Experimental Analysis of a Privacy-Preserving Scalar Product Protocol*

Zhiqiang Yang Rebecca N. Wright
Computer Science Department
Stevens Institute of Technology
Hoboken, NJ 07030 USA
zyang|rwright@cs.stevens.edu

Hiranmayee Subramaniam
Stevens Institute of Technology graduate
hiran@polypaths.com

Abstract

The recent investigation of privacy-preserving data mining has been motivated by the growing concern about the privacy of individuals when their data is stored, aggregated, and mined for information. In an effort towards practical algorithms for privacy-preserving data mining solutions, we analyze and implement solutions to an important primitive: the privacy-preserving scalar product of two vectors held by different parties. Privacy-preserving scalar products are an important component of privacy-preserving data mining algorithms, particularly when data is vertically partitioned between two or more parties.

We examine a cryptographically secure privacy-preserving data mining solution in different computational settings. Our experimental results show that in the absence of special-purpose hardware accelerators or practical optimizations, the computational complexity, rather than the communication complexity, is the performance bottleneck. We also evaluate several practical optimizations to improve the efficiency.

Keywords: privacy, data mining, scalar product protocol.

1 Introduction

Privacy-preserving data mining is intended to address conflicting goals. On the one hand, it is often desirable to extract information from collected data. On the other hand, there are often legitimate concerns about the privacy of personal data, proprietary data, and other sensitive information. Privacy-preserving data mining, in which certain computations are allowed, while other information is to remain protected, was first introduced in 2000 by Agrawal and Srikant [AS00] and Lindell and Pinkas [LP02]. Since then, extensive research has been devoted to privacy-preserving data mining and other privacy-preserving primitives efficient enough to be used on extremely large data sets (e.g., [AD01, FIM+01, CIK+01, ESAG02, VC02, KC02, EGS03, VC03, FNP04, AMP04, WY04]).

In general, this research has been divided into solutions that provide strong cryptographic privacy protection, which require more computational overhead and have so far been limited to extremely simple (but useful) functions, and those that use perturbation, which provide weaker privacy properties, but allow much more efficient solutions and allow computation of more sophisticated data mining functions.

Our work provides an experimental evaluation of a cryptographic solution to the problem of computing the scalar product of two vectors. The particular solution we use for our experiments appears in [WY04] and is proven secure in [GLLM04]. The scalar product primitive is extremely useful in privacy-preserving data mining both when data is vertically partitioned and when it is arbitrarily partitioned between two or more parties. In these cases, such vectors are typically binary vectors representing which of the records that each party are compatible with a particular set of assignments to all the attributes. (Hence, even if the underlying data

*This work was supported by the National Science Foundation under Grant No. CCR-0331584. A preliminary version of this work appears as [SWY04].

Appears in *International Journal of Computer Systems Science and Engineering*, Vol. 21, No. 1, 2006, pp. 47–52.

itself is non-binary, it is binary scalar products that are typically required). Examples include association rules [VC02], naive Bayes classifiers [VC04], Bayesian networks [WY04, MSK04, YW05], and k-means clustering [JW05].

Our results show that the total running time needed is quite high, but it becomes feasible if certain straightforward optimizations are done, such as some precomputation before the actual computation is to be done. Unless special hardware accelerators or practical optimizations are used, the computational delay caused by the encryption operations is the bottleneck, while the communication delay is significantly less.

To our knowledge, our implementation is one of the first implementations of cryptographically secure privacy-preserving database computations. Relatedly, Malkhi et al.'s recent implementation [MNPS04] of Yao's general secure two-party computation solution [Yao86] provides the first general secure multiparty computation results, and demonstrates that many computations on relatively small data sets can be done extremely efficiently. Indeed, secure multiparty computation and cryptographically strong privacy-preserving database computations, largely considered only theoretical, seem to be on the cusp of practicality as both theoretical and technological advances have improved their performance. Therefore, this kind of initial experimental work is an important contribution to understanding where such results are within the realm of practice and where further improvements are still needed.

Further details of the scalar product protocol we use are given in Section 2. We report on our experimental results in Section 3.

2 Privacy-Preserving Scalar Products

As previously discussed, the scalar product, or inner product, of two binary vectors is a frequently used computation in privacy-preserving data mining applications. Given two vectors $\mathbf{z} = (x_1, \ldots, x_n)$ and $\mathbf{z}' = (y_1, \ldots, y_n)$ of the same length, their scalar product is $\mathbf{z} \cdot \mathbf{z}' = \sum_{i=1}^{n} x_i y_i$.

Distributed privacy-preserving scalar product protocols are an important primitive for privacy-preserving data mining. Several such protocols have been proposed (e.g. [CIK+01, DA01, VC02, LKR03, WY04, FNP04, GLLM04, MSK04]), with varying degrees of security. A nice overview of the problem and the security properties of some solutions appears in [GLLM04]. In a privacy-preserving scalar product protocol, one party Alice holds $\mathbf{z}$ and the other party Bob holds $\mathbf{z}'$. One or both parties are supposed to learn $\mathbf{z} \cdot \mathbf{z}'$. Ideally, neither party should learn anything about the other party's input beyond what is implied by his or her own vector and his or her result (where his or her result is either the scalar product or nothing, depending on whether the party was supposed to learn it), though some of the proposed protocols provide only weaker notions of privacy.

To be useful as a privacy-preserving primitive that can be used as a sub-protocol in a larger privacy-preserving protocol, it is often desirable to use a privacy-preserving scalar product protocols in which Alice and Bob learn *additive secret shares* of the resulting scalar product, rather than either party learning the scalar product itself. For example, if Alice holds $\mathbf{z}$, Bob holds $\mathbf{z}'$, and the scalar product $\mathbf{z} \cdot \mathbf{z}'$ is known to be less than M, then Alice learns r_A and Bob learns r_B, where r_A and r_B are random integers, called *shares*, between 0 and $M-1$ such that $r_A + r_B \bmod M = \mathbf{z} \cdot \mathbf{z}'$. Therefore, together

Setting: Alice has a binary vector $\mathbf{z}_a = (a_1, \cdots, a_n)$ and Bob has a binary vector $\mathbf{z}_b = (b_1, \cdots, b_n)$.

Goal: Bob learns $\mathbf{z}_a \cdot \mathbf{z}_b + R$ and Alice learns R, where R is a random number chosen by Alice.

1. Bob generates a cryptographic key pair (PK, SK) of a semantically secure homomorphic encryption scheme and sends the public key PK to Alice. We denote encryption using PK by $e(\cdot)$ and decryption using SK by $d(\cdot)$.
2. Bob encrypts his elements using PK and sends the vector $(e(b_1), \cdots, e(b_n))$ of encryptions to Alice.
3. Alice generates a random number R and encrypts it using PK.
4. Alice computes $P = e(R) \cdot \prod_{i=1}^{n} y_i$, where $y_i = e(b_i)$ if $a_i = 1$ and $y_i = 1$ if $a_i = 0$. Alice sends P to Bob.
5. Bob decrypts P to get $d(P) = R + \sum_{i=1}^{n} a_i \cdot b_i$.

Figure 1. Privacy-Preserving Scalar Product Protocol

Alice and Bob "know" $\mathbf{z}\cdot\mathbf{z}'$, but individually neither learns any information about its value. We call such a protocol a *privacy-preserving scalar product share protocol*. Most of the above protocols can be modified to act as privacy-preserving scalar product share protocols with no performance penalty.

The privacy-preserving scalar product share protocol that we use for our experiments is a relatively straightforward one based on *semantically secure homomorphic encryption*. The protocol was presented in [WY04] and—in the version that computes the product itself rather than the shares—proven secure in [GLLM04]. The protocol as we implement it is presented in Figure 1.

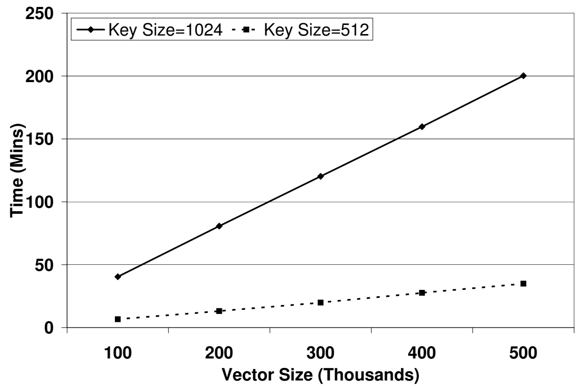

Figure 2. Overall Running Time of the Protocol

3 Experimental Results

We implemented the scalar product protocol shown in Figure 1 and measured the computation and communication performance. We implemented the protocol in Java and C. The Java version uses the Java security package to perform cryptographic operations and the C implementation uses the OpenSSL libraries. For the semantically secure homomorphic encryption, we use the Paillier public key scheme [Pai99] as the building block of the protocol. We tested the protocol performance with cryptographic keys of 512 bits and 1024 bits, respectively. We experimented across various vector sizes from $10,000$ elements to $500,000$ elements, where each element was either 0 or 1. On average, the performance results from our Java experiments were around five times slower than those of similar C experiments; except in Section 3.4, we report only the C numbers here.

The experimental data was measured in different computational settings. Our results show that computation time prevails over the communication time, accounting for the bulk of the total running time.

3.1 Performance Results without Any Optimization

Figures 2–5 show experimental results of the direct implementation of the solution described in Section 2, without any optimizations. The experimental environment is two NetBSD systems running on AMD Athlon 2GHz processors with 512M memory. Alice's process ran on one of the computers in our experiment, and Bob's on the other. They were connected by an Ethernet.

Figure 2 shows the overall running time of the protocol with different key sizes of 512 and 1024 bits respectively. 1024-bit cryptographic key provide much stronger security than 512 bits, but the security comes at a price: the protocol with 512-bit key is five times faster than the one with 1024-bit key. For vectors of $200,000$ elements, the protocol with 512-bit key takes 17.4 minutes, and the protocol with 1024-bit key takes 80.7 minutes.

Our results illustrate linear time performance, as expected. The bulk of the execution time is attributable to Alice's computation of the n public key encryptions of her input vector. Figure 3 shows that Alice's computation dominates, taking 98% of the overall running time of the protocol. The time for Bob's computation is significantly less, and the communication time of the protocol is also much less.

Figures 4 and 5 represent the communication time and Bob's computational time in the protocol with various vector sizes. Our results show that in the absence of any practical optimizations or specialized hardware to accelerate Alice's encryption, computation time is the bottleneck for the protocol's performance. In Sections 3.2–3.4, we evaluate several straightforward practical optimizations.

3.2 Streaming Execution and Pipeline parallelism

Noting that both Alice's computation and Bob's computation can be done in a single "streaming" pass through their inputs, we implemented "batching" of the client processing, in which Alice batches her processing of indices into smaller sized chunks, performing and sending the encryptions of the indices in each chunk before proceeding to the next

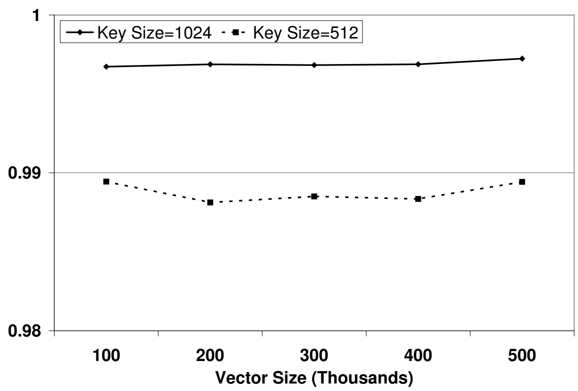

Figure 3. **Ratio of Alice's Computational Time to Overall Running Time**

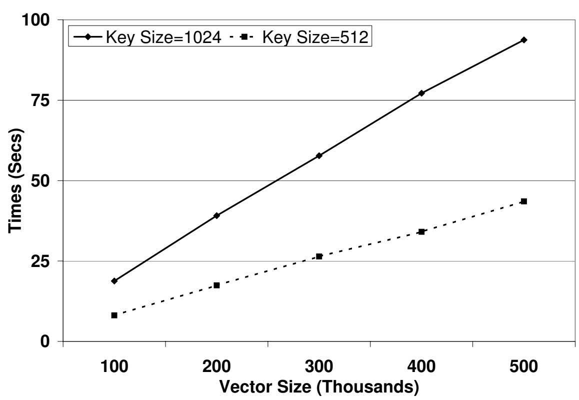

Figure 4. **Bob's Computational Time**

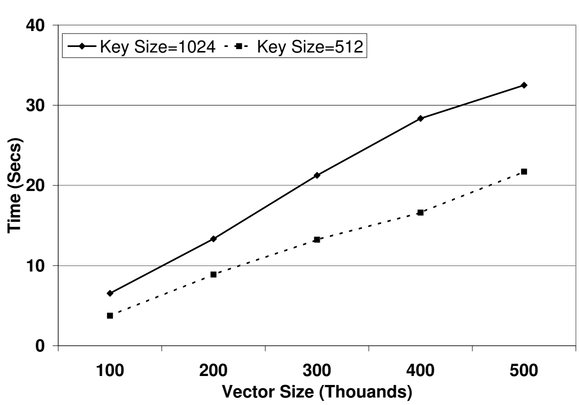

Figure 5. **Communication Time**

chunk. Upon receiving each chunk, Bob can continue computing the partial product.

In addition to taking advantage of pipeline parallelism, this approach also reduces the memory requirements of both Alice and Bob. At any point in time, Alice has to allocate memory needed to hold only one chunk of her vector rather than the whole vector. Similarly, Bob needs only hold a single vector chunk in memory at one time. The optimal chunk size depends on the relative communication and computation speeds, as well as the overhead in processing messages and memory access. In order to achieve maximum parallelization, ideally all three activities (communication of one batch, client processing of the next batch, and server processing of the previous batch) should require approximately the same amount of time.

In the computational setting of Section 3.1, Bob's computation and the communication account for only 2% overhead of the overall overhead, so pipeline parallelism, at best, slightly improves the performance. However, if Bob is running on a slow computer or the protocol is running over a slow communication channel, the overall running time can be more substantially reduced via pipeline parallelism.

We ran the protocol with a 512-bit key to test the effect of pipeline parallelism. Figure 6 compares the overall runtime of the protocol with and without batching data. In this experiment, which ran on two computers with 1GHz CPU and they are slower than the setting of Section 3.1, we took a batch size of 100 elements, resulting in approximately a 10% reduction in overall runtime.

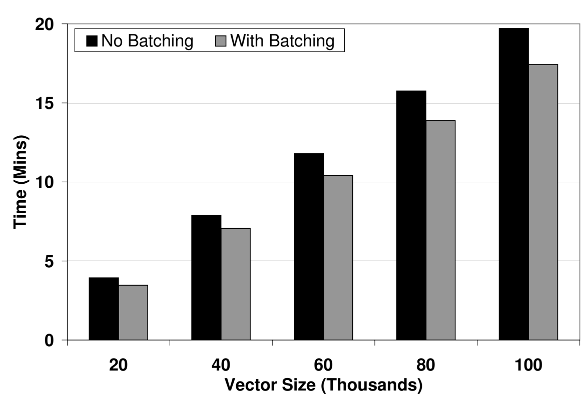

Figure 6. **The Comparison of Overall Running Time with and without Batching**

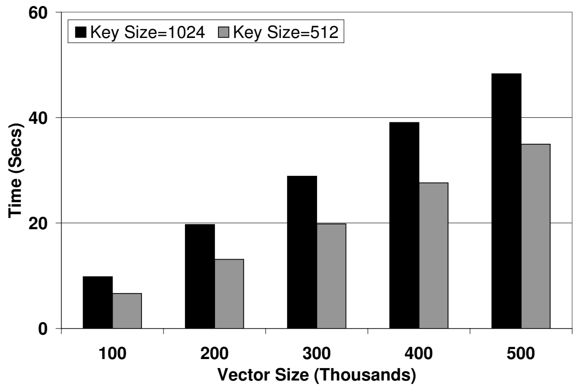

Figure 7. **The Overall Running Time with Preprocessing**

3.3 Preprocessing the vectors

This optimization aims to reduce the overall computation complexity by having Alice encrypt her vector offline in advance and store the encrypted data. Even if Alice does not know in advance which vector elements will be 0 and which will be 1, she can simply encrypt a large number of 0's and a large number of 1's to use later. (Recall that because semantically secure encryption is used, each encryption of 0 will be different from the other encryptions of 0, and similarly for the encryptions of 1.) When Alice needs to send encrypted data to the server, she can just retrieve the appropriate encryptions. The optimization is useful for mobile devices, e.g. PDAs, that have limited computing power but reasonable amounts of storage.

The experimental results of this optimization are shown in Figure 7, with overall on-line execution times reduced to about 19.72 seconds for a database of $200,000$ elements, under the 1024-bit cryptographic key size. Alice's processing time, now simply to read the stored encryptions and send them to Bob, is much smaller. All other components remain unchanged; Bob's computation time becomes the dominant factor.

3.4 Using Multiple Clients in Parallel

This alternative aims at reducing the time spent by Alice in encrypting her data by partitioning the task of encryption among multiple clients, while still protecting Bob's privacy.

In this setting, k clients work in cooperation. Each client is responsible for $1/k$th of the database, and will interact with Bob to learn a partial product corresponding to the chosen data in that part of the database. However, learning these partial sums violates Bob's privacy. Accordingly, Bob uses a randomized blinding to protect the partial sums; the blinding is removed by the clients only after the partial products are combined into a single product, as shown in Figures 8 and 9 for $k = 3$.

In phase one, k clients $C_1, C_2, \ldots, C_k$ are involved each holding a vector of size n/k elements. (If the database size n is not a multiple of k, then some of the clients should hold $\lceil n/k \rceil$ elements and some should hold $\lfloor n/k \rfloor$ elements.) The clients independently and in parallel choose their own encryption keys and interact with the server to learn a blinded encryption of the appropriate partial sum. That is, the server chooses random numbers R_1, R_2, .., R_k such that $\sum_{i=1}^{k} R_i = 0 \pmod{M}$ (where again M must be chosen sufficiently large). When computing the product to return to client C_i, Bob also computes $E(R_i)$ and multiplies it into the product. This has the effect of adding R_i to the partial sum P_i.

In phase two, the clients combine their partial sums and remove the blinding factor:

1. Client C_1 sends its blinded partial sum to client C_2.

2. In turn, each client C_i adds the value received from client C_{i-1} to its own blinded sum and sends the result to client C_{i+1}.

3. Client C_k receives the blinded partial sum from client C_{k-1}, adds it to its blinded partial sum to generate the total unblinded sum, and broadcasts the result to all the other clients.

The results in Figure 10 show performance results for $k = 3$. The overall execution time is reduced by a factor of approximately 2.99, which represents a 3-fold improvement minus a small overhead for the combining phase. Note that we implemented multiple clients only for our Java implementation, so these performance numbers are significantly higher than those in earlier graphs. They are shown only to indicate the close to 3-fold improvement. The use of k clients would result in approximately a k-fold reduction in execution time.

4 Conclusions

We have experimentally evaluated an important primitive in privacy-preserving data mining, that of

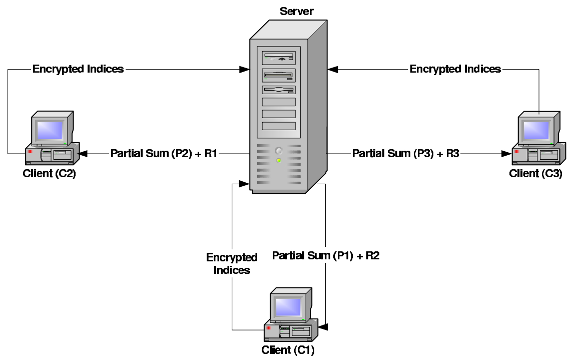

Figure 8. Multiple Clients ($k = 3$): Phase 1

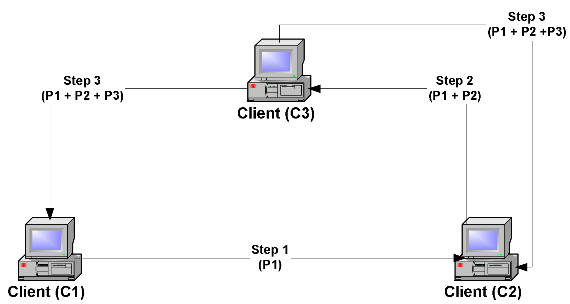

Figure 9. Multiple Clients ($k = 3$): Phase 2

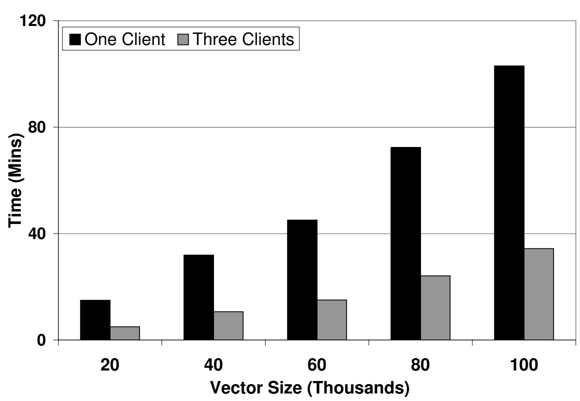

Figure 10. Performance Improvement with Three Clients (Java implementation)

privately computing shares of the scalar product between two binary vectors held by two parties. Neither party learns anything about the other party's private data.

Our experimental results show that the running time needed is quite high, though perhaps feasible in some settings where privacy is considered sufficiently important. In a direct implementation, overall running times are around 80 minutes for a database of 200,000 elements in a high-speed communication environment. With straightforward optimizations, the running times are only a few seconds, clearly within the realm of practice for many applications. Unless practical optimizations or specialized hardware are used to accelerate encryptions, computation delay is the major bottleneck of performance of our implementation.

References

[AD01] M. Atallah and W. Du. Secure multi-party computational geometry. In *Proc. of the Seventh International Workshop on Algorithms and Data Structures*, pages 165–179. Springer-Verlag, 2001.

[AMP04] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k^{th}-ranked element. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 40–55. Springer-Verlag, 2004.

[AS00] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450. ACM Press, May 2000.

[CIK+01] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proc. of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 293–304. ACM Press, 2001.

[DA01] W. Du and M. Atallah. Privacy-preserving cooperative statistical analysis. In *Proc. of the 17th Annual Computer Security Applications Conference*, pages 103–110, 2001.

[EGS03] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222. ACM Press, 2003.

[ESAG02] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–228. ACM Press, 2002.

[FIM+01] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. N. Wright. Secure multiparty computation of approximations. In *Proc. of the 28th International Colloquium on Automata, Languages and Programming*, pages 927–938. Springer-Verlag, 2001.

[FNP04] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer-Verlag, 2004.

[GLLM04] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikäinen. On private scalar product computation for privacy-preserving data mining. In *Proc. of the Seventh Annual International Conference in Information Security and Cryptology*, LNCS. Springer-Verlag, 2004. to appear.

[JW05] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 593–599. ACM Press, 2005.

[KC02] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proc. of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'02)*, pages 24–31, June 2002.

[LKR03] K. Liu, H. Kargupta, and J. Ryan. Multiplicative noise, random projection, and privacy preserving data mining from distributed multi-party data. Technical Report TR-CS-03-24, Computer Science and Electrical Engineering Department, University of Maryland, Baltimore County, 2003.

[LP02] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. of Usenix Security Symposium 2004*, 2004.

[MSK04] D. Meng, K. Sivakumar, and H. Kargupta. Privacy-sensitive bayesian network parameter learning. In *Proc. of the Fourth IEEE International Conference on Data Mining*, Brighton, UK, 2004.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residue classes. In *Advances in Cryptology – EUROCRYPT 99*, volume 1592 of *LNCS*, pages 223–238, 1999.

[SWY04] H. Subramaniam, R. N. Wright, and Z. Yang. Experimental analysis of privacy-preserving statistics computation. In *Proc. of the VLDB Worshop on Secure Data Management*, pages 55–66, August 2004.

[VC02] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 639–644. ACM Press, 2002.

[VC03] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. In *Proc. of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215. ACM Press, 2003.

[VC04] J. Vaidya and C. Clifton. Privacy preserving naive Bayes classifier on vertically partitioned data. In *2004 SIAM International Conference on Data Mining*, 2004.

[WY04] R. N. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *Proc. of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 713–718. ACM Press, 2004.

[Yao86] A. Yao. How to generate and exchange secrets. In *Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[YW05] Z. Yang and R. N. Wright. Improved privacy-preserving Bayesian network parameter learning on vertically partitioned data. In *Proc. of the International Workshop on Privacy Data Management (held in conjunction with ICDE '05)*, Tokyo, Japan, April 2005.