Data Mining and Collusion Resistance

Samuel Shepard[+], Ray Kresman[*], Larry Dunning[*]

Abstract—The field of data mining provides techniques for new knowledge discovery—finding patterns in the data set such as classifiers and association rules that may be useful to the data miner. Privacy preserving data mining seeks to allow users to share data while ensuring individual and corporate privacy concerns are addressed. Recently algorithms have been introduced to maintain privacy even when all but two parties collude. This paper builds upon previous algorithms, putting cycle-partitioned secure sum into the mathematical framework of edge-disjoint Hamiltonian cycles.

I. Introduction

Data is collected every day by a varied and diverse array of individuals—from governments to grocery stores, from game enthusiasts to research scientists. The field of data mining provides techniques for new knowledge discovery—finding patterns in the data set such as classifiers and association rules that may be useful to the data miner. For example, one could mine a set of retail transactions from a store to generate a set of association rules based upon the pattern of the sales. In particular, one association rule might reveal that the purchase of office chairs is often associated with the purchase of fancy staplers.

Association rules are governed chiefly by two statistical parameters: support and confidence. Support is a measure of how frequently a given set of items appears in the database (DB). Confidence, on the other hand, is a conditional probability; that is, given a set of items, what is the likelihood another disjoint set of items will be in the same itemset.

While we are able to discover interesting new association rules by mining a single site, we may discover new rules or learn the true value of our local rules if we can mine our data with a set of peers. Distributed data mining (DDM) offers the data miner a larger data set with the possibility of stronger and, perhaps, novel association rule findings. One good review is given in [6]. It should be apparent to the reader that when working with DDM algorithms efficiency concerns are paramount because the algorithm must scale with the number of different sites or nodes. Communication and coordination of information are only two very important problems to DDM. However, DDM comes at the cost of having to work with a diverse set of one's peers whose honesty, friendliness, and corporate affiliation may not favor the open collaboration of one's private data. While it is useful to mine data with one's peers, it may be disadvantageous to divulge private information to those peers if they are also your competitors. Privacy-preserving data mining (PPDM) seeks to address just such an issue.

Manuscript received March 1, 2009.
Authors address:
(+) University of Toledo, Health Science Campus, 3000 Arlington Avenue, Toledo, Ohio 43614. sammysheep@gmail.com

(*) Department of Computer Science, Bowling Green State University, Bowling Green, OH 43403; kresman@cs.bgsu.edu; dunning@cs.bgsu.edu.

The rest of this paper is organized as follows: Section II provides a motivation for our work by presenting the well-known secure sum problem, and the idea of collusion resistance in commuting such a sum. In section III, we extend secure sum and provide a mathematical framework. Concluding remarks are presented in Section IV.

II. Secure Sum and Collision Resistance

One interesting piece of research suggests relying more on the anonymity of values to achieve a measure of security, that is, if a data miner cannot figure out which value belongs to whom, a level of privacy is therefore afforded when combined with cryptographic techniques [2]. [1] present a number of useful algorithmic primitives for PPDM, including secure set union, secure size of intersection, and secure sum. The goal of SS is simple: let the value of each site's individual input be masked while the global sum of all inputs is universally known. In other words, the individual site's privacy is preserved.

Suppose one wished to compute the global (aggregate) support count for some arbitrary item in the database—as is done in algorithms finding association rules—such that each individual site's support count is revealed to no other parties. Assume also a circuit network topology. First, the initiating site, $S1$, in the network circuit of data miners will calculate a random number R and add it to its local support count, V_1, for the item in question. Next, S_1 will send $R + V_1$ to the next site, S_2, who will add its local support count, V_2, to the sum it received before sending on the final summation $R + V_1 + V_2$ to the next site in the circuit. The next site will repeat this process until the circuit closes. When circuit does close, the initiating node, S_1, will receive the sum of all local supports plus the R value. Since S_1 knows R, the global support can then be easily calculated by subtracting out the R value. Effectively, the R value masks any site from knowing the exact value of the summation of support counts of all the previous sites in the network circuit. In particular, the R value protects S_1 from divulging its support count to $S2$ at the beginning of the circuit. An example of SecureSum is shown in Figure 1 for three participating sites.

SS is important because it allows one to transmit data securely without the high computation cost of encryption methods. However, in its current form, Secure Sum is highly vulnerable to collusion. For example, consider sites S_1, S_2, S_3 as above. To find V_2, observe that $V_2 = (V_2 + V_1 + R) - (V_1 + R)$. Indeed, for any arbitrary S_i, V_i can be found if S_{i-1} and S_{i+1} collude.

Collusion resistance, in terms of SS, means keeping the values generated at any individual node secret, such that if other nodes share information together or collude, the secret values cannot be determined.

Consider a simple analogy for collusion resistance. Bob, Eve, Ray, and Alice each contribute anonymously to their mother's birthday party. Their father thanked them each for the \$35 it cost all of them to put on the party. Eve is jealous of Alice and would like to know if she gave more to the party than her sister, so she convinces Ray to *collude* with her and to tell her how much he gave. Ray agrees. Eve gave \$15 and Ray gave \$10, so Eve knows Bob and Alice gave \$10 = \$35 − (\$15 + \$10). Therefore she correctly concludes she gave more than Alice. However, Eve would also like to rub it in, so she asks Bob to tell her how much he gave too. Bob refuses, leaving Eve without any method to compute Alice's exact contribution to the party. So we learn from this story that the key to collusion resistance involves increasing the number of people (or nodes) required to compute the private data values of the remaining persons or nodes.

Vaidya and Clifton [7] lay the foundation for a cycle-partitioned secure sum algorithm by noting how any node in a secure sum can divide its value into random "shares" (we use the word "partition") such that each share is passed along a different secure sum circuit or cycle. Collusion protection is then achieved when no node has the same neighboring nodes twice for each cycle. This observation is carried out in a collusion resistant PPDM algorithm implicitly based on two edge disjoint Hamiltonian cycles [8]. Here each node divides its count value into two random partitions and performs a SS operation for each partition—making sure its neighbors in the algorithm are all different.

The work from [8] was then extended by Urabe et al [4] to work with D "routes" where each route is equivalent to edge-disjoint Hamiltonian cycles—meaning each count value is divided up into D random partitions at each site and passed along the SS cycle.

Another unique method to providing collusion resistance by creating shares but without the need for edge-disjoint Hamiltonian cycles was given in [5]. This last method also relies on SS and on the need for a fully-connected network but sends data to nodes in a tree-like pattern for each round of the algorithm.

In the next Section, we present our own version of a cycle-partitioned secure sum algorithm, similar to (Urabe et al [4] but provide a mathematical proof for its collusion resistance based on edge-disjoint Hamiltonian cycles.

III. CYCLE-PARTITIONED SECURE SUM

A cycle-partitioned secure sum (CPSS) algorithm is similar to SS in that values are added sequentially along a cycle that starts and ends with an initiating site or node, N_1. At the beginning of the cycle, N_1 adds random number to its count value before sending it to the next node in the cycle. N_1 must also subtract that random number at the end of the cycle to obtain the global sum (the sum of all count values for any particular itemset).

CPSS differs from secure sum in that instead of just using one cycle for summing, the nodes are arranged into C different edge-disjoint Hamiltonian cycles. At each node, the count value is then divided into C random, non-zero, integer partitions–one partition for each edge-disjoint Hamiltonian cycle. Each partition is then sent to the next node in the appropriate cycle.

In the same fashion, the initiating node adds its count value to a random number R and then randomly partitions this value to send along each cycle. When $N1$ receives the subtotals from each circuit or cycle at the end of the computation, the subtotals are added together and the random number R is subtracted out to obtain the global sum. The pseudo code for CPSS is given in Appendix I - Algorithm 1. Appendix I also defines the function *RandomlyPartition* - Algorithm 2, which is needed by Algorithm 1 of Appendix I.

Figure 2 shows an example running of CPSS, Algorithm 1, for $M = 5$ nodes. Notice that for each node in this example there are a total of four adjacent nodes: two receiving and sending.

Suppose we wished to compute the count value (V_2) of node 2 from Figure 2 by *collusion*. In order to compute $V_2 = 7$, observe that it requires the two incoming nodes and the two receiving nodes to collude. That is, it requires N_3 and N_5 on the receiving end and N_1 and N_4 on the sending end respectively: $7 = (5 + 13) - (2 + 9)$. Thus, we observe from this example that 4 nodes were required to compute the remaining node's count value. It is clear then that we need a good definition to describe what it exactly means to be collusion resistant. We adopt the following formal definition to describe collusion resistance for CPSS.

Definition 1. CPSS is K-collusion resistant if no K nodes or fewer working together in a set can compute the count value of any node not belonging to the K-node set.

Using the above definition, we will show that CPSS confers K-collusion resistance proportional to the number of edge-disjoint Hamiltonian cycles in the network.

Theorem 1 [3]. Given any network with C edge-disjoint Hamiltonian cycles and M nodes such that $M > 4$, CPSS is K-collusion resistant with $K = 2 \cdot C - 1$.

Proof of this theorem is given in Appendix II. The proof relies on the following observations: There are C edge-disjoint Hamiltonian cycles. Each node, N_k, partitions its value V_k into C non-zero, integers to send on each of these C cycles. The source node, in addition, splits the random number R, into C partitions, and sends one through each of the C cycles. As seen in Appendix II, since each cycle is edge-disjoint and contains N_1, (see equation (6) of Appendix II) implies that C unique nodes other than N_1 are needed to compute R. This fact is used in computing V_1, the value that node N_1 generated. Full details are given in Appendix II.

IV. CONCLUDING REMARKS

In this paper we described an algorithm for mining data that is resistant to collusion by participating sites. We also provided a mathematical proof of the algorithm. In a forthcoming paper, we will add a feature – anonymous ID – that allows a node to opt out from the computation in an anonymous manner. Opt-out by a node is desirable when its contribution is more sensitive – much larger or much smaller than everyone else; in that case the global sum may be garbled without revealing the identity of the node(s) that opted out.

REFERENCES

[1] Clifton, C., Kantarcioglu, M., Vaidya, J., Lin, X., & Zhu, M. Y. (2002). Tools for privacy preserving distributed data mining. SIGKDD Explor. Newsl., 4(2), 28–34.

[2] Ishai, Y., Kushilevitz, E., Ostrovsky, R., & Sahai, A. (2006). Cryptography from anonymity. In Focs '06: Proceedings of the 47th IEEE symposium on foundations of computer science (focs'06) (pp. 239–248). Washington, DC, USA: IEEE Computer Society.

[3] Shepard, S (2007). Anonymous Opt-Out And Secure Computation In Data Mining. pp. 54. Master's Thesis. Department of Computer Science, Bowling Green State University, OH.

[4] Urabe, S., Wang, J., & Takata, T. (2004, November). A collusion-resistant approach to distributed privacy preserving data mining. In T. Gonzalez (Ed.), Parallel and distributed computing and systems (Vol. 436, p. 626-631). MIT Cambridge, USA: ACTA Press

[5] Urabe, S., Wang, J., Kodama, E., & Takata, T. (2007, February). A high collusion-resistant approach to distributed privacy-preserving data mining. In H. Burkhart (Ed.), Parallel and distributed computing and networks (p. 326-331). Austria: ACTA Press.

[6] Verykios, V. S., Bertino, E., Fovino, I. N., Provenza, L. P., Saygin, Y., & Theodoridis, Y. (2004). State-of-the- art in privacy preserving data mining. SIGMOD Rec., 33(1), 50–57.

[7] Vaidya, J., & Clifton, C. (2004, November/December). Privacy-preserving data mining: Why, how, and when. IEEE Security and Privacy, 2(6), 19–27.

[8] Wang, J., Fukasawa, T., Urabe, S., Takata, T., & Miyazaki, M. (2006). Mining frequent patterns securely in distributed system. IEICE - Trans. Inf. Syst., E89-D(11), 2739–2747.

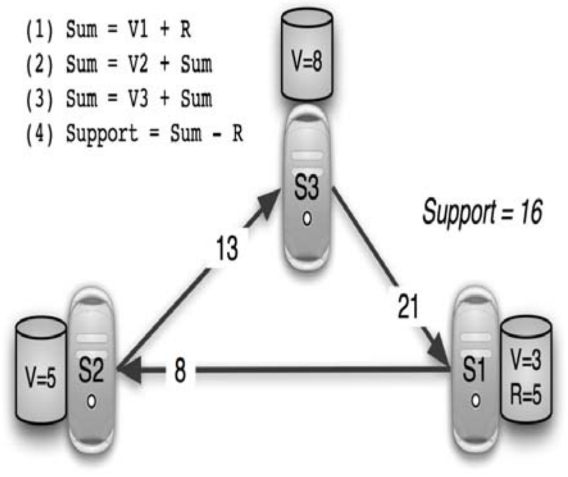

Figure 1: Secure Sum, N = 3

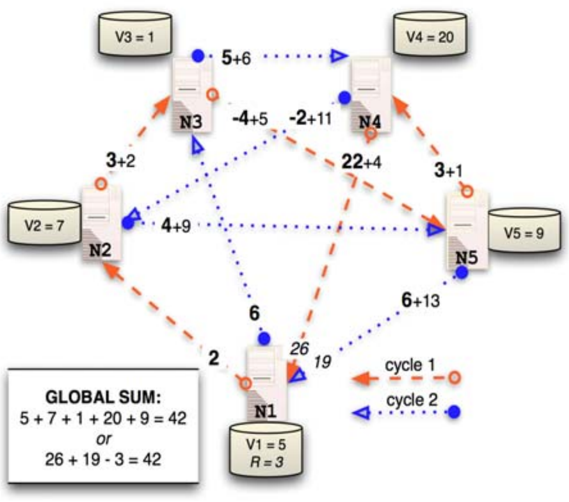

Figure 2: Cycle-partitioned Secure Sum, M = 5

Algorithm 1 CPSS: Cycle-Partitioned Secure Sum

```
Require: Given $M$ nodes and $C$ edge-disjoint Hamiltonian cycles.
Ensure: $\sum_{k=1}^{M} V_k = GlobalSum$
    {For $N_1$, the initiating node.}
 1: $R \Leftarrow NewRandomNumber()$
 2: $RandomlyPartition(V_1 + R, C)$ {$\sum_{i=1}^{C} V_1^{(i)} = V_1 + R,\ 0 \neq V_1^{(i)} \in \mathbb{Z}$}
 3: for $i = 1$ to $C$ do
 4:     Send $V_1^{(i)}$ to the next node in cycle $i$.
 5: end for
    {For each node, partition & add to subtotals in each cycle.}
 6: for $k = 2$ to $M$ do
 7:     $RandomlyPartition(V_k, C)$ {$\sum_{i=1}^{C} V_k^{(i)} = V_k, 0 \neq V_k^{(i)} \in \mathbb{Z}$}
 8:     for $i = 1$ to $C$ do
 9:         $V_k^{(i)} \Leftarrow V_k^{(i)} + V_{received}^{(i)}$
10:         Send $V_k^{(i)}$ to the next node in cycle $i$.
11:     end for
12: end for
    {$N_1$ receives $C$ values.}
13: $GlobalSum = \sum_{i=1}^{C} V_{received}^{(i)} - R$
14: Broadcast $GlobalSum$ to each other node.
```

Algorithm 2 RandomlyPartition(A, C)

```
Require: Given integer value $A$.
Ensure: $\sum_{i=1}^{C} A_i = A$.
1: $Sum \Leftarrow 0$
2: for $i = 1$ to $C - 1$ do
3:     $A_i \Leftarrow NewRandomNumber()$ {$0 \neq NewRandomNumber() \in \mathbb{Z}$}
4:     $Sum \Leftarrow Sum + A_i$
5: end for
6: $A_C \Leftarrow A - Sum$
```

We also define the function $RandomlyPartition$ in Algorithm 2, which is needed by Algorithm 1. Suppose, for example, that we have $C = 5$ cycles and wish to randomly partition the count value of $A = 7$ at some fixed node to send along each cycle. First we generate 4 random numbers: $16, -2, 3, -15$. We also call these random numbers "random partitions." Next, we see that $7-(16+-2+3+-15) = 7-2 = 5$, so the fifth random partition is 5. Finally, we may observe that our five random numbers $(16, -2, 3, -15, 5)$ do in fact partition our count value: $16 + -2 + 3 + -15 + 5 = 7 = A$.

Theorem 1. *Given any network with C edge-disjoint Hamiltonian cycles and M nodes such that $M > 4$, CPSS is K-collusion resistant with $K = 2C - 1$.*

Proof. Given a graph with $M \geq 5$ nodes decomposed into $C \geq 2$ edge-disjoint Hamiltonian cycles, and given any fixed node N_k with value V_k, where $1 \leq k \leq M$, show that no set of $2C - 1$ nodes can compute V_k when N_k is not in the set. In other words, a minimum of $2C$ nodes different from N_k are required to compute V_k.

We denote our edge-disjoint Hamiltonian cycles as $H^{(i)}$ with $1 \leq i \leq C$. For any fixed i, let $(E_p^{(i)})_{p=1}^M$ be the sequence of edges for that cycle, and let $(H_q^{(i)})_{q=1}^M$ be the sequence of nodes in that cycle i. The random number generated by N_1 we designate as R with random partitions $R^{(i)}$ such that $R = \sum_{i=1}^C R^{(i)}$.

Recall that for any fixed k, N_k partitions its value V_k into C non-zero, integers to send on each cycle $H^{(i)}$, such that $\sum_{i=1}^C V_k^{(i)} = V_k$. In order to define the computability of these values, we use the weight function w. The weight function for a given edge is equal to the running total of values on that cycle as computed by the source node of the specified edge:

$$w(E_j^{(i)}) = R^{(i)} + \sum_{p=1}^{j} V_{h_p}^{(i)}, \text{ where each } h_p = H_p^{(i)}. \tag{1}$$

The substitution of $h_p = H_p^{(i)}$ is given for simplicity. Furthermore, notice that $\sum_{p=1}^M V_{h_p}^{(i)} = \sum_{p=1}^M V_p^{(i)}$ for any fixed i and that $w(E_1^{(i)}) = V_1 + R^{(i)}$ for all i. Using (1), we can compute $V_k^{(i)}$ for any fixed k such that $2 \leq k \leq M$ (the computation for V_1 is to follow):

$$V_k^{(i)} = w(E_z^{(i)}) - w(E_{z-1}^{(i)}) \qquad 2 \leq z \leq M \tag{2}$$

$$= R^{(i)} + \sum_{p=1}^{z} V_{h_p}^{(i)} - (R^{(i)} + \sum_{p=1}^{z-1} V_{h_p}^{(i)}) \tag{3}$$

$$= V_k^{(i)} + \sum_{p=1}^{z-1} V_{h_p}^{(i)} - \sum_{p=1}^{z-1} V_{h_p}^{(i)} = V_k^{(i)} \qquad V_k^{(i)} = V_{h_z}^{(i)} \tag{4}$$

In general, for any two consecutive edges in a Hamiltonian cycle, say D_1 and D_2, we have $D_1 = \{a, b\}$ and $D_2 = \{b, c\}$ such that a, b, c are each unique nodes in the cycle. Suppose then for any fixed i and fixed $k \geq 2$ that $N_k = E_k^{(i)} \cap E_{k-1}^{(i)}$ (two consecutive edges in cycle i), then the Hamiltonicity of our cycle implies that there exist *two* unique nodes different from N_k in $E_k^{(i)} \cup E_{k-1}^{(i)}$. These nodes may collude together to compute $V_k^{(i)}$ as shown in (2)-(4). However, to compute V_k for any fixed $k \geq 2$ we need:

$$V_k = \sum_{i=1}^{C} V_k^{(i)} \tag{5}$$

So to compute V_k for any fixed $k \geq 2$ it requires C partitions of V_k, each partition computable by exactly two unique nodes other than N_k. Since our choice of $H^{(i)}$ was arbitrary and any two cycles are *edge-disjoint*, we see that the two nodes computing each $V_k^{(i)}$ are each distinct. To see this is true, pick a node not N_k to be adjacent to N_k in two different cycles, then we have the same edge in two cycles, violating our edge-disjointedness. Hence, for any arbitrary $k \geq 2$, $2C$ unique nodes other than N_k are required to compute V_k. Moreover, since each $V_k^{(i)}$ was non-zero, not less than $2C$ nodes are needed, and therefore, $2C - 1$ nodes, of which N_k is not a part, cannot compute V_k.

Next, for $k = 1$, we have a similar situation, only that the random number R is first added to V_1 before it is partitioned and sent to the next node in each cycle. So to find R, we rely on the fact that each node is sent the $GlobalSum$ at the end of the computation:

$$R = \sum_{i=1}^{C} w(E_M^{(i)}) \ - GlobalSum \tag{6}$$

$$= \sum_{i=1}^{C} (R^{(i)} + \sum_{p=1}^{M} V_{h_p}^{(i)}) \ - GlobalSum \quad \text{(from 1)} \tag{7}$$

$$= \sum_{i=1}^{C} R^{(i)} \ + \sum_{i=1}^{C}\sum_{p=1}^{M} V_p^{(i)} \ - \sum_{k=1}^{M} V_k \tag{8}$$

$$= R + \sum_{p=1}^{M}\sum_{i=1}^{C} V_p^{(i)} \ - \sum_{k=1}^{M} V_k \tag{9}$$

$$= R + \sum_{p=1}^{M} V_p \ - \sum_{k=1}^{M} V_k = R \tag{10}$$

The edge $E_M^{(i)}$, for any fixed i with $1 \le i \le C$, contains node N_1 and another, different node (by the definition of a Hamiltonian cycle). Since each cycle is edge-disjoint and contains N_1, then (6) implies that C unique nodes other than N_1 are needed to compute R. Now we use knowledge of R to compute V_1:

$$V_1 = \sum_{i=1}^{C} w(E_1^{(i)}) \ - R \tag{11}$$

$$= \sum_{i=1}^{C} (R^{(i)} + \sum_{p=1}^{1} V_p^{(i)}) \ - R \tag{12}$$

$$= \sum_{i=1}^{C} R^{(i)} \ + \sum_{i=1}^{C} V_1^{(i)} \ - R \tag{13}$$

$$= R + V_1 - R = V_1 \tag{14}$$

It is clear that $E_1^{(i)}$, for any fixed i with $1 \le i \le C$, contains node N_1 and a different node (by the definition of a Hamiltonian cycle). Because cycles are edge-disjoint and each edge contains N_1, then the nodes not N_1 are each unique from N_1 and each other. From (11) we see that there are C such nodes other than N_1 needed to *finish* the computation of V_1.

To show that the set of C nodes different from N_1 found in (6) and the other set of C nodes different from N_1 found in (11) are all distinct, notice that $N_1 = E_1^{(i)} \cap E_M^{(i)}$ and that the definition of Hamiltonian cycle implies that, for any fixed i with $1 \le i \le C$, there exist two unique nodes not N_1 in $E_1^{(i)} \cup E_M^{(i)}$. Since cycles are edge-disjoint and because the first and last edge of every cycle each contains N_1, every node not N_1 must be unique. Hence a total of $2C$ nodes are needed to compute V_1. Since each $V_1^{(i)}$ is non-zero, not less than $2C$ nodes are required for this computation, that is, $2C - 1$ nodes not N_1 cannot compute V_1.

Thus, we have shown that for a node N_k with value V_k, for *any* fixed k with $1 \le k \le M$, a *minimum* of $2C$ nodes not N_k are required to compute V_k, that is, $2C - 1$ nodes not N_k cannot compute V_k. So by the arbitrariness of k and the definition of K-collusion resistance, CPSS has a collusion resistance of $K = 2C - 1$. $\square$