A Fast Secure Dot Product Protocol with Application to Privacy Preserving Association Rule Mining

Changyu Dong[1] and Liqun Chen[2]

[1] Department of Computer and Information Sciences,University of Strathclyde
changyu.dong@strath.ac.uk
[2] Hewlett-Packard Laboratories
liqun.chen@hp.com

Abstract. Data mining often causes privacy concerns. To ease the concerns, various privacy preserving data mining techniques have been proposed. However, those techniques are often too computationally intensive to be deployed in practice. Efficiency becomes a major challenge in privacy preserving data mining. In this paper we present an efficient secure dot product protocol and show its application in privacy preserving association rule mining, one of the most widely used data mining techniques. The protocol is orders of magnitude faster than previous protocols because it employs mostly cheap cryptographic operations, e.g. hashing and modular multiplication. The performance has been further improved by parallelization. We implemented the protocol and tested the performance. The test result shows that on moderate commodity hardware, the dot product of two vectors of size 1 million can be computed within 1 minute. As a comparison, the currently most widely used protocol needs about 1 hour and 23 minutes.

1 Introduction

Data mining, which generally means the process of discovering interesting knowledge from large amounts of data, has become an indispensable part of scientific research, business analytics, and government decision making. Privacy has always been a concern over data mining. Regulations on data privacy become tighter and tighter. Legislation includes various US privacy laws (HIPAA, COPPA, GLB, FRC, etc.), European Union Data Protection Directive, and more specific national privacy regulations. Since the groundbreaking work [4, 21], there have been a lot of research in privacy preserving data mining (PPDM). However, privacy is not cost free. The overhead of privacy preserving protocols is often too high when a large amount of data needs to be protected. Therefore, how to make protocols secure and also efficient becomes a major challenge to PPDM.

The focus of this paper is association rule mining [2], which discovers association patterns from different sets of data and this is one of the most popular and powerful data mining methods. There have been several privacy preserving solutions for association rule mining. According to [27], those solutions can be divided into two approaches: randomization-based and cryptography-based. In the randomization-based approach [10, 24, 32] , there is a centralized server who aggregates data from many clients and discovers association rules. To protect privacy, data owners (the clients) use some statistical methods to distort the data before sending it to the server. In the distortion procedure, noise is added to the data in order to mask the values of the records while still allow the distribution of the aggregation data to be recovered. The goal is to prevent the server from learning the original data while still be able to discover the rules. However, it has been shown in [20, 17] that in many cases, the original data can be accurately estimated from the randomized data using techniques based on Spectral Filtering, Principal Component Analysis and Bayes estimate. In recent years, the cryptography-based approach becomes very popular. This is because cryptography provides well-defined models for privacy and also a large toolset. In this approach, data is usually split among multiple parties horizontally [19, 25] or vertically [28, 29]. In general, associate rule mining on vertically partitioned data is much more computationally intensive than on horizontally partitioned data because little data summarization can be done locally. Therefore performance is a more acute problem when dealing with vertically partitioned data.

As we will see in section 2, the core of the algorithm for association rule mining on vertically partitioned data is a secure dot product protocol. Therefore the key to make privacy preserving association rule mining practical is to improve the efficiency of the underlying dot product protocol. Many secure dot product protocols have been developed in the past [8, 9, 28, 18, 1, 13, 11, 29, 5, 6, 26]. However, [8, 28, 18] have been shown to be insecure or incorrect, [9] requires an additional third party, and the others require at least $O(n)$ modular exponentiation operations, where n is the size of the input vectors. Modular exponentiation is a costly operation, thus those protocols are very inefficient and cannot be used in real applications that handle large datasets.

The main contribution of this paper is a very efficient secure dot product protocol, which can significantly accelerate privacy preserving association rule mining. Our analysis and implementation show that our dot product protocol is orders of magnitude faster than previous ones. The efficiency comes from the fact that the protocol relies mostly on cheap cryptographic operations, i.e. hashing, modular multiplication and bit operations. The efficiency can be further increased by parallelization. It is worth mentioning that in addition to association rule mining, secure dot product has also been shown to be an important building block in many other PPDM algorithms such as naive Bayes classification [30], finding K-Nearest Neighbors (KNN) [31] and building decision trees [9]. Therefore our protocol can also be used to boost the performance of those PPDM tasks. We also provide a informal security analysis. We have proved the protocol to be secure in the semi-honest model in terms of multiparty secure computation [14]. For space reason, the proof is omitted here and will appear in the full version.

The paper is organized as follows: in Section 2, we briefly review the privacy preserving Apriori algorithm for association rule mining on vertically partitioned data; in Section 3, we describe our secure dot product protocol and the underlying cryptographic building blocks; in Section 4, we report the performance measurements based on the prototype we have implemented, and compare it against the most widely used secure dot product protocol; in section 5, we conclude the paper and discuss possible future work.

Algorithm 1: Privacy Preserving Apriori Algorithm [28]

```
L_1 ={large 1-itemsets};
for $k = 2; L_{k-1} \neq \emptyset; k++$ do
    $C_k$ =apriori-gen($L_{k-1}$);
    for each candidate $c \in C_k$ do
        if all the attributes in c are entirely at $P_1$ or $P_2$ then
            that party independently calculates $c.count$;
        else
            let $P_1$ have attributes 1 to $l$ and $P_2$ have the remaining $m$ attributes
            construct $\boldsymbol{A}$ on $P_1$'s side and $\boldsymbol{B}$ on $P_2$'s side where $\boldsymbol{A} = \prod_{i=1}^{l} \boldsymbol{X}_i$ and $\boldsymbol{B} = \prod_{i=l+1}^{l+m} \boldsymbol{X}_i$;
            compute $c.count = \boldsymbol{A} \cdot \boldsymbol{B}$
        end
        $L_k = L_k \cup c | c.count \geq minsup$;
    end
end
Answer=$\cup_k L_k$;
```

2 Privacy Preserving Association Rule Mining

Association rules show attribute values that appear frequently together in a given dataset. The problem can be formalized as follows [2]: let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items. Let D be a set of transactions and T be a transaction. Each transaction T is a set of items such that $T \subseteq I$. We say T contains X if $X \subseteq T$. An association rule is of the form $X \Rightarrow Y$ where $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. A rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain X also contain Y. A rule $X \Rightarrow Y$ has support s in the transaction set D if $s\%$ transactions in D contain $X \cup Y$. The goal of association rule mining is to find all rules having high support and confidence. One classic algorithm for association rule mining is the Apriori algorithm [3]. The Apriori algorithm can find all frequent itemsets, i.e. itemsets that appear in the transaction set at least $minsup$ times, where $minsup$ is a threashold. After all frequent itemsets have been found, association rules can be generated straightforwardly.

The computation model we consider is two parties each holds part of the transaction set that is vertically partitioned, i.e. one party holds some attributes and the other party holds the rest attributes. In [28], a secure algorithm for finding association rules on vertically partitioned data was proposed. The algorithm (see Algorithm 1) is a straightforward extension of the Apriori algorithm. For each attribute, the party holding it can create a binary vector whose length is the size of the transaction set: the absence or presence of the attribute can be represented as 0 and 1. As shown in [28], we can reduce privacy preserving association rule mining to securely computing the dot products of the binary vectors. Candidate itemsets can be generated as in the Apriori algorithm.In the simplest case where the candidate itemset has only two attributes, for example an attribute X_1 held by P_1 and X_2 held by P_2, the two parties can compute the dot product of the corresponding vectors $\boldsymbol{X}_1 \cdot \boldsymbol{X}_2 = \sum_{i=1}^{n} \boldsymbol{X_1}[i] \cdot \boldsymbol{X_2}[i]$. Then the dot product is the support count, i.e. how many times the itemset appears in the transaction set, and is tested against a predefined threshold $minsup$. If the dot product is greater or equal to the threashold, then the candidate itemset is a frequent itemset. This approach can be easily extended to itemsets that have more attributes. Most of the steps in Algorithm 1 can be done locally. The only step that needs to be computed securely between the two parties is step 10, where a dot product needs to be calculated. Being the only cryptographic step, step 10 is the most time consuming part of the algorithm, and its running time dominates the total running time of the algorithm. Therefore improving the performance of the secure dot product protocol is key to improving the performance of the entire mining algorithm.

3 The Secure Dot Product Protocol

To make privacy preserving association rule mining efficient, we need an efficient secure dot product protocol. In this section, we present such a protocol and analyze its security. The protocol is built on two well-defined cryptographic primitives: the Goldwasser–Micali Encryption and the Oblivious Bloom Intersection.

3.1 Cryptographic Building Blocks

Goldwasser–Micali Encryption The Goldwasser–Micali (GM) encryption is a semantically secure encryption scheme [15]. The algorithms are as follows:

– Key generation algorithm $\mathcal{G}$: it takes a security parameter k, and generates two large prime numbers p and q, computes $n = pq$ and a quadratic non-residue x for which the Jacobi symbol is 1. The public key pk is (x, n), and the secret key sk is (p, q).
– Encryption algorithm $\mathcal{E}$: to encrypt a bit $b \in \{0, 1\}$, it takes b and the public key (x, n) as input, and outputs the ciphertext $c = y^2 x^b \bmod n$, where y is randomly chosen from Z_n^*.
– Decryption algorithm $\mathcal{D}$: it takes a ciphertext c and the private key as input, and outputs the message b: $b = 0$ if the Legendre symbol $(\frac{c}{p}) = 1$, $b = 1$ otherwise.

There are two reasons why we use the GM encryption in our protocol: it is efficient for our purpose and it is homomorphic. The inputs to our protocols are bit vectors and they must be encrypted bit-by-bit in the protocol. Notice that the GM encryption and decryption operations involve only modular multiplications and Legendre symbol computation. Both are very efficient and the computational costs are on the same order of symmetric key operations. This is in contrast to the public key encryption schemes required by the other secure dot product protocols. Those encryption schemes require modular exponentiations, which is usually thousands of time slower. Therefore using the GM encryption makes our protocol much more efficient. The GM encryption is known to be homomorphic and allows computation to be carried out on ciphertexts. More specifically, for two ciphertexts $c_1 = \mathcal{E}(pk, b_1)$ and $c_2 = \mathcal{E}(pk, b_2)$, their product $c_1 c_2$ is a ciphertext of $b_1 \oplus b_2$, where $\oplus$ is the bitwise exclusive or (XOR) operator. This property will be used in our protocol to allow a party to blindly randomize ciphertexts generated by the other party.

Oblivious Bloom Intersection Oblivious Bloom Intersection (OBI) [7] is an efficient and scalable private set intersection protocol. A private set intersection protocol is a protocol between two parties, a server and a client. Each party has a private set as input. The goal of the protocol is that the client learns the intersection of the two input sets, but nothing more about the server's set, and the server learns nothing. In Section 3.3, we will see how to convert the computation of the dot product of two binary vectors into a set intersection problem. Previously, the private set intersection protocols are equally, or even more, costly as secure dot product protocols. Therefore PSI based secure dot product protocols have no advantage in terms of performance. This situation is changed by the recently proposed OBI protocol. The OBI protocol adapts a very different approach for computing set intersections. It mainly bases on efficient hash operations. Therefore it is significantly faster than previous private set intersection protocols. In addition, the protocol can also be parallelized easily, which means performance can be further improved by parallelization. The protocol is secure in the semi-honest model and an enhanced version is secure in the malicious model.

Briefly, the semi-honest OBI protocol works as follows: the client has a set C and the server has a set S. Without loss of generality, we assume the two sets have the same size, i.e. $|C| = |S| = w$. The protocol uses two data structures: the Bloom filter and the garbled Bloom filter. Both data structures can encode sets and allow membership queries. The two parties agree on a security parameter k and chooses k independent uniform hash functions. The hash functions are used here to build and query the filters. The client encodes its set into a Bloom filter, which is a bit vector. The server encodes its set into a garbled Bloom filter, which is a vector of k-bit strings. The size of the Bloom filter (and also the garbled Bloom filter) depends on the security parameter k and set cardinality w. More precisely, the filter size is $k \cdot w \cdot \log_2 e$. The client then runs an oblivious transfer protocol with the server. The oblivious transfer protocol can be implemented efficiently with hash functions. The number of hash operations required by the oblivious transfer protocol is linear in the filter size. The result of the oblivious transfer protocol is that the server learns nothing and the client obtains another garbled Bloom filter that encodes the set intersection $C \cap S$. The client can query all elements in C against this garbled Bloom filter to find the intersection.

We refer the readers to [7] for more details regarding OBI. We will show in Section 3.3 that by combining OBI with the GM encryption, we can get a much more efficient secure dot product protocol.

3.2 Security Model

We study the problem within the general framework of Secure Multiparty Computation. Briefly, there are multiple parties each has a private input, they engage in a distributed computation of a function such that in the end of the computation, no more information is revealed to a participant in the computation than what can be inferred from that participants input and output [14]. Several security models have been defined in this framework. In this paper, we consider the semi-honest model [14]. In this model, adversaries are honest-but-curious, i.e. they will following the protocol specification but try to get more information about honest party's input. Although this is a weak model, it is appropriate in many real world scenarios where the parties are not totally untrusted.

Besides, semi-honest protocols can be upgraded to full security against malicious adversaries using a generic technique [14].

3.3 The Protocol

P_1**'s input**: A binary vector $\boldsymbol{X_1}$ of size n and density at most d.
P_2**'s input**: A binary vector $\boldsymbol{X_2}$ of size n and density at most d.
Other input: A security parameter k, the GM Encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$.
Phase 1: Sets initialization

1.1 P_1 generates a random key pair (pk, sk) for the GM encryption, and sends the public key to P_2.
1.2 P_1 then encrypts $\boldsymbol{X_1}$ bit by bit using the public key, and sends the ciphertext $(\mathcal{E}(pk, \boldsymbol{X_1}[1]), ..., \mathcal{E}(pk, \boldsymbol{X_1}[n]))$ as an ordered list to P_2.
1.3 P_2 generates a vector of n distinct random numbers $\boldsymbol{R}$, then creates an empty set T_2, for $1 \leq j \leq n$, if $X_2[j] = 1$, $T_2 = T_2 \cup \boldsymbol{R}[j]$.
1.4 P_2 generates a set C that has n elements $\{c_1, ..., c_n\}$. For $1 \leq j \leq n$, $c_j = (\boldsymbol{R}[j], \mathcal{E}(pk, X_1[j]) \cdot \mathcal{E}(pk, 0))$.
1.5 P_2 chooses a random permutation θ, and sends $C' = \theta(C)$ to P_1.
1.6 P_1 creates an empty set T_1, then for each $1 \leq i \leq n$ and $c'_i = (u_i, v_i)$ in C', if $\mathcal{D}(sk, v_i) = 1$, then $T_1 = T_1 \cup u_i$.

Phase 2: Output Dot Product

2.1 P_1 uses the set T_1 and P_2 uses the set T_2 as inputs. They engage in an execution of the OBI protocol. P_1 plays the role of the OBI client and gets $T_1 \cap T_2$, P_2 plays the role of the OBI server and gets nothing.
2.2 P_1 counts the element in the intersection and outputs $d = |T_1 \cap T_2|$, which is the dot product $\boldsymbol{X_1} \cdot \boldsymbol{X_2}$.

Fig. 1: The Secure Dot Product Protocol

A method [16] that converts the binary vector dot product problem into computing the cardinality of set intersection is as follows: given two binary vectors $\boldsymbol{X_1}$ and $\boldsymbol{X_2}$ both of cardinality n, we can construct two sets S_i $(i \in \{1, 2\})$ such that j is an element of S_i if the $\boldsymbol{X_i}[j] = 1$, i.e. $S_i = \{j \mid \boldsymbol{X_i}[j] = 1\}$. Then the dot product $\boldsymbol{X_1} \cdot \boldsymbol{X_2} = |S_1 \cap S_2|$. In the light of this, in [16] the authors proposed an approximate secure dot product protocol that works by estimating the cardinality of the set intersection.

Our protocol is also based on this set intersection cardinality idea, but can compute the exact intersection cardinality rather than just an approximation. The protocol is shown in Fig. 1. We now explain the rationale behind the design and discuss the security informally. Our insight is that we can build a secure binary vector dot product protocol on top of a private set intersection protocol. It seems trivial: let the two parties convert their private vectors into sets as described above, then run a private set intersection protocol between them, the protocol outputs the set intersection $S_1 \cap S_2$ thus $|S_1 \cap S_2|$ can be obtained as a by-product. This solution, however is flawed. The main problem is that it leaks more information than desired. In fact, the party who obtains the intersection as the output of the private set intersection protocol learns not only the dot product, but also some bits in the other party's vector: for any j in the intersection, the jth-bit of the other party's vector must be 1. To prevent this, in our protocol, we do not use the sets S_1 and S_2 directly, but map them into T_1 and T_2 such that the following two properties hold: (1) $|T_1 \cap T_2| = |S_1 \cap S_2|$, (2) at the end of the protocol P_1 obtains $T_1 \cap T_2$, from which it can obtain $|S_1 \cap S_2|$, but no other information.

The first property ensures the correctness of our protocol. To see why this property holds, observe that in step 1.3, P_2 maps each $1 \leq j \leq n$ to a random number $\boldsymbol{R}[j]$. The mapping is used by both parties to map S_i to T_i: for each $j \in S_i$, or equivalently for each j such that $\boldsymbol{X_i}[j] = 1$, its counterpart $\boldsymbol{R}[j]$ is in T_i. Thus $|T_1 \cap T_2| = |S_1 \cap S_2|$ is guaranteed. The difficulty here is how to let P_1 correctly generate T_1. For P_2, it can easily generate T_2 because it knows the mapping. However, to maintain privacy, P_1 is not allowed to know the mapping. In order to allow P_1 to generate T_1, P_2 creates a labelled permutation of the vector $\boldsymbol{R}$ (step 1.4, 1.5). For each $\boldsymbol{R}[j]$, the label assigned to it is a ciphertext of $\boldsymbol{X}_1[j]$. In step 1.6, P_1 can decrypt the label, if it encrypts 1, then P_1 knows the corresponding vector element should be put into T_1.

The second property ensures the privacy of our protocol. It is easy to see why P_2 gets no more information: the only message P_2 gets in phase 1 is the encrypted version of $\boldsymbol{X_1}$, then by the security of the GM encryption, P_2 gets no information about $\boldsymbol{X_1}$. Phase 2 is essentially a PSI execution and P_2 plays the role of the PSI server, thus by the security of the PSI protocol, it learns nothing. For P_1, it receives C' in step 1.6 and each c'_i is a tuple $(u_i, v_i) = (\boldsymbol{R}[j], \mathcal{E}(pk, X_1[j]) \cdot \mathcal{E}(pk, 0))$. C' allows P_1 to generate T_1 but no more than that. Since $C' = \theta(C)$ and θ is a random permutation, the order of an element in the set i does not leak any information. Also $u_i = \boldsymbol{R}[j]$ for some j is a random number generated independently of j, and the mapping from j to $\boldsymbol{R}[j]$ is known only by P_2, thus u_i leaks no information. On the other hand $v_i = \mathcal{E}(pk, X_1[j]) \cdot \mathcal{E}(pk, 0)$, so v_i incorporates randomness generated by P_2, therefore P_1 cannot link it back to the ciphertext $\mathcal{E}(pk, X_1[j])$ generated by itself. $\mathcal{D}(sk, v_i)$ gives only 1 bit information, i.e. there exists a j such that $\boldsymbol{X_i}[j] = 1$. That is exactly what P_1 should know. At the end of phase 1, P_1 holds a set T_1 which it knows is a equivalent of S_1 but cannot link the elements in T_1 back to elements in S_1: any element in T_1 can be any element in S_1 with equal likelihood. A consequence is that the output in step 2.1 $T_1 \cap T_2$ gives P_1 no more information about $S_1 \cap S_2$ other than the cardinality of the intersection.

The protocol in Fig. 1 has only one output: P_1 gets the dot product and P_2 gets nothing. In some applications, it is preferable to have two outputs such that P_1 gets a number a and P_2 gets a number b and $a + b$ is the dot product. This can be done by executing the protocol in Fig. 1 twice and let the parties switch roles in the two executions. Suppose Alice has a vector $\boldsymbol{Y_1}$ and Bob has a vector $\boldsymbol{Y_2}$, both are of size n. Bob first splits $\boldsymbol{Y_2}$ randomly into two n-bit vectors $\boldsymbol{Y_3}$ and $\boldsymbol{Y_4}$, such that $\boldsymbol{Y_2} = \boldsymbol{Y_3} + \boldsymbol{Y_4}$. In the first round, Alice plays the role of P_1 and uses $\boldsymbol{Y_1}$ as her input. Bob plays the role of P_2 and uses $\boldsymbol{Y_3}$ as input. Alice gets an output a after the execution. Then in the second round, Alice plays the role of P_2 and still uses $\boldsymbol{Y_1}$ as her input. Bob plays the role of P_1 and uses $\boldsymbol{Y_4}$ as his input. Then Bob receives an output b. The outputs satisfy that $a + b = \boldsymbol{Y_1} \cdot \boldsymbol{Y_2}$. To see that, observe that $a = \boldsymbol{Y_1} \cdot \boldsymbol{Y_3}$ and $b = \boldsymbol{Y_1} \cdot \boldsymbol{Y_4}$, so $a + b = \boldsymbol{Y_1} \cdot (\boldsymbol{Y_3} + \boldsymbol{Y_4}) = \boldsymbol{Y_1} \cdot \boldsymbol{Y_2}$.

3.4 Efficiency

From the description of the protocol in Fig. 1 we can see that the computational and communicational complexities of phase 1 are both $O(n)$, where n is the size of the vector. Phase 2 is a single execution of the OBI protocol. Therefore the computational and communicational complexities of this phase are also $O(n)$.

More specifically: in phase 1 the total computational cost is $3n$ modular multiplications plus computing n Legendre symbols, in phase 2 the total computational cost is $2d \cdot k \cdot n(1 + \log_2 e)$ hash operations, where d is the density of the vectors, k is the security parameter (e.g. 80 or 128) and e is the natural logarithm constant. As a comparison, the protocol in [13], which is based on Paillier homomorphic encryption, requires n modular exponentiations and n modular multiplications. At first glance it seems that our protocol is not as efficient as the protocol in [13]. However a closer analysis shows the opposite. This is because modular exponentiation, required by [13], is much more expensive than the operations requires by our protocol. To better illustrate the difference, we show the running time of the cryptographic operations at 80-bit security in Table 1. The modulus used is 1024-bit. The time was measured on a Mac Pro with 2.4 GHz Intel Xeon CPUs. For the cheap operations, the time shown is the average of 1,000,000 executions and for modular exponentiation, the time is the average of 1,000 executions. We can see the difference is 3 orders of magnitude. At higher security levels, the difference is even bigger.

sha-1 hash	mod mul.	Legendre symbol	mod exp.
0.2×10^{-6}	0.8×10^{-6}	5.4×10^{-6}	3.7×10^{-3}

Table 1: Average running time of cryptographic operations at 80-bit security (in seconds)

Communication wise, our protocol in phase 1 transfers $2n$ element in group Z_N^* and in phase 2 transfers $2 \cdot d \cdot k \cdot n \cdot \log_2 e$ k-bit strings. On the other hand, the communication cost of the protocol in [13] is n element in group $Z_{N^2}^*$. Our protocol consumes more bandwidth than the protocol in [13]. The bandwidth consumption of our protocol depends on d, the density of the vector. In the worst case where $d = 1$, the bandwidth consumed by our protocol is about 10 times as much as the protocol in [13], but when $d = 0.1$, the bandwidth consumption of our protocol is only twice that of the protocol in [13]. In real applications the density is often small thus the difference is not significant.

4 Implementation and Performance

We implemented our protocol in Figure 1 and measured the performance. The protocol was implemented in C. The implementation uses OpenSSL [22] and GMP [12] for the cryptographic operations. We used the Oblivious Bloom Intersection as the underlying PSI protocol. We obtained the source code of the OBI protocol, which is also in C, and incorporated it in our implementation. As a reference, we also implemented Goethals et al's protocol [13] in C. Goethals et al's protocol relies on additive homomorphic encryption and we use the Paillier public key scheme [23] as the building block of the protocol. We tested the protocols with 80 bits symmetric keys and1024 bits public keys. The experiments were conducted on two Mac computers. P_1 ran on a Macbook Pro laptop with an Intel 2720QM quad-core 2.2 GHz CPU and 16 GB RAM. P_2 ran on a Mac Pro with 2 Intel E5645 6-core 2.4GHz CPUs and 32 GB RAM. The two computers were connected by 1000M Ethernet. The two parties communicate through TCP socket. In all experiments, we use randomly generated bit vectors as inputs. We use synthesized data rather than real data because of the following two reasons: firstly the performance is not affected by the nature of the input data; secondly the performance of our protocol varies with the density of the vectors and it is hard to demonstrate the worst case performance with real data.

We first show the overall running time of the protocols with different vector sizes. The performance of our protocol depends on density of the vectors. In this experiment we measured the worst case performance by setting the density of the vectors used in our protocol to 1. In the experiment, each party uses one thread for computation and another one for network communication. The result is shown in table 2. As we can see in the table, our protocol is more than 20 times faster than Goethals et al's protocol.

Vector size	1,000	10,000	100,000	1,000,000
Our Protocol	0.27	2.25	22.01	238.37
Goethals et al	5.06	49.58	509	5039

Table 2: Total Running Time Comparison (in Seconds)

In real world applications, the density of the vectors is less than 1 and our protocol can be more efficient. The performance of our protocol has been further improved by exploiting parallelization. Oblivious Bloom Intersection, the underlying PSI protocol, is highly parallel. The implementation of the Oblivious Bloom Intersection protocol has a parallel mode which allows the program to utilize all available CPU cores and distribute the computation among them. The total running time can be significantly shortened if the program is running on multi-core systems. In the next experiment, we measured the performance of our protocol running in non-parallel and parallel modes with different densities. In the experiment, we set the vector size $n = 1,000,000$ and varied the density from 0.1 to 1. The result is shown in Figure 2. As we can see in the figure, the parallel mode does increase the performance significantly. The total running time in the non-parallel mode is $1.8\times$ – $4.2\times$ of that in the parallel mode. On the other hand, the total running time in each mode increases linearly with the vector density. When the density is 0.1, the total running time is 18.9 and 34.1 seconds in the parallel and non-parallel modes respectively, while when the density is 1 the numbers become 57 and 238 seconds. To compare, we also plot the total running time of Goethals et al's protocol when the vector size is set to $5,000$ and $50,000$. The running time of Goethals et al's protocol is not affected by the vector density. As we can see, the total running time of our protocol in all cases is less than that of the Goethals et al's protocol with $n = 50,000$. The total running time of Goethals et al's protocol is linear in n. That means our protocol is at least 20 times faster. Our protocol is more than100 times faster than Goethals et al's when running in the non-parallel mode and when the vector density is 0.1, and is 200 times faster when running in the parallel mode and with a vector density below 0.2.

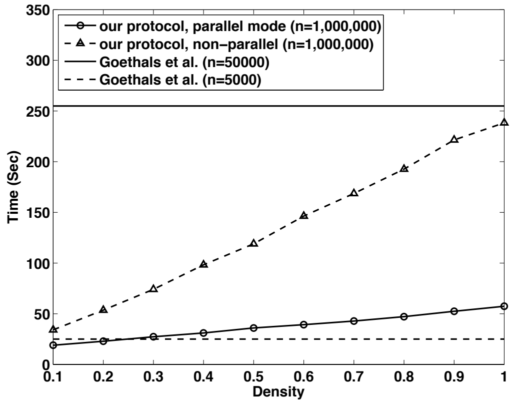

Fig. 2: Running Time in Non-Parallel and Parallel Modes with Various Vector Density

We also measured the bandwidth consumption. In the experiment, we set the vector size $n = 1,000,000$ and varied the density. The bandwidth consumption of Goethals et al's protocol with $n = 1,000,000$ was measured for comparison. The result is shown in Figure 3. Goethals et al's protocol consumes about 266 MB bandwidth. The bandwidth consumption of our protocol is about $1.9\times$ and $9.5\times$ of that when the density is 0.1 and 1 respectively. The number is quite close to our estimation in section 3.4. If the network connection is very slow, then Goethals et al's protocol can be faster than ours because the bottleneck is the network speed. Given the above measurements, we can roughly estimate when to switch. For example, if the vector size is 1,000,000 and density is 1, then when the network speed is less than 3.8 Mbps, Goethals et al's protocol should be used; if the density becomes 0.1, then Goethals et al's protocol becomes faster only when the network speed is less than 0.4 Mbps.

5 Conclusion

In this paper we investigated how to accelerate association rule mining on big datasets in a privacy preserving way. To this end, we developed a provably secure and very efficient dot product protocol. The protocol is based on the Goldwasser–Micali Encryption and Oblivious Bloom Intersection. The security of the protocol can be proved in the semi-honest model. By avoiding expensive cryptographic operations such as modular exponentiation, the performance of our protocol is much better than previous ones. We

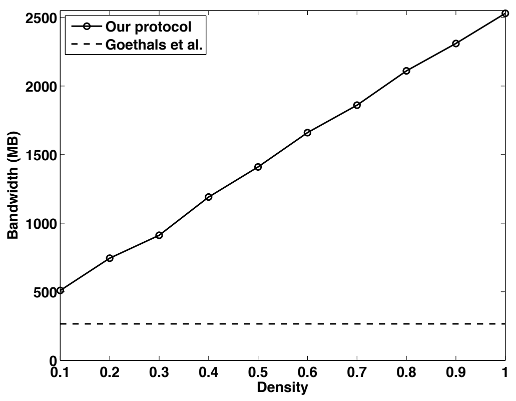

Fig. 3: Bandwidth Consumption

implemented the protocol and compared the performance against the currently most widely used secure dot product protocol. The results show that our protocol is orders of magnitude faster.

As future work, we would like to extend the protocol to multiple parties. We would also like to investigate how to improve the efficiency of other PPDM tasks. As we mentioned, our protocol can be used as a sub-protocol in many other PPDM tasks. We need also efficient constructions for other building blocks in the PPDM tasks. Another future direction is to implement the protocol in frameworks such as MapReduce, so that we can take advantage of the processing power provided by large scale distributed parallel computing, e.g. cloud computing.

Acknowledgements. We would like to thank the anonymous reviewers. Changyu Dong is supported by a Science Faculty Starter Grant from the University of Strathclyde.

References

1. Agrawal, R., Evfimievski, A.V., Srikant, R.: Information sharing across private databases. In: SIGMOD Conference. (2003) 86–97
2. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD Conference. (1993) 207–216
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB. (1994) 487–499
4. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: SIGMOD Conference. (2000) 439–450
5. Amirbekyan, A., Estivill-Castro, V.: A new efficient privacy-preserving scalar product protocol. In: AusDM. (2007) 209–214
6. Cristofaro, E.D., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: CANS. (2012) 218–231
7. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: An efficient and scalable protocol. In: ACM Conference on Computer and Communications Security. (2013)

8. Du, W., Atallah, M.J.: Privacy-preserving cooperative statistical analysis. In: ACSAC. (2001) 102–110
9. Du, W., Zhan, Z.: Building decision tree classifier on private data. In: Proceedings of the IEEE international conference on Privacy, security and data mining - Volume 14. CRPIT '14, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2002) 1–8
10. Evfimievski, A.V., Srikant, R., Agrawal, R., Gehrke, J.: Privacy preserving mining of association rules. In: KDD. (2002) 217–228
11. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: EUROCRYPT. (2004) 1–19
12. GMP. http://gmplib.org/
13. Goethals, B., Laur, S., Lipmaa, H., Mielikäinen, T.: On private scalar product computation for privacy-preserving data mining. In: ICISC. (2004) 104–120
14. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
15. Goldwasser, S., Micali, S.: Probabilistic encryption and how to play mental poker keeping secret all partial information. In: STOC. (1982) 365–377
16. He, X., Vaidya, J., Shafiq, B., Adam, N.R., Terzi, E., Grandison, T.: Efficient privacy-preserving link discovery. In: PAKDD. (2009) 16–27
17. Huang, Z., Du, W., Chen, B.: Deriving private information from randomized data. In: SIGMOD Conference. (2005) 37–48
18. Ioannidis, I., Grama, A., Atallah, M.J.: A secure protocol for computing dot-products in clustered and distributed environments. In: ICPP. (2002) 379–384
19. Kantarcioglu, M., Clifton, C.: Privacy-preserving distributed mining of association rules on horizontally partitioned data. In: DMKD. (2002)
20. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the privacy preserving properties of random data perturbation techniques. In: ICDM. (2003) 99–106
21. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: CRYPTO. (2000) 36–54
22. OpenSSL. http://www.openssl.org/
23. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT. (1999) 223–238
24. Rizvi, S., Haritsa, J.R.: Maintaining data privacy in association rule mining. In: VLDB. (2002) 682–693
25. Tassa, T.: Secure mining of association rules in horizontally distributed databases. IEEE Transactions on Knowledge and Data Engineering **99**(PrePrints) (2013) 1
26. Tassa, T., Jarrous, A., Ben-Ya'akov, Y.: Oblivious evaluation of multivariate polynomials. J. Mathematical Cryptology **7**(1) (2013) 1–29
27. Vaidya, J., Clifton, C., Zhu, Y.: Privacy Preserving Data Mining. Advances in Information Security. Springer (2006)
28. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: KDD. (2002) 639–644
29. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security **13**(4) (2005) 593–622
30. Vaidya, J., Kantarcioglu, M., Clifton, C.: Privacy-preserving naïve bayes classification. VLDB J. **17**(4) (2008) 879–898
31. Wong, W.K., Cheung, D.W.L., Kao, B., Mamoulis, N.: Secure knn computation on encrypted databases. In: SIGMOD Conference. (2009) 139–152
32. Zhang, N., Wang, S., Zhao, W.: A new scheme on privacy preserving association rule mining. In: PKDD. (2004) 484–495