Efficient Cryptographic Primitives for Private Data Mining

Mark Shaneck*
Liberty University
Dept. of Computer Science
Lynchburg, VA
mshaneck@liberty.edu

Yongdae Kim
University of Minnesota
Dept. of Computer Science
Minneapolis, MN
kyd@cs.umn.edu

Abstract

Data mining is frequently obstructed by privacy concerns. In many cases data is distributed, and bringing the data together in one place for analysis is not possible due to privacy laws (e.g. HIPAA) or policies. Privacy preserving data mining techniques have been developed to address this issue by providing mechanisms to mine the data while giving certain privacy guarantees. However, when these techniques are built on cryptographic primitives, while providing strong privacy, they are often too inefficient to be used in practical settings. To this end, we address the problem of efficiency by investigating trade-offs that can be made in the trust model. By making reasonable concessions in the trust model, that is, by adding a non-collaborative third party, we can achieve great gains in efficiency. We show this by creating a novel protocol for privately computing dot product, a foundational primitive for many private data mining activities. We also investigate how to extend our protocol in the case when a third party cannot be completely trusted by both participating parties, thus reducing the amount of trust needed in the third party. We then show experimentally the gains in efficiency that can be realized in the computation of the private dot product using this model.

1. Introduction

Techniques for Privacy Preserving Data Mining were introduced to address concerns by privacy advocates. The growing need to analyze data leads to greater access to the data. If this data contains private information, steps must be taken to ensure that the privacy is not breached.

This concern, while having a beneficial effect on the protection of private information, can have a negative effect on the analysis of the data. In many cases data is distributed, and bringing the data together in one place for analysis is not possible due these privacy laws or policies. Now it is not the (honest) data miner's intention to compromise any private information. In fact, the results of the data mining rarely compromise privacy. This is due to the fact that data mining itself is concerned with retrieving high level data and trends about the data. The main purpose of the data mining is to take the large amounts of information, that would be impossible to analyze on an individual record by record basis, and extract some interesting trends or statistics.

To this end, a large amount of research has focused on addressing the problem of how to extract this high level information from the distributed data sets, while providing guarantees on the protection of the privacy of the data. This field of research, called Privacy Preserving Data Mining, typically concerns itself with one of two problems. The first is preparing the data for release. That is, the private data will be released to the data miner (or the public), and thus the data must first be altered in such a way so as to prevent any privacy breaches. The second angle of attack for privacy preserving data mining research is to modify the data mining algorithm itself to allow it to be run in a distributed way such that no private information is released to the other participating parties. This second method, which is the focus of this work, typically makes use of secure multiparty computation techniques from the field of cryptography.

This second approach, while typically being able to provide superior privacy guarantees, has a major limitation. Since it relies on techniques from Secure Multiparty Computation, the underlying primitives upon which these algorithms are built are usually very inefficient. Typically these underlying primitives are built upon public key encryption techniques, which have significant computation requirements. Now when performed infrequently, these techniques are very useful. However, due to the nature of data mining problems, that is that they are built to examine vast amounts of data, these underlying primitives must often be executed many times, leading to algorithms that are unable to be applied practically on large data sets.

In this paper we investigate what compromises can be made in order to achieve a practical scheme for privacy

*Work done while at the University of Minnesota

IEEE computer society

preserving data mining. In this work, we focus on one of the most important cryptographic primitives used in privacy preserving data mining - the private computation of dot product. By examining the underlying trust model of the private computations, we can introduce a non-collaborative third party to the effect of greatly simplifying the computations. Previous works [19, 18, 46] have investigated a similar trust model, yet the schemes built were inflexible in the amount of trust required in the third party, in that the third party cannot be distributed. For more details, we refer the reader to Section 5.

The contributions of this work are as follows. First, we modify the trust model to include a semi-trusted third party and examine the practicality of this modification. We then develop a protocol to privately compute the dot product. We also discuss how the dot product protocol can be distributed across multiple "third parties", thus reducing the amount of trust in any single "third party". Finally, we implemented the polynomial-based dot product protocol and evaluated it, comparing it to the current state of the art in secure dot product computation, based on homomorphic encryption.

The remainder of this paper is organized as follows. First we discuss the modification of the trust model and its implications in Section 2. We then discuss the improved protocols in Section 3, including a discussion of its limitations and how they can be address by distributing the third party, reducing the amount of trust required in the third party. We then show in Section 4 the details of the implementation and the results of comparing it with the other main dot product primitive. Finally, we discuss related work in Section 5 and conclude in Section 6.

2. Modified Trust Model

One of the main reasons why efficient secure primitives have not been achieved is because most attempts have been made to ensure that the algorithm is secure in a general setting. This is a desirable goal, but frequently renders the resulting algorithm too inefficient for practical situations. This section, however, addresses the question of what compromises can we make to allow us to construct an efficient and usable dot product algorithm. If there are some reasonable trade-offs that we can make, while not sacrificing security or trivializing the problem, which allow us to construct efficient primitives, then we can enable cryptographic privacy preserving data mining techniques to be more useful in practical settings. It is our claim that, by making the trade-off of a third party, trusted only to not collaborate with others, then we can obtain an efficient protocol for privately computing dot products, comparisons, and divisions. Further, the construction of the dot product protocol allows the non-collaborating third party to be distributed across multiple hosts, reducing the amount of total trust needed in any single extra party.

At this point the question becomes, is the introduction of a non-collaborating third party really a realistic assumption that we can make? Since we are interested in making cryptographic approaches workable for practical settings, it is important that the model for the problem is a realistic one as well. We claim that this model is in fact realistic in many application scenarios. One situation where this would be a realistic model is one in which the participants are in a cooperative relationship and are mainly prevented from obtaining access to other data by privacy laws. An example of this would be medical data, where doctors or medical researchers are using the data to, for example, diagnose a patient. In this case the third party could be another hospital or even a government agency, such as the Centers for Disease Control and Prevention (CDC) [13], which would be interested in having the results be accurately computed, and could be trusted not to actively try to compromise the data of the patients. Another possibility would be that the third party could be played by an unbiased organization, such as a consumer advocacy group. For example, if the participants were insurance companies trying to perform distributed fraud detection, the third party could be played by the Coalition Against Insurance Fraud [12], which would have an interest not only in participating correctly (to further work against insurance fraud), but also in protecting the privacy of the insured (as they are made up, in part, of consumers and consumer groups).

Attempts have been made in the past to introduce a non-collaborating third party [19, 18, 46]. One of the major limitations of these past approaches is that the third party cannot be distributed. This limits the applicability of the proposed schemes to ones in which the trust to not collude can be placed in a single party. However, in our approach for dot product, the proposed protocol easily distributes from a single non-collaborating party to many, where the trust assumption is that the majority of the parties do not collude with one of the protocol participants (i.e. Alice or Bob).

3. Dot Product Protocol

This primitive is the foundation for many other higher level computations, especially the computation of Euclidean distance shares [47]. In addition, many private data mining protocols have been built using dot product as the main component (e.g. [45]). Thus many private data mining computations can benefit directly from this protocol. The dot product calculation simply computes the value of the dot product of two vectors, where each party has one vector as its input. In this work, we are interested in the computation of additive shares of the result. In other words, we are computing the function $PDP : (\vec{x}, \vec{y}) \mapsto (c_1, c_2)$ where both c_1 and c_2 are uniformly distributed in $\mathbb{Z}_p$ when viewed in-

dependently, and $c_1 + c_2 = \vec{x} \cdot \vec{y} \mod p$, for sufficiently large p.

3.1 Polynomial Secret Sharing

Our approach is built upon the technique of polynomial secret sharing [42], in which a number (i.e. the secret) is shared by creating a random polynomial over a field which has as its y intercept the secret, and the shares are simply the values of the polynomial evaluated at different x values. The degree of the polynomial determines the number of shares required to reconstruct the secret.

Polynomial secret sharing was introduced by Shamir [42] as a method to efficiently share secrets in a way that the secret can be reconstructed when a threshold number of parties collaborate. This is done, informally, by constructing a random polynomial f , with the degree t equal to the desired threshold minus 1, and the free coefficient equal to the secret s. The secret is then shared by evaluating the polynomial at different points i, and sending the values $f(i)$ to different players. Now as a property of polynomials, given any w shares, where $w \leq t$, a player cannot learn any information about the secret s, since there are an infinite number of polynomials of degree t passing through those w points. However, with the addition of just one more share, the secret can be retrieved, by reconstructing the unique t-degree polynomial that passes through those $t + 1$ points.

Furthermore, it has been shown that simple arithmetic (i.e. addition and multiplication) can be done using polynomial shares [8, 23]. If the polynomial f shares the secret s, and the polynomial g shares the secret r, then the polynomial $f + g$ shares the secret $s + r$, and thus addition of two shares can be done by locally summing shares. Multiplication is similar, in that multiplication of shares results in a polynomial that shares the product of the secrets, however the resulting polynomial is of greater degree and has coefficients that are not independently random. This, however, can be addressed efficiently, by using simple arithmetic and two rounds of communication.

For completeness, we will briefly describe how this can be achieved. Let us assume that A (with an ID of 1) has the secret s and that B (with an ID of 2) has the secret r. Also, let us assume that the party C (with an ID of 3) is another participant in the protocol. A first sends shares of its secret to B and C, namely $f(2)$ and $f(3)$, and stores its own share $f(1)$. B also sends shares of its secret to A and B, namely $g(1)$ and $g(3)$ and stores its share $g(2)$. Now each party can compute the multiplication of their shares, i.e. A computes $f(1)g(1)$, B computes $f(2)g(2)$, and C computes $f(3)g(3)$. Now these values are re-shared with new polynomials, that is $h_A(x)$ shares the value $f(1)g(1)$, $h_B(x)$ shares the value $f(2)g(2)$, and $h_C(x)$ shares the value $f(3)g(3)$. Thus, $h_B(1)$ and $h_C(1)$ are sent to A, and so on. Now each party can take the shares that they have to compute new shares of the multiplication, by multiplying the shares with the value from the first row of the inverse of the $(2t + 1) \times (2t + 1)$ Van Der Monde matrix defined by $a_{ij} = i^{j-1}$. In our case, $t = 1$, and the values are $\lambda_1 = 3, \lambda_2 = -3$, and $\lambda_3 = 1$. Thus, for example, A computes its share as $\lambda_1 h_A(1) + \lambda_2 h_B(1) + \lambda_3 h_C(x)$. This is shown in Figure 1.

Thus, given polynomial shares of secrets, simple addition and multiplication can be done efficiently, using only multiplications and not modular exponentiations.

3.2. Basic Protocol

Since we are able to perform multiplication and addition of numbers that are shared in this fashion, we can compute the dot product. We assume that there are two parties and each has n vectors, each with d dimensions. Since there are two parties, we can set the degree of the random polynomials to 1 (thus requiring 2 shares to reconstruct the secret).

In a general case, we will want to compute all n^2 pairwise dot products (i.e. if we want to compute all pairwise distances between the data points in two data sets), and thus we describe how to compute these dot products in parallel. For applications that only require single dot products, it is straightforward to set $n = 1$, and our technique will reduce to that case. Since the multiplications are done first, and since polynomial secret shares can be added locally, without any interaction, we can compute all the multiplications in parallel first, followed by the local additions, to compute all the needed shares. Since the polynomial secret share multiplication scheme requires two rounds of communication, and we can do all the multiplications in parallel, then we only need two rounds of communication to compute all n^2 dot products.

More formally, let us assume that A has n vectors $\vec{x_1}, \ldots, \vec{x_n}$ and that B has n vectors $\vec{y_1}, \ldots, \vec{y_n}$. Each vector has d dimensions, which will be denoted as $\vec{x_i} = \langle x_{i1}, \ldots, x_{id} \rangle$. Together they are trying to compute $\vec{x_i} \cdot \vec{y_j}$ for $1 \leq i, j \leq n$. Thus, they need to compute the multiplications $x_{ik}y_{jk}$ for $1 \leq i, j \leq n$ and $1 \leq k \leq d$. These values can be computed using the scheme from [8, 23], given that the individual components are first shared polynomially. Let us call A's (resp. B's) shares resulting from this multiplication $(x_{ik}y_{jk})^A$ (resp. $(x_{ik}y_{jk})^B$). Once these are computed, the dot product shares $(\vec{x_i} \cdot \vec{y_j})^A$ and $(\vec{x_i} \cdot \vec{y_j})^B$ can be computed by locally computing (respectively) $\sum_{k=1}^{d} (x_{ik}y_{jk})^A$ and $\sum_{k=1}^{d} (x_{ik}y_{jk})^B$. This, when done for each i and j will give the polynomial shares for each pairwise dot product.

The specification of the dot product requires that the output be additively shared, so we must convert from poly-

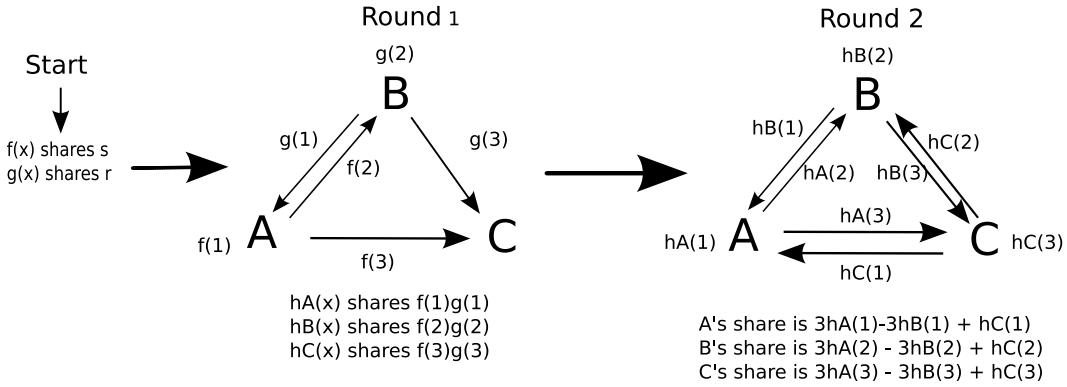

Figure 1: Polynomial Share Multiplication Protocol

nomial shares to additive shares. However, this is a well known technique [2]. Since this dot product protocol is simply a repeated application of the secure polynomial secret share multiplication protocol, the security proof that this dot product protocol is information-theoretically secure is straightforward, and thus is omitted.

3.3 Distributing the Third Party

One of the most obvious drawbacks from this trust model is the fact that it requires the use of a third party. In order to multiply the polynomial shares [8, 23] for a polynomial of degree t, we need to have at least $2t + 1$ players. In our case $t = 1$ (since there are two parties), and thus we require 3 parties. In making this observation, we must discuss what role this extra player must play, and how must trust is required for this player. It turns out that this player needs only to be trusted to not collaborate with either of the other parties. If the third player does not collaborate, he only sees uniformly distributed random values, and thus learns no information about the other parties' data. However, since the degree of the polynomial is 1, any two shares of a given secret are sufficient to recover the secret. Thus, the security of our scheme relies on the non-collaboration of the third party that is used.

However, it would be desirable to reduce the amount of trust in this third party. This can be achieved by using more than one "third" party. By the nature of polynomial secret sharing schemes, we can simply use a higher degree polynomial. The effect of increasing the degree of the polynomial is such that more parties would need to collaborate to break the security of the protocol. In fact, in order to break the protocol, a majority of the third parties will need to collaborate. Of course, increasing the number of parties also increases the required communication. For $m = 2t+1$ parties, the communication requirement is $O(m^2)$. Thus the level of trust required in the additional parties can be distributed across several entities.

4. Dot Product Protocol Efficiency

In this section we compare the efficiency of the proposed dot product protocol with the standard protocol based on homomorphic encryption, followed by a discussion on the limitations of this approach as opposed to other approaches.

In this section we examine how the performance of our dot product protocol compares to the standard method of computing dot product [25], which is based on homomorphic encryption. When computing the dot product using homomorphic encryption, one party encrypts each vector component individually, which it then sends to the other party. Due to the homomorphic properties of Paillier's cryptosystem [39], both addition and multiplication can be performed on the plain-texts, without ever decrypting. This enables the dot products to be computed without decrypting, and a random value added, with this resulting encrypted value sent back to the original party. This party can then decrypt, thus providing both parties with additive shares, which appear random when viewed independently. We have implemented both this scheme and the polynomial based scheme (using the GMP [24] library).

One of the most intensive uses of this dot product protocol are those in which all pairwise dot products are computed between two (horizontally partitioned) sets of data [43, 47]. This is the situation that we emulate in our experimental analysis. In our experiment, we generate n vectors (randomly) with d dimensions. We then execute the dot product protocol to compute shares of all n^2 pairwise dot products. This time is measured and plotted for the different values of n. We kept d constant, equal to 10 throughout these experiments. Both protocols are linearly dependent on d, and one can extrapolate the results to see how the performance would compare when d is increased. The most important item in the performance is that while the number of additions and multiplications are somewhat higher for the polynomial scheme, the additions and multiplications done in the homomorphic scheme are executed using modular

exponentiation, whereas the polynomial scheme computes them with regular addition and multiplication.

Before looking at the results from the experiments, let us take a look at the expected results, based on a theoretical analysis. First, for the communication we can see that the number of elements that are sent (total bandwidth) for the homomorphic scheme is $nd + n^2$, as first each vector component is encrypted and sent by the first party to the second party, and then the n^2 dot product shares are sent back to the first party. The polynomial scheme has a higher required number of elements, with $4nd + 4n^2d$[1]. However, since the elements for the homomorphic scheme are ciphertexts (from Paillier's scheme), and we are using an RSA modulus of 1024 bits, then the elements are each 2048 bits. On the other hand, the elements in the polynomial scheme are simply elements from the field we are working in, which is fixed at $2m + d + 1$ bits. Unless the data mining application requires very large numbers, or (more likely) very high number of dimensions, the polynomial scheme will in fact require less overall bandwidth.

For the computation requirement, we see a much larger difference between the two schemes. Let us assume that, for the homomorphic scheme, party A encrypts all of its components and sends them to B, who performs the homomorphic operations, and finally A decrypts the shares. We also assume that each party has m vectors with d dimensions. Also, we assume that Paillier is optimized through the use of precomputation, where the encryptions are partially computed ahead of time, which reduces the number of modular exponentiations from 2 to 1 per encryption. Finally, we will ignore modular additions, as they will be vastly overshadowed by the modular multiplications and exponentiations.

We can see that the total computational load for A is md encryptions and m^2 decryptions. Similarly, for B the load is m^2d homomorphic multiplications, m^2d+m^2 homomorphic additions, and m^2 encryptions. Now to express this in terms of modular exponentiations, we note that each Paillier encryption requires 1 modular exponentiation and 1 modular multiplication, and each decryption requires 1 modular exponentiation and 2 modular multiplications. Also, a homomorphic multiplication under the Paillier scheme requires 1 modular exponentiation and a homomorphic addition requires 1 modular multiplication. This all adds up to a load of $m^2 + md$ modular exponentiations and $2m^2 + md$ modular multiplications for A and $m^2d + m^2$ modular exponentiations and $m^2d + 2m^2$ modular multiplications for B. We also point out here that the modular operations are done using the value n^2 as the modulus, where n is an RSA modulus. Thus, for a 1024 bit RSA modulus, the operations are done using a 2048 bit modulus. Now for the polynomial scheme, let us assume that we are using polynomials of degree 1, since we are evaluating a two-party function. Thus, a polynomial evaluation is equivalent to 1 modular multiplication. In the first round of the protocol, both A and B are required to perform $3md$ polynomial evaluations, and thus $3md$ modular multiplications. In the second round of the protocol, all three parties are required to perform $2m^2d$ polynomial evaluations, with A and B performing an additional $4m^2d$ modular multiplications. Thus, A and B have to perform a total of $6m^2d + 3md$ modular multiplications and the third party must perform $2m^2d$ modular multiplications. Also note here that the modular operations are done within the field specified above, which is larger than the largest possible dot product. The computational load is summarized in Table 1.

We can see that while the polynomial scheme requires a small constant factor more modular multiplications, there are no modular exponentiations. As such, the polynomial scheme will vastly outperform the homomorphic encryption based scheme in terms of computational load.

Note that in [48] various comparisons were performed using some optimizations that were mentioned in [25]. We did not implement these optimizations, however we refer the interested reader to [48] for details about how these optimizations can improve the efficiency of the homomorphic encryption based dot product protocol. The majority of these optimizations are proposed for binary vectors, and as we are targeting a more general solution, these are not included in this comparison.

4.1. Experimental Evaluation

We implemented both the polynomial scheme described in this paper and the standard homomorphic scheme [25]. We used Paillier's cryptosystem [39] as the homomorphic encryption scheme required, which we implemented using the GMP Library [24]. The homomorphic scheme required two parties, for which we used two computers. The second party, which performs the homomorphic operations, was run on a 3.8 GHz dual-processor 64-bit machine, with 2 MB cache and 2 GB memory. The first party, which performs the initial encryption and final decryption, was run on a 2.2 GHz single processor 32-bit machine with 512 KB cache and 1 GB memory. These two machines were both located on the University of Minnesota network, separated by 2 hops, with a mean ping time of approximately 0.6 milliseconds. For the polynomial scheme, the parties designated A and B were both run on the dual processor machine, with the non-collaborating third party run on the single processor machine.

The first graph (Figure 2) shows the comparison between the homomorphic scheme and the polynomial scheme. The

[1] This assumes a certain optimization where the third party does nothing that it does not need to do. In particular, it does not compute shares of the product as the two main parties do, but rather participates at a minimum level.

Table 1: Comparison of Computation Requirements

	Homomorphic Scheme		Polynomial Scheme	
Party	Mod. Exp.	Mod. Mult.	Mod. Exp.	Mod. Mult.
A	$m^2 + md$	$2m^2 + md$	0	$6m^2d + 3md$
B	$m^2d + m^2$	$m^2d + 2m^2$	0	$6m^2d + 3md$
C	-	-	0	$2m^2d$

numbers shown are for the party that took the longest[2]. As we can see, the polynomial scheme provides a speedup of over three orders of magnitude, around 1000X (for $n = 100$) to 1500X (for $n = 500$). This is quite significant in that for $n = 500$, i.e. where each party has 500 vectors, it takes only about 13 seconds for the polynomial scheme, as opposed to 5.5 hours for the homomorphic encryption scheme. We can see this as well in the projected run times, shown in Figure 3, which shows projected run times based on the experimental results for values of n from $1,000$ to $1,000,000$. For n equal to $1,000,000$, the homomorphic scheme will take almost $2,500$ years to complete while the polynomial scheme takes just 1.68 years. Now obviously neither scheme is really feasible for such a large input, and we discuss future work to approach this in Section 6. However, as we look at smaller sized problems, we see that the proposed dot product scheme can realistically be used. For example, for $n = 100,000$, the polynomial scheme will finish in (projected) 6 days, whereas the homomorphic scheme will take (projected) 24 years. For $n = 10,000$, the polynomial scheme will take (projected) 1.5 hours to complete, whereas the homomorphic scheme will take (projected) over 90 days. Thus, using this scheme, we can see that sizable problems can be tackled using this scheme in realistic amounts of time.

5. Related Work

Much work has been done in the field of Secure Multiparty Computation, starting with the seminal work by Yao [49], when he posed the Millionaire's Problem. This led to the proposal of the general solution [26] which states that any computable functionality can be computed securely. At a high level this works since every computable functionality can be reduced to the evaluation of a circuit, and the general solution gives protocols for computing a logically complete set of Boolean circuit gates. However, this is typically too inefficient for computations of any significant size, and thus much work has been done to reduce problems to certain primitives and to provide solutions for those primitives. These primitives include protocols such as Oblivious Transfer [40, 22, 16, 38, 44, 1, 36], Oblivious Polynomial Evaluation [37], Homomorphic Encryption [39], and Secure Comparison [28, 31, 9, 35] (i.e. solutions to Yao's Millionaire Problem).

In addition to general secure primitives, there are primitives that are highly applicable to data mining. The most prominent of these is the dot product, to which we proposed an improvement in this work. Many solutions to the dot product have been proposed [19, 18, 29, 45, 25, 17, 41], and since it is of such importance in this work we will highlight some of the proposed solutions.

The most closely related work is by Du and Zhan [19, 18] in which they propose a similar trust model, in which a third party is introduced that is trusted to not collaborate. This third party sends random values to the two parties participating in the dot product protocol, which they can use to compute the shares of the dot product. The efficiency gain is similar to ours. However the main difference is that since the third party sends random values which are used to blind the vector components, the third party cannot be distributed to reduce the trust, while our solution for dot product has the ability to increase the degree of the polynomial and include multiple third parties, with the stipulation that a majority of them are trusted to not collaborate. Thus, while the two solutions are similar in the case of three parties, our dot product solution has a distinct advantage in that it can be extended to five or more parties to reduce the amount of trust in any given party.

Other dot product protocols have been proposed that make use of random matrices to blind the data [29, 45]. These protocols require only two parties to be involved and do not require public key techniques for their computation and thus are efficient. However, they do not have security proofs, and thus it is difficult to guarantee that they do not leak information. In fact, one [45] has been proven to be insecure [25]. Also, dot product protocols have been proposed that make use of Oblivious Transfer [17]. This too was proven insecure [25], and thus stresses the importance of using provably secure protocols, instead of protocols that have security arguments without an associated proof of security. Without these proofs, we cannot be certain that the protocols are indeed secure.

Dot product protocols have also been proposed that make use of public key techniques, most notably Goethals et al. [25], where homomorphic encryption is used to construct

[2]Since the messaging was asynchronous, the parties can have different (though similar) run times, and thus the longest run time was selected in each case as representative of the worst case scenario.

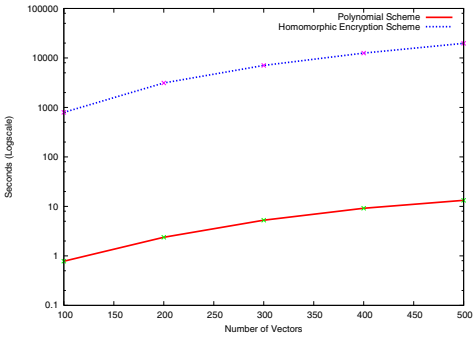

Figure 2: Experimental Run Times

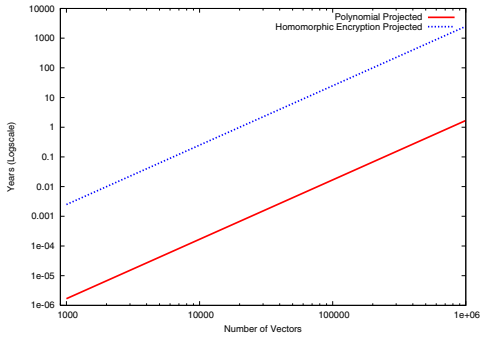

Figure 3: Projected Run Times For Large Inputs

a provably secure dot product protocol. The major limitation however of this approach is that it is computationally intensive, and too inefficient to be used in computing large numbers of dot products. Other techniques for computing dot products include using stochastic processes [41].

6. Conclusion and Future Work

In this work we have explored the benefits that can be gained from a change in the trust model. By introducing a semi-trusted (non-collaborative) third party, we can achieve significant gains in performance, bringing many applications of cryptographically secure data mining much closer to being practical on data sets of useful size.

For future work we intend to continue to investigate trade-offs that can be made to improve performance, in particular for the area of nearest neighbor search, a higher level data mining computation that relies on the primitives discussed in this paper, and forms the basis of many other data mining algorithms, such as LOF outlier detection [10, 11], kNN classification [15], and SNN clustering [20, 21, 30]. The main trade-off we intend to investigate is in accuracy. Much work has been done in the area of approximate nearest neighbor search [5, 3, 4, 6, 7, 14, 27, 32, 33]. We intend to investigate if these methods can also be applicable in a privacy preserving setting. Another method of achieving a performance gain in nearest neighbor search is to reduce the search space. By searching a subset of the remote data set for neighbors of a given point, we can reduce the complexity of the search, with the obvious implication that some true neighbors may be missed. However, if the search space is reduced in such a way that mainly nearby points are searched and remote points are ignored, then the missed neighbors will be minimized, and the erroneous neighbors that are found may be close enough. However metrics to accurately measure the effectiveness of the search need to be developed and ideally, guarantees need to be provided, so that there are firm bounds placed on the loss of accuracy, and quantitative measures in place of the actual effects of the accuracy loss.

References

[1] Bill Aiello, Yuval Ishai, and Omer Reingold. Priced Oblivious Transfer: How to Sell Digital Goods. In *Proceedings of Advances in Cryptology - Eurocrypt*, 2001.

[2] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, 2002.

[3] Sunil Arya and Theocharis Malamatos. Linear-Size Approximate Voronoi Diagrams. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2002.

[4] Sunil Arya, Theocharis Malamatos, and David Mount. Space-Efficient Approximate Voronoi Diagrams. In *Proceedings of the Symposium on Theory of Computing*, 2002.

[5] Sunil Arya and David Mount. Approximate Nearest Neighbor Queries in Fixed Dimensions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[6] Sunil Arya, David Mount, Nathan Netanyahu, Ruth Silverman, and Angela Wu. An Optimal Algorithm for Approximate Neaest Neighbor Searching in Fixed Dimensions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[7] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[8] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the 20th Annual Symposium on the Theory of Computing*, 1988.

[9] Ian Blake and Vladimir Kolesnikov. Conditional Encrypted Mapping and Comparing Encrypted Numbers. In *Proceedings of Financial Cryptography*, 2006.

[10] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. OPTICS-OF: Identifying Local Outliers. In *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, 1999.

[11] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the ACM International Conference on Management of Data*, 2000.

[12] Coalition Against Insurance Fraud. `http://www.insurancefraud.org`, 2007.

[13] Centers for Disease Control and Prevention. `http://www.cdc.gov/`, 2007.

[14] Kenneth Clarkson. An Algorithm for Approximate Closest-point Queries. In *Proceedings of the ACM Symposium on Computational Geometry*, 1994.

[15] T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[16] Claude Crépeau. Equivalence Between Two Flavours of Oblivious Transfer. In *Proceedings of Advances in Cryptology*, 1987.

[17] Wenliang Du and Mikhail Atallah. Privacy-Preserving Cooperative Statistical Analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001.

[18] Wenliang Du and Zhijun Zhan. A Practical Approach to Solve Secure Multi-party Computation Problems. In *Proceedings of the 2002 Workshop on New Security Paradigms*, 2002.

[19] Wenliang Du and Zhijun Zhan. Building Decision Tree Classifier on Private Data. In *Proceedings of the Workshop on Privacy, Security, and Data Mining at The 2002 IEEE International Conference on Data Mining*, 2002.

[20] Levent Ertöz, Michael Steinbach, and Vipin Kumar. A New Shared Nearest Neighbor Clustering Algorithm and its Applications. In *Workshop on Clustering High Dimensional Data and its Applications, Proceedings of Text Mine '01, First SIAM International Conference on Data Mining*, 2001.

[21] Levent Ertöz, Michael Steinbach, and Vipin Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Proceedings of the SIAM International Conference on Data Mining*, 2003.

[22] S. Even, O. Goldreich, and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[23] Rosario Gennaro, Michael Rabin, and Tal Rabin. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing*, 1998.

[24] GNU Multiple Precision Arithmetic Library. `http://www.swox.com/gmp/`, 2007.

[25] Bart Goethals, Sven Laur, Helger Lipmaa, and Taneli Mielikëinen. On Private Scalar Product Computation for Privacy-Preserving Data Mining. In *Proceedings of the 7th Annual International Conference in Information Security and Cryptology*, 2004.

[26] Oded Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004.

[27] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Symposium on Theory of Computing*, 1998.

[28] Ioannis Ioannidis and Ananth Grama. An Efficient Protocol for Yao's Millionaire's Problem. In *Proceedings of the Hawaii International Conference on System Sciences*, 2003.

[29] Ioannis Ioannidis, Ananth Grama, and Mikhail Atallah. A Secure Protocol for Computing Dot-Products in Clustered and Distributed Environments. In *Proceedings of the 31st International Conference on Parallel Processing*, 2002.

[30] R. A. Jarvis and E. A. Patrick. Clustering Using a Similarity Measure Based on Shared Nearest Neighbors. *IEEE Transactions on Computers*, C22(11):1025–1034, 1973.

[31] Eike Kiltz, Ivan Damgaard, Matthias Fitzi, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant Round Multi-Party Computation for Equality, Comparison, Bits and Exponentiation. In *Proceedings of the third Theory of Cryptography Conference*, 2006.

[32] Jon Kleingberg. Two Algorithms for Nearest-Neighbor Search in High Dimensions. In *Proceedings of The ACM Symposium on Theory of Computing*, 1997.

[33] Eyal Kushlevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. In *Proceedings of The ACM Symposium on Theory of Computing*, 1998.

[34] Sven Laur and Helger Lipmaa. On Private Similarity Search Protocols. In *Proceedings of the 9th Nordic Workshop on Secure IT Systems*, 2004.

[35] Hsiao-Ying Lin and Wen-Guey Tzeng. An Efficient Solution to the Millionaires' Problem Based on Homomorphic Encryption. In *Proceedings of the Third International Conference on Applied Cryptography and Network Security*, 2005.

[36] Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *Proceedings of the 8th Information Security Conference*, 2005.

[37] Moni Naor and Benny Pinkas. Oblivious Transfer and Polynomial Evaluation. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 1999.

[38] Moni Naor and Benny Pinkas. Efficient Oblivious Transfer Protocols. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, 2001.

[39] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of Eurocrypt*, 1999.

[40] Michael Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report Technical Report TR-81, Harvard University, 1981.

[41] Pradeep Ravikumar, William W. Cohen, and Stephen E. Fienberg. A Secure Protocol for Computing String Distance Metrics. In *Proceedings of the ICDM Workshop on Privacy and Security Aspects of Data Mining*, 2004.

[42] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):512–613, 1979.

[43] Mark Shaneck, Yongdae Kim, and Vipin Kumar. Privacy Preserving Nearest Neighbor Search. Technical Report 06-014, University of Minnesota, 2006.

[44] Wen-Guey Tzeng. Efficient 1-Out-of-n Oblivious Transfer Schemes with Universally Usable Parameters. *IEEE Transactions on Computers*, 53(2):232–240, 2004.

[45] Jaideep Vaidya and Chris Clifton. Privacy-Preserving Association Rule Mining in Vertically Partitioned Data. In *Pro-*

ceedings of the ACM International Conference on Knowledge Discovery and Data Mining, 2002.

[46] Jaideep Vaidya and Chris Clifton. Leveraging the "Multi" in Secure Multi-Party Computation. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, 2003.

[47] Jaideep Vaidya and Chris Clifton. Privacy-Preserving Outlier Detection. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, 2004.

[48] Zhiqiang Yang, Rebecca Wright, and Hiranmayee Subramaniam. Experimental Analysis of a Privacy-Preserving Scalar Product Protocol. *International Journal of Computer Systems Science and Engineering*, 21(1):47–52, 2006.

[49] A. C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.