption of such computation is that the original data is centralized at one site or owned by a single party.

However, there are many situations where we may want to construct contingency tables from multiple data sources and/or ownerships. For example, in the health care domain, each hospital holds its patients' medical records. Consider a scenario where doctors are trying to find out the relationship between a certain rare disease and the effectiveness of different treatments. Given the small number of instances, it is beneficial for all the hospitals to start with constructing contingency tables from their combined medical records. However, in these circumstances, the privacy of the patients is a major concern, which may prevent the eventual collaboration between hospitals. Similar conflicts have been observed in other domains, such as financial services, telecommunications, and government agencies ([3], [6]). Therefore, how to solve such problems in a privacy-preserving way is an emerging issue. In this regard, privacy-preserving data mining is a very closely related area, and the work in this area is quite relevant.

During the past years, privacy-preserving data mining [24] has attracted much attention from the research community since the seminal papers by Agrawal and Srikant [2] and Lindell and Pinkas [18]. Primarily the focus has been on creating privacy-preserving variants of different data mining tasks. Two main solution approaches have been followed. In the randomization approach, "noise" is added to the data before the data mining process, and then reconstruction techniques are used to mitigate the impact of the noise from the data mining results[2, 1, 10, 22]. However, there is some debate about the security properties of such algorithms[17, 15]. On the other had, cryptographic solutions following the secure multiparty computation framework ([25], [14], [13]) aim to achieve "perfect" privacy and limit disclosure only to information that can be inferred from each participant's own input and the results. Given that the general method for secure multiparty computations does not scale well to large dataset problems, a number of efficient methods (i.e., secure sum, secure size of set intersection, secure scalar product, etc.) have been developed ([7], [12]). These methods demonstrate provable privacy on individual information and bounds on information released. Another important fact is their applicability - we can use them as primitive tools to develop secure solutions for some specific applications. We follow this approach for our work.

In this paper, we present solutions to construct a general n-way contingency tables from distributed data in a privacy-preserving way. Two solutions are presented. The first solution assumes that the data is horizontally partitioned between parties, where different data objects with the same attributes are owned by each party. For this approach, we follow the underlying idea of secure sum protocol discussed in [7]. The second one focuses on the vertically partitioned data, in which different attributes for the same set of data objects are owned by each party. This solution is based on the secure scalar protocol [12]). In the horizontal partition case, we assume that there are three or more parties involved. Clearly, in the two-party case, one can first construct its local contingency table. Subtraction of the local table from the global contingency table reveals the other party's contingency table. Therefore much of the information we try to protect is revealed even though we follow a completely secure protocol to compute the global result. Clearly, at least three parties are necessary for security. There is no such constraint for the vertically partitioned case.

The remainder of this paper is organized as follows. We first formally define the problem in Section 2. In Section 3 we present the proposed algorithms for secure contingency tables computation. The solutions are presented for horizontally partitioned data as well as vertically partitioned data. Along with the algorithms, a detailed computation cost analysis and security analysis is provided. Finally, Section 4 concludes the paper and provides directions for future work.

2 Problem Definition

Consider data such as shown in Figure 1, where A_j denotes the jth attribute, R_i denotes the ith record, and v_{ij} denotes the value of the jth attribute for the ith record. We

	A_1	A_2	...	A_n
R_1	v_{11}	v_{12}	...	v_{1n}
R_2	v_{21}	v_{22}	...	v_{2n}
...	...	...	...	...
R_m	v_{m1}	v_{m2}	...	v_{mn}

Fig. 1. Categorical Data Table

assume that each attribute is categorical. Therefore, each value v_{ij} is a nominal value. In itself, this table is sufficient to construct a contingency table which can be used for information processing, like extraction of association rules or statistical analysis. However, consider that several parties collectively gather this data. Thus, each party independently possesses only part of the data – either several rows or several columns. Due to privacy/security concerns, the parties are not willing to release their raw data to the other parties or to any outside third party. However, they may wish to perform global data analysis using contingency tables or even be willing to allow a third party to do such analysis as long as it only gets the data analysis results as opposed to the raw data. Though the parties could compute their local contingency tables, clearly these could be very different from the global table and thus lead to quite inaccurate results. Therefore, they wish to compute the global contingency table in a privacy-preserving fashion. We denote this as the problem of **Secure Integration**. Specifically, this problem can be divided as **Secure Horizontal Integration**, where each party owns several rows, and **Secure Vertical Integration**, where each party owns several columns.

In this paper, we assume that all attributes are categorical – thus all values are nominal. In general, it is easily possible to discretize numerical data to form categorical attributes as well. Now to compute the contingency table of Figure 1, we only need to count the number of records having the same attributes values. Formally, assume attribute A_i has d_i distinct values, denoted by $\{a_{i,1}, a_{i,2}, ..., a_{i,d_i}\}$. Thus, the contingency table resulting from the integral table in Figure 1 is a n-dimensional matrix $C_{d_1 \times d_2 ... \times d_n}$, where cell $c_{j_1,j_2,...,j_n}$ denotes the count of records in the table having the value $\{a_{1,j_1}, a_{2,j_2}, ..., a_{n,j_n}\}$. Obviously, the sum of all of the cells is the total number of records.

We now illustrate this with an example. Figure 2 shows data records on shopping lists, along with its corresponding contingency table. The shopping list collects data on two attributes, the drinks and fruits that are bought. Here, Drink has distinct values ”Beer” and ”Coke”, and Fruit has distinct values ”Apple” and ”Orange”. After counting the records having the same values, we have its contingency table, which is a 2×2 table shown on the right.

In the following, we will give the formal definition of contingency table and the corresponding secure integration problems as well.

Definition 1 (Contingency Table). *Given a dataset of m records and n attributes (as shown in Figure 1), where the distinct values of attribute A_i are denoted by $\{a_{i,1}, a_{i,2}, ..., a_{i,d_i}\}$, its contingency table is defined as a n-dimensional matrix $C_{d_1 \times d_2 ... \times d_n}$, where cell $c_{j_1,j_2,...,j_n}$ denotes the count of records having the value $\{a_{1,j_1}, a_{2,j_2}, ..., a_{n,j_n}\}$.*

	Drink	Fruit
R_1	Beer	Apple
R_2	Coke	Apple
R_3	Coke	Orange
R_4	Beer	Apple

	Beer	Coke
Apple	2	1
Orange	0	1

Fig. 2. A Shopping List Table and its Contingency Table

Definition 2 (Construct Contingency Table on Horizontally Partitioned Data). *The global dataset (such as shown in Figure 1) is shared by many parties separately, each of whom owns different set of data objects with the same set of attributes. The parties want to construct the contingency table of the whole table together securely without letting others know the detailed data they own.*

Definition 3 (Construct Contingency Table on Vertically Partitioned Data). *The global dataset (such as shown in Figure 1) is shared by many parties separately, each of whom owns different set of attributes but for the same set of data objects. The parties want to construct the contingency table of the whole table together securely without letting others know the detailed data they own.*

The complete n-way contingency table can also be used to compute smaller contingency tables (for example, a 2-way contingency table looking at the correlation of two attributes) simply by summing up over the cells of all the other attributes. However, it might be useful to directly compute the smaller contingency tables. This can easily be done by using the same protocols but on a reduced subset of the data.

3 Secure Construction of Contingency Tables

As we have mentioned earlier, a contingency table is basically a table of counts. The count in a cell with respect to two or more attribute values is computed as the total number of co-occurrences of these values in a dataset. This seems simple for the centralized data. For computing the contingency tables from distributed data, a global view of the data needs to be composed by combining all the individual data belonging to different parties. However, these parties may not be willing to share/reveal their own data for reasons discussed in Section 1. The proposed approaches in this section enable computation of the contingency tables from the distributed data without requiring parties to reveal any details about their own data.

In a distributed environment, different models for data partitioning have been proposed (i.e.,[24], [16]). Here, we consider the two most common and practical models - horizontal partitioning and vertical partitioning of data. In each case, we present secure protocols for the general case of computing n-way contingency tables in a decentralized manner. In the following, we describe each of the protocols in more detail and illustrate them with examples.

3.1 On Horizontally Partitioned Data

Algorithm In the horizontally partitioned data case, each party owns different set of data objects with the same set of attributes. The protocol for secure computation of contingency tables from horizontally partitioned data is depicted in Algorithm 1. This protocol is run by all k parties. We assume there are n attributes, namely, $A_1, A_2, \ldots, A_n$, in common for all the parties. Each attribute can take a number of distinct values, i.e., for A_i, these values are denoted by $a_{i,1}, a_{i,2}, \ldots, a_{i,d_i}$, that is, there are a total of d_i distinct values for A_i. Hence, the general contingency tables from such data can be represented as n-way $d_1 \times d_2 \times \cdots \times d_n$ contingency table, denoted by CT.

Given that the data is horizontally partitioned among the parties, it is possible for each party P_i first locally computes a n-way contingency table LCT_i from its own data. LCT_i is equivalent to CT in dimension but only includes the count of the attribute values co-occurring in the data owned by P_i. Then, the next step is to securely sum the counts of the corresponding cells in all LCT_i along each dimension. The spirit of the performed secure operations follows that of the secure sum protocol presented in [7]. The basic idea is to designate P_1 as the master site. P_1 generates a n-way matrix of random numbers R, uniformly chosen from $[0 \ldots z-1]$ (we assume that this is the range in which all the cell count values fall). Then, for any given cell count c in LCT_1, P_1 adds the corresponding random number r to it - $(r+c) \bmod z$, and sends its therefore randomized local contingency table RCT_1 to the next party, say P_2. P_2 will learn nothing about the actual cell counts in LCT_1. This is because P_1 has uniformly at random chosen r from range $[0 \ldots z-1]$, and all of the randoms are different for each cell. Therefore, for each cell, the number $(r + c) \bmod z$ is also uniformly distributed across the range.

From P_2 to P_{k-1}, each party does the same operations as follows. Party P_i receives RCT_{i-1} from P_{i-1}, and adds the cell counts in this randomized contingency table to the corresponding cell counts in its local contingency table LCT_i, resulting in RCT_i. Again, since all the cell values in RCT_{i-1} are uniformly distributed over the range $[0 \ldots z-1]$, P_i learns nothing from it. Then, P_i passes RCT_i to next party P_{i+1}.

After receiving RCT_{k-1} from P_{k-1}, P_k performs the same sum mentioned above and sends the resulting table RCT_k back to P_1. Finally, P_1, knowing the random matrix R, subtracts each corresponding r from the cell of RCT_k to get the actual contingency table CT.

Example In this section, we will give an example to illustrate Algorithm 1. The tables on the left of Figures 3, 4 and 5 are patient treatment response tables possessed by three different hospitals, where each record represents a patient and their identifications are suppressed. The tables on the right are their associated contingency tables. For the convenience of reading, we display 3-dimensional tables in a 2-dimensional way.

Now, we show how to employ Algorithm 1 to construct a contingency table on these three separated tables. First, the holder of Figure 3 generates a random table. Suppose it is as the table on the left of Figure 6. Then the holder of Figure 3 adds the random data to his original data and passes the result to the holder of Figure 4. As the holder of Figure 4 knows that the received data having been masked, he cannot figure out the real table. Then according to the protocol, he adds his data to the received table and passes

Algorithm 1 Secure Construction of Contingency Tables for Horizontally Partitioned Data

```
Require: $k$ parties $P_1, \dots, P_k$.
Require: n attributes $A_1, \dots, A_n$. For each attribute $A_i$, there is a set of distinct values
    $a_{i,1}, a_{i,2}, \dots, a_{i,d_i}$ $(i = 1, 2, \dots, n)$.
Require: The counts in the computed contingency tables lie in the range $[0..z]$
Require: OUTPUT: n-way $d_1 \times d_2 \times \cdots \times d_n$ contingency table $CT$.
 1: for $i \leftarrow 1 \dots k$ do
 2:     At $P_i$: Compute the local n-way $d_1 \times d_2 \times \cdots \times d_n$ contingency table $LCT_i$.
 3: end for
 4: $P_1$ generates a n-way $(d_1 \times d_2 \times \cdots \times d_n)$ matrix of random numbers $r$, uniformly chosen
    from $[0..z]$.
 5: At $P_1$: Given any local cell value $c$ in $LCT_i$, add the corresponding random number $r$ from
    the random matrix to compute the sum $(r + c) \bmod z$, resulting in the contingency table
    $RCT_1$
 6: $P_1$ sends $RCT_1$ to $P_2$
 7: for $i \leftarrow 2 \dots k-1$ do
 8:     At $P_i$: Sums each cell's count in its local $LCT_i$ with the corresponding cell count in
        $RCT_{i-1}$, resulting in the contingency table $RCT_i$.
 9:     $P_i$ passes $RCT_i$ to $P_{i+1}$.
10: end for
11: $P_k$ performs the above sum operation and sends the resulting contingency table $RCT_k$ to
    $P_1$.
12: $P_1$ subtracts the corresponding $r$ (of the random matrix) from each cell in $RCT_k$ and gets
    the result $CT$.
13: return $CT$
```

it to the holder of Figure 5. Then, the holder of Figure 5 does the same thing and passes the result to the first person. The returned table is the table on the right of of Figure 6. Finally, the first person subtracts the returned table by the random table generated by him and gets the final contingency table.

3.2 On Vertically Partitioned Data

Algorithm When the data is vertically partitioned among k parties, it is assumed that each party owns different set of attributes but for the same set of data objects. Let the overall naturally ordered attribute set be $\{A_1, \dots, A_n\}$, and for each attribute A_i, there is a set of distinct values $\{a_{i,1}, a_{i,2}, \dots, a_{i,d_i}\}$ $(i = 1, 2, \dots, n)$. For simplicity, we assume that P_1 owns the consecutive attributes $\{A_1, \dots, A_{p_1}\}$, P_2 owns $\{A_{p_1+1}, \dots, A_{p_2}\}$, …, P_k owns $\{A_{p_{k-1}+1}, \dots, A_n\}$. Let the number of participating data objects be m. Indeed, this is not a restriction, since, for a large dataset, we can divide the data into chunks of size m, and invoke the protocol on the chunks. Then, we can sum the resulting sub-contingency tables. In the end, the result is a n-way $d_1 \times d_2 \times \cdots \times d_n$ contingency table CT.

In order to compute the count in cell $c_{j_1, j_2, \dots, j_n}$ of $CT(1 \le j_1 \le d_1, 1 \le j_2 \le d_2, \dots, 1 \le j_n \le d_n)$, each party P_i first needs to perform the following local computations in order to get a representative vector X_i. The first step is that, for each of its

	Center	Treatment	Response
R_1	1	1	2
R_2	2	1	1
R_3	2	2	2

Center	Treatment	Response 1	Response 2
1	1	0	1
1	2	0	0
2	1	1	0
2	2	0	1

Fig. 3. Treatment Response Data and Contingency Table I

	Center	Treatment	Response
R_4	2	1	2
R_5	1	1	2
R_6	2	2	1

Center	Treatment	Response 1	Response 2
1	1	0	1
1	2	0	0
2	1	0	1
2	2	1	0

Fig. 4. Treatment Response Data and Contingency Table II

attributes A_s ($p_{(i-1)} + 1 \leq s \leq p_i$), P_i transforms the corresponding attribute values in its data into a binary vector $V_s = \{v_{s,1}, \ldots, v_{s,m}\}$. Specifically, $v_{s,t}$ is set to 1 if the t-th data value of A_s equals $a_{s,j_s}(1 \leq t \leq m)$); otherwise 0. Then, for all $\boldsymbol{V_s}$, P_i locally computes their product such that $X_i = \{x_{i,1}, \ldots, x_{i,m}\}$ and $x_{i,t} = \prod_{s=p_{(i-1)}+1}^{p_i} v_{s,t}$.

After the above local computations, all the parties engage in a secure k-vector product protocol, as described in algorithm 3, with their respective input vector $X_i(i = 1, \ldots, k)$. The result of the secure k-vector product protocol is the count in cell $c_{j_1,j_2,\ldots,j_n}$. All the other cells' count of CT can be done in the same way. Clearly, the security of the protocol completely depends on that of the secure k-vector product protocol, which we shall elaborate below.

Secure k-Vector Product Protocol In this section, we discuss and present the solutions for securely computing the vector product problem.

First, let us consider k = 2 case, which is also known as scalar or dot product. Assume that party P_1 has vector X while party P_2 has vector Y, and each vector has the cardinality n. Let $\boldsymbol{X} = (x_1, \ldots, x_n)$, $\boldsymbol{Y} = (y_1, \ldots, y_n)$. The scalar product of vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ is defined as:

$$\sum_{i=1}^{n} x_i * y_i$$

The goal of the secure computation is that, at the end of the protocol, each party would get $\boldsymbol{X} \cdot \boldsymbol{Y}$ while knowing nothing about the other party's vector. The protocol proposed by Goethals et al. [12] is quite simple and provably secure. The main idea behind the protocol is to use a homomorphic encryption system including the Benaloh cryptosystem [4], the Naccache-Stern cryptosystem [19], the Paillier cryptosystem [21], the Okamoto-Uchiyama cryptosystem [20], and the Goldwasser-Micali cryptosystem [5]. Besides the standard guarantees, homomorphic encryption, as a semantically-secure public-key encryption, has the additional property that given any two encryptions $E(A)$ and $E(B)$, there exists an operation $\otimes$ such that $E(A) \otimes E(B) = E(A * B)$, where $*$ is either addition or multiplication (in some abelian group). The cryptosystems mentioned

	Center	Treatment	Response
R_7	1	1	2
R_8	1	1	2
R_9	2	2	2

Center	Treatment	Response 1	Response 2
1	1	0	2
1	2	0	0
2	1	0	0
2	2	0	1

Fig. 5. Treatment Response Data and Contingency Table III

Center	Treatment	Response 1	Response 2
1	1	1	1
1	2	2	0
2	1	2	0
2	2	1	2

Center	Treatment	Response 1	Response 2
1	1	1	5
1	2	2	0
2	1	3	1
2	2	2	4

Fig. 6. Generated Random Table and Returned Table

above are additively homomorphic (thus the operation $*$ denotes addition, and the operation $\otimes$ denotes multiplication). Using such a system, it is quite simple to create a secure scalar product protocol. The key is to note that $\sum_{i=1}^{n} x_i \cdot y_i = \sum_{i=1}^{n}(x_i + x_i + \cdots + x_i)$ (y_i times). If P_1 encrypts her vector and sends in encrypted form to P_2, P_2 can use the additive homomorphic property to compute the scalar product.

In the following, we extend the idea behind the above secure scalar product protocol to securely compute k-vector product ($k \geq 3$), which is defined as follows.

Assume that there are k parties ($P_1, P_2, \ldots, P_k$), where each party P_i has a (0, 1) vector $\boldsymbol{X_i}$ of cardinality n. Let $\boldsymbol{X_i} = \{x_{i,1}, \ldots, x_{i,n}\}$ ($i = 1, \ldots, k$). Our goal here is to securely compute $\sum_{j=1}^{n} \prod_{i=1}^{k} x_{i,j}$ without requiring each party to disclose its vector.

The key to computing this securely lies in the fact that each row contributes a 1 to the final answer, if and only if, each party has a 1 for that row. The key is to keep this information secure. The protocol starts with one party, say P_1, who first generates a private and public key pair (sk, pk) for a semantically secure homomorphic encryption system and sends pk to other parties. Then, P_1 encrypts each of its vector elements $x_{1,j}$ and sends the encrypted value $w_{1,j} = E_{pk}(x_{1,j})$ ($j = 1, \ldots, n$) to P_2. For each j, if $x_{2,j}$ = 0, P_2 sends to P_3, $E_{pk}(0)$; otherwise, it sends to P_3 $E_{pk}(x_{1,j}) \cdot E_{pk}(0)$ – this effectively hides the value it has received from its neighbor. To see how this gives the right answer, recall that the vectors contain values of either 0 or 1. If P_2 has 0 as the current value of $x_{2,j}$, then no matter what values the other parties have, $\prod_{i=1}^{k} x_{i,j} = 0$. Therefore, P_2 can send out $E_{pk}(0)$. On the other hand, if P_2 has 1 as the current value, then $\prod_{i=1}^{k} x_{i,j} = x_{1,j} \cdot \prod_{i=3}^{k} x_{i,j}$, and, based on the additive homomorphic property, $E_{pk}(x_{1,j}) \cdot E_{pk}(0) = E_{pk}(x_{1,j})$. That is, it doesn't affect the final result. However, this way makes the computations secure and prevents collusion (i.e., between P_1 and P_3, since it hides the value sent by P_1 to P_2). In both cases, P_2 sends out different encrypted values from those of P_1. Therefore, the other parties will be unable to figure out the actual values, even when they collude.

The above operations done by P_2 are repeated on by the following party $P_i(i = 3, \ldots, k_1)$, one following the other, on its own vector. Finally, P_k, who finally de-

Algorithm 2 Secure Construction of Contingency Tables for Vertically Partitioned Data

```
Require: $k$ parties $P_1, \dots, P_k$
Require: n attributes $A_1, \dots, A_n$. For each attribute $A_i$, there is a set of distinct values
    $\{a_{i,1}, a_{i,2}, \dots, a_{i,d_i}\}$ $(i = 1, 2, \dots, n)$. The number of data objects is $m$.
Require: For simplicity, we assume that $P_1$ owns $\{A_1, \dots, A_{p_1}\}$, $P_2$ owns $\{A_{p_1+1}, \dots, A_{p_2}\}$,
    $\dots$, $P_k$ owns $\{A_{p_{k-1}+1}, \dots, A_n\}$.
Require: OUTPUT: n-way $d_1 \times d_2 \times \dots \times d_n$ contingency table $CT$.
1: For any given cell $c_{j_1,j_2,\dots,j_n}$ of $CT(1 \le j_1 \le d_1, 1 \le j_2 \le d_2, \dots, 1 \le j_n \le d_n)$, its
   count is computed as follows:
2: for $i \leftarrow 1 \dots k$ do
3:     At $P_i$: encode the data values of its attribute $A_s$ $(p_{(i-1)} + 1 \le s \le p_i)$ into a binary vector
       $\boldsymbol{V_s}$ of size $m$ such that $V_s = \{v_{s,1}, \dots, v_{s,m}\}$ and $v_{s,t}$ = 1 if the $t$-th data value of $A_s$
       equals $a_{s,j_s}(1 \le t \le m))$ ; otherwise 0.
4:     At $P_i$: Locally compute the product $X_i$ of all $\boldsymbol{V_s}$ such that $X_i = \{x_{i,1}, \dots, x_{i,m}\}$ and
       $x_{i,t} = \prod_{s=p_{(i-1)}+1}^{p_i} v_{s,t}$.
5: end for
6: $P_1, \dots, P_k$ invoke the secure vector product protocol (algorithm 3) to compute $VP$ =
   $\sum_{t=1}^{m} \prod_{i=1}^{k} x_{i,t}$, which is the count for cell $c_{j_1,j_2,\dots,j_n}$.
7: Compute all the other cells' count of $CT$ in the same way.
8: return $CT$
```

cides on $w_{k,j}$, computes $w = \prod_{j=1}^{n} w_{k,j}$ sends w back to P_1. P_1 decrypts it using her private key and, again, based on additive homomorphic property, gets the result of $\sum_{j=1}^{n} \prod_{i=1}^{k} x_{i,j}$.

The specific details of the protocol are given in Algorithm 3. One problem lies with collusion. Since P_1 owns the secret key corresponding to the public key, it can easily decrypt any of the intermediate messages. Thus, P_1 can collude with other parties to breach the security of the protocol. However, this can be avoided by using threshold encryption. In threshold encryption, all parties own the public key, but the decryption key is split between all parties so that at least a certain number of parties (over a threshold) are required to successfully decrypt a message. This can effectively remove the problem of collusion.

Example In this section, we will give an example to illustrate Algorithm 2. The table on the left of Figure 7 is the global table. There are three parties, each of whom holds one attribute (i.e., one column). To get the contingency table, they follow Algorithm 2 exactly. For illustration , we show the procedures of calculating the count of records having value (center=1, treatment=1, response=1). For that particular value, three parties have the corresponding vectors $\{1, 0, 0, 0, 1, 0, 1, 1, 0\}$, $\{1, 1, 0, 1, 1, 0, 1, 1, 0\}$ and $\{0, 1, 0, 0, 0, 1, 0, 0, 0)\}$ respectively. The product of these three vectors is the count. According to Algorithm 3, party one generates a pair of public key pb and private key pv and passes the encrypted message $\{E_{pb}(1, r_{11}), E_{pb}(0, r_{12}), ..., E_{pb}(0, r_{19})\}$ to party two. Party two receives the message and executes the operations associative with its own values by Algorithm 3. For example, for the first component, as the value is 1, encrypt $E_{pb}(1, r'_{21})$ and multiply with $E(1, r_{11})$ to get $E(1, r_{21})$. For the second

Algorithm 3 Secure k-Vector Product Protocol

```
Require: $k$ parties $P_1, \dots, P_k$.
Require: Each party $P_i$ has input vector $\boldsymbol{X_i} = \{x_{i,1}, \dots, x_{i,n}\}$ $(i = 1, \dots, k)$
Require: $P_1, P_2, \dots, P_k$ get the output $VP = \sum_{j=1}^{n} \prod_{i=1}^{k} x_{i,j}$
 1: $P_1$ generates a private and public key pair $(sk, pk)$ for a semantically secure homomorphic
    encryption system.
 2: $P_1$ broadcasts $pk$ to $P_2, \dots, P_k$.
 3: for $j = 1 \dots n$ do
 4:   $P_1$ sends to $P_2$ $w_{1,j} = E_{pk}(x_{1,j})$.
 5: end for
 6: for $i = 2 \dots k - 1$ do
 7:   At $P_i$:
 8:   for $j = 1 \dots n$ do
 9:     if $x_{i,j}$= 0 then
10:       $w(i, j) = E_{pk}(0)$.
11:     else
12:       $w(i, j) = E_{pk}(w_{i-1,j}) \cdot E_{pk}(0)$
13:     end if
14:     $P_i$ sends to $P_{i+1}$ $w_{i,j}$
15:   end for
16: end for
17: At $P_k$:
18: for $j = 1 \dots n$ do
19:   if $x_{i,j}$= 0 then
20:     $w(k, j) = E_{pk}(0)$.
21:   else
22:     $w(k, j) = E_{pk}(w_{i-1,j}) \cdot E_{pk}(0)$
23:   end if
24: end for
25: $P_k$ computes $w = \prod_{j=1}^{n} w_{k,j}$
26: $P_k$ sends $w$ to $P_1$
27: $P_1$ computes $VP = D_{sk}(w) = \sum_{j=1}^{n} \prod_{i=1}^{k} x_{i,j}$.
```

	Center	Treatment	Response
R_1	1	1	2
R_2	2	1	1
R_3	2	2	2
R_4	2	1	2
R_5	1	1	2
R_6	2	2	1
R_7	1	1	2
R_8	1	1	2
R_9	2	2	2

Center	Treatment	Response 1	Response 2
1	1	0	4
1	2	0	0
2	1	1	1
2	2	1	2

Fig. 7. Vertically Partitioned Table and Resulting Contingency Table

component, as the value is 0, generate $E_{pb}(0, r_{22})$ directly. Following this algorithm, party two passes the data to party three finally. Party three does the same operations and gets $\{E_{pb}(0, r_{31}), E_{pb}(0, r_{32}), ..., E_{pb}(0, r_{39})\}$. Then multiply them together to get $E_{pb}(0, r'')$ and send it back to party one. Party one uses his private key to decrypt it and gets the final product value 0. The same operations are repeated for the other cells of the contingency table. The final result is the table on the right of Figure 7.

3.3 Communication and Computation Costs

We now give cost estimates for constructing contingency tables using the protocols we have presented. Let the number of participating parties be k. The total number of cells in the resulting contingency table is $d = d_1 \times d_2 \times \cdots \times d_n$.

First, we analyze the cost for the horizontal partition case. The dominating cost for algorithm 1 is communication cost. Let u be the number of bits in representing the count values in cells of the contingency table. Then, the total bits in order to pass the whole contingency table is $(d * u)$. In the protocol, there are k passes of the contingency table around the parties. Therefore, the protocol requires $(d * u * k)$ bits of communication. Clearly, the cost mainly depends on the dimension of the contingency table. We can see there is a tradeoff here. If a higher dimension contingency table is needed, it would incur higher communication cost. Computation cost is not significant since the only computation carried out are a series of sums.

For the protocol on the vertically partitioned data, we analyze the cost in terms of the following actual operations: encryptions, multiplications, and decryptions. This is because these are the dominating factors in the protocol. As we have mentioned in the above section, the secure construction of contingency tables on vertically partitioned data makes use of the secure vector product protocol given in algorithm 3, which is also the only part involving the secure computations for computing a cell count. Given the k parties, each has a vector of size m after some local computations. Then, for each cell in the contingency table, all the parties engage in the k-vector product protocol, which requires $m * k$ encryptions, $m * (k * p + 1)$ multiplications (where p is the percentage of 1's in the vectors), and 1 decryption. Therefore, for constructing the contingency table with d cells, the total number of encryptions required to be performed is $m * k * d$, while the total number of multiplications required is $m * (k * p + 1) * d$ and the total number of decryptions is d. Essentially, the cost of the encryptions dominate the overall cost.

We ran tests on a SUN Blade 1000 workstation with a 900 Mhz processor and 1 gigabyte of RAM. A C implementation of the Okamoto-Uchiyama [20] encryption system was used. The key size was fixed at 1152 bits, which is more than sufficient for most applications. With this setup, 1000 encryptions require on average around 13s. Also, the time for encryption/decryption cost increases approximately linearly with the number of encryptions. Thus, it is very easy to estimate the actual time required for different number of parties, different vector sizes, and different contingency table sizes. For example, 5 parties with vectors of size 1000 and contingency tables of size 25 would require approximately 28 minutes. The time required would be significantly lower with smaller key sizes and with use of special purpose encryption hardware. Secondly, it is also possible to use a much more efficient size of set intersection protocol[23] to compute the k party scalar product. While this is orders of magnitude more efficient, the downside is the increased disclosure – the size of the intersection sets of all of the subsets is also revealed. If this is acceptable, the more efficient protocol should definitely be used.

3.4 Security Analysis

In the above sections, we gave secure protocols for integrating contingency tables securely in both the horizontal and vertical partitioning case. However, do these protocols really protect each participating party's privacy? In this section, we will discuss the security of the protocols. Further, the discussion considers two factors, the protocols themselves and the specific concerns in the context of contingency tables.

First, consider the multi-party secure sum protocol for horizontal partitioning. This protocol is as secure as figuring out the random data added by the first party. As the random data is generated only by one party, it can be proven that the protocol is secure. However, this conclusion is based on the assumption that no parties collude. If party $i - 1$ and party $i + 1$ collude, they can get the value of party i by subtracting, without knowing the added random data. To against this type of attacks, we may apply the method proposed by [26]. First, divide the sub-contingency table that each party holds into multiple parts. Second, for each party, put all users into a ring randomly and use the same multi-party secure sum protocol to get the sum of parts. Finally, party one sums up all the sums to get the final result. Since at each time, the order of parties is random, it avoids the collusion of some parties at the cost of extra computation. People may argue that if there are only three parties, this method dose not work. Considering the real cases of integrating contingency table, it is fair to assume there are more than 3 parties involved. Thus, multi-party secure sum protocol is secure in terms of the protocol itself.

There are other concerns from the domain of contingency table itself. Suppose one party has data A and the final sum is A also. This means that party can infer that all other parties hold empty tables, without knowing the added random data. In our paper, we assume it is not a threat. This assumption meets our experience in practice. A contingency table is trying to catch the count of every combination of attribute values. The cells of zero do not interest any party. Actually, in the case of sparse contingency table, each party can even assume all the cells are zero. Another possibility is to not remove the final randoms from the sum. Instead the first and the last party (P_1 and P_k) can perform a secure addition and comparison to check if the actual value of the cell is above a certain threshold. If so, the value can be computed. If not, the value is discarded. Now, the parties may simply infer that the actual value is below a threshold without knowing exactly how much it is. However, this could become quite expensive in terms of computation cost.

Now, let us consider the protocol for vertical partitioning of data. It uses the key property of semantic security encryption systems that $E_{pb}(1, r1)$ and $E_{pb}(0, r2)$ are indistinguishable. While this is true, the protocol itself is quite secure. However, the domain of contingency table brings some specific concerns. In our problem, the message that each party sends out is a 0-1 vector. One concern is that if one party has a vector of m cells of 1 and the final multi-vector product is m, this party can infer that the cells of all other parties at the same coordinates are 1. However, that may happens rarely in a large database. Even it happens, that party can at most infer a small part of data. In terms of this, we assume this type of threat is not harmful. As earlier, we can cause the parties to see random splits of the scalar product and use secure comparisons to find out if the actual result should be shared, though this again would be computationally expensive. Another concern is that of collusion among the parties. As discussed earlier, using threshold encryption systems can solve this problem. Also, to overcome this, we may borrow the method for horizontally partitioned data. For each count, put all parties in a ring randomly. Thus, each time the party generating keys is different and the colluded person do not stay together. Another way dealing with potential privacy leak is to generate contingency tables for fewer attributes, like every two or three. Thus, get a set of lower-dimensional tables instead of a whole high-dimensional data. This way will improve the privacy of all parties significantly, while at the cost of loosing lots of information. The generated tables would not help people learn the relationship among all attributes. This is a tradeoff. Its selection may depend on the data self and the real goals for the contingency table. We intend to carefully examine this tradeoff in the future.

4 Conclusion

In this paper, we have presented the problem of secure construction of contingency tables from distributed data and suggested some solutions. Our methods are reasonably efficient and secure. More work is still required to figure out how much information can be inferred from the contingency table itself. It may even be unnecessary to compute the entire contingency table depending on what you would like to learn from it. In the future, we intend to propose secure methods to perform basic analysis on the contingency table such as the chi square test or Fisher's exact test directly without computing the entire table.

References

1. D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 247–255, 2001.

2. R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data*, pages 439–450, 2000.
3. R. Bedi. *Money Laundering - Controls and Prevention*. ISI Publications., 1st edition, 2004.
4. J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In A. Odlyzko, editor, *Advances in Cryptography - CRYPTO86: Proceedings*, volume 263, pages 251–260. Springer-Verlag, Lecture Notes in Computer Science, 1986.
5. M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption that hides all partial information. In R. Blakely, editor, *Advances in Cryptology – Crypto 84 Proceedings*. Springer-Verlag, 1984.
6. L. Cauley. Nsa has massive database of americans' phone calls, May 2006. (USA Today).
7. C. Clifton, M. Kantarcioglu, X. Lin, J. Vaidya, and M. Zhu. Tools for privacy preserving distributed data mining. *SIGKDD Explorations*, 4(2):28–34, Jan. 2003.
8. J. Cornfield. A method of estimating comparative rates from clinical data: Applications to cancer of the lung, breast, and cervix. *Journal of the National Cancer Institute*, 11:1269?1275, 1951.
9. P. Dellaportas and C. Tarantola. Model determination for categorical data with factor level merging. *Journal of the Royal Statistical Society*, 67:269?283, 2005.
10. A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228, 2002.
11. S. E. Fienberg. *The Analysis of Cross-classified Categorical Data*. M.I.T. Press, Cambridge, MA, 2nd edition, 1980.
12. B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen. On Secure Scalar Product Computation for Privacy-Preserving Data Mining. In C. Park and S. Chee, editors, *The 7th Annual International Conference in Information Security and Cryptology (ICISC 2004)*, volume 3506, pages 104–120, December 2–3, 2004.
13. O. Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.
14. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game - a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229, 1987.
15. Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, Baltimore, MD, June 13-16 2005.
16. G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 593–599, New York, NY, USA, 2005. ACM Press.
17. H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, 2003.
18. Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000*, pages 36–54. Springer-Verlag, Aug. 20-24 2000.
19. D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 59–66, San Francisco, California, United States, 1998. ACM Press.
20. T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - Eurocrypt '98, LNCS 1403*, pages 308–318. Springer-Verlag, 1998.
21. P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592*, pages 223–238. Springer-Verlag, 1999.

22. S. J. Rizvi and J. R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 682–693, Hong Kong, Aug. 20-23 2002. VLDB.
23. J. Vaidya and C. Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4), Nov. 2005.
24. J. Vaidya, C. Clifton, and M. Zhu. *Privacy-Preserving Data Mining*. Advances in Information Security. Springer-Verlag, 1st edition, 2005.
25. A. C. Yao. Protocols for secure computation (extended abstract). In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982.
26. H. Yu and J. Vaidya. Secure matrix addition, 2004. (UIOWA Technical Report).