P4P: Practical Large-Scale Privacy-Preserving Distributed Computation Robust against Malicious Users

Yitao Duan
NetEase Youdao
Beijing, China
duan@rd.netease.com

John Canny
Computer Science Division
University of California, Berkeley
jfc@cs.berkeley.edu

Justin Zhan
National Center for the Protection of Financial Infrastructure
South Dakota, USA
justin.zhan@ncpfi.org

Abstract

In this paper we introduce a framework for privacy-preserving distributed computation that is practical for many real-world applications. The framework is called Peers for Privacy (P4P) and features a novel heterogeneous architecture and a number of efficient tools for performing private computation and ensuring security at large scale. It maintains the following properties: (1) Provably strong privacy; (2) Adequate efficiency at reasonably large scale; and (3) Robustness against realistic adversaries. The framework gains its practicality by decomposing data mining algorithms into a sequence of vector addition steps that can be privately evaluated using a new verifiable secret sharing (VSS) scheme over *small* field (e.g., 32 or 64 bits), which has the same cost as regular, non-private arithmetic. This paradigm supports a large number of statistical learning algorithms including SVD, PCA, k-means, ID3, EM-based machine learning algorithms, etc., and all algorithms in the statistical query model [36]. As a concrete example, we show how singular value decomposition (SVD), which is an extremely useful algorithm and the core of many data mining tasks, can be done efficiently with privacy in P4P. Using real-world data and actual implementation we demonstrate that P4P is orders of magnitude faster than existing solutions.

1 Introduction

Imagine the scenario where a large group of users want to mine their collective data. This could be a community of movie fans extracting recommendations from their ratings, or a social network voting for their favorite members. In all the cases, the users may wish not to reveal their private data, not even to a "trusted" service provider, but still obtain verifiably accurate results. The major issues that make this kind of tasks challenging are the scale of the problem and the need to deal with cheating users. Typically the quality of the result increases with the size of the data (both the size of the user group and the dimensionality of per user data). Nowadays it is common for commercial service providers to run algorithms on data set collected from thousands or even millions of users. For example, the well-publicized Netflix Prize (http://www.netflixprize.com/) data set consists of roughly 100M ratings of 17,770 movies contributed by 480K users. At such a scale, both private computation and verifying proper behavior become impractical (more on this). In other words, privacy technologies fail to catch up with data mining algorithms's appetite and processing capability for large data sets.

We strive to change this. Our goal is to provide a privacy solution that is practical for many (but not all) real-world applications at reasonably large scale. We introduce a framework called Peers for Privacy (P4P) which is guided by the natural incentives of users/vendors and today's computing reality. On a typical computer today there is a six orders of magnitude difference between the crypto operations in large field needed for secure homomorphic computation (order of milliseconds) and regular arithmetic operations in small (32- or 64-bit) fields (fraction of a nano-second). Existing privacy solutions such as [11, 29] make heavy use of public-key operations for information hiding or verification. While they have the same asymptotic complexity as the standard algorithms for those problems, the constant factors imposed by public-key operations are prohibitive for large-scale systems. We show in section 3.3 and section 7.2 that they cannot be fixed with trivial changes to support applications at our scale. In contrast, P4P's main computation is based on verifiable secret sharing (VSS) over *small* field. This allows private arithmetic operations to have the *same* cost as regular, non-private arithmetic since both are manipulating the same-sized numbers with similar complexity. Moreover, such a paradigm admits extremely efficient zero-knowledge (ZK) tools that are practical even at large scale. Such tools are indispensable in dealing with cheating participants.

Some of techniques used in P4P were initially introduced in [21]. However, the focus of [21] is to develop an efficient zero-knowledge proof (ZKP) (for detecting cheating users) and prove its effectiveness. It leaves open how the ZKP should be incorporated into the computation to force proper behavior. As we will show, this is not trivial and requires additional tools, probably tailored to each application. In particular, [21] does not deal with the threat of cheating users changing their data during the computation. This could cause the computation to produce incorrect results. Such practical issues are not addressed in [21].

We fill in the missing pieces and provide a comprehensive solution. The contributions of this paper are: (1) We identify three key qualifications a practical privacy solution must possess, examine them in light of the changes in large-scale distributed computing, and formulate our design. The analysis not only provides rationales for our scheme, but also can serve as a guideline for practitioners to appraise the cost for obtaining privacy in their applications. (2) We introduce a new ZK protocol that verifies the consistency of user's data during the computation. This protocol complements the work of [21] and ensures the correctness of the computation in the presence of active user cheating. (3) We demonstrate the practicality of the framework with a concrete example, a private singular value decomposition (SVD) protocol. Prior to our work, there is no privacy solution providing comparable performance at such large scales. The example also serves as a tutorial showing how the framework can be adapted to different applications. (4) We have implemented the framework and performed evaluations against alternative privacy solutions on real-world data. Our experiments show a dramatic performance improvement. Furthermore, we have made the code freely available and are continuing to improve it. We believe that, like other secure computation implementations such as [46, 39, 5, 40], P4P is a very useful tool for developing privacy-preserving systems and represents a significant step towards making privacy a practical goal in real-world applications.

2 Preliminaries

We say that an adversary is passive, or semi-honest, if it tries to compute additional information about other player's data but still follows the protocol. An active, or malicious adversary, on the other hand, can deviate arbitrarily from the protocol, including inputting bogus data, producing incorrect computation, and aborting the protocol prematurely. Clearly active adversary is much more difficult to handle than passive ones. Our scheme is secure against a hybrid threat model that includes both passive and active adversaries. We introduce the model in section 4.

The privacy guarantee P4P provides is *differential privacy*, a notion of privacy introduced in [25], further refined by [24, 23], and adopted by many latest works such as [9, 43, 42, 8, 41]. Differential privacy models the leakage caused by releasing some function computed over a data set. It captures the intuition that the function is private if the risk to one's privacy does not substantially increase as a result of participating in the data set. Formally it is defined as:

Definition 1 (Differential Privacy [25, 24]) $\forall \epsilon, \delta \geq 0$, *an algorithm* $\mathcal{A}$ *gives* (ϵ, δ)*-differential privacy if for all* $S \subseteq Range(\mathcal{A})$, *for all data sets* D, D' *such that* D *and* D' *differ by a single record*

$$\Pr[\mathcal{A}(D) \in S] \leq \exp(\epsilon) \Pr[\mathcal{A}(D') \in S] + \delta$$

There are several solutions achieving differential privacy for some machine learning and data mining algorithms (e.g., [24, 9, 43, 42, 8, 41]). Most require a trusted server hosting the entire data set. Our scheme removes such a requirement and also provides tools for handling a more adversarial setting where the data sources may be malicious. [4] is also a distributed and differentially private scheme for binary sum functions but it is only secure in a semi-honest model.

Differential privacy is widely used in the database privacy community to model the leakage caused by answering queries. P4P's reliance on differential privacy is as follows: During the computation, certain aggregate information (including the final result) is released (other information is kept hidden using cryptographic means). This is also modeled as query responses computed over the entire data set. Measuring such leakage against differential privacy allows us to have a rigorous formulation of the risk each individual user faces. By tuning the parameters ϵ and δ we can control such risk and obtain a system with adequate privacy as well as high efficiency. Another nice property of using differential privacy is that it can cover the final results (in contrast secure MPC in cryptography does not) therefore the protection is complete. Integrating differential privacy into secure computation has been accepted by the cryptography community [4] and our work can been seen as a concrete and highly efficient instantiation of such an approach to secure computation of some algorithms.

3 Design Considerations

Our design was motivated by careful evaluation of goals, available resources, and alternative solutions.

3.1 Design Goals

Our goal is to provide practical privacy solutions for some real-world applications. To this end, we identify three properties that are essential to a practical privacy solution:

1. **Provable Privacy**: Its privacy must be rigorously proven against well formulated privacy definitions.

2. **Efficiency and Scalability**: It must have adequate efficiency at reasonably large scale, which is an absolute necessity for many of today's data mining applications. The scale we are targeting is unprecedented: to support real-world application both the number of users and the number of data items per user are assumed to be in millions.

3. **Robustness**: It must be secure against realistic adversaries. Many computations either involve the participation of users, or collect data from them. Cheating of a small number of users is a realistic threat that the system must handle.

To the best of our knowledge, no existing works, or trivial composition of them, attain all three. Ours is the first, with open-source code, supporting all these properties.

3.2 Available Resources

During the past few years the landscape of large-scale distributed computing has changed dramatically. Many new resources and paradigms are available at very low cost and many computations that were infeasible at large scale in the past are now running routinely. One notable trend is the rapid growth of "cloud computing", which refers to the model where clients purchase computing cycles and/or storage from a third-party provider over the Internet. Vendors are sharing their infrastructures and allowing general users access to gigantic computing capability. Industrial giants such as Microsoft, IBM, Yahoo!, and Google are all key players in the game. Some of the cloud services (e.g., Amazon's Elastic Compute Cloud, http://aws.amazon.com/ec2.) are already available to general public at very cheap price.

The growth of cloud computing symbolizes the increased availability of large-scale computing power. We believe it is time to re-think the issue of privacy preserving data mining in light of such changes. There are several significant differences:

1. Could computing providers have very different incentives. Unlike traditional e-commerce vendors who are naturally interested in users data (e.g., purchase history), the cloud computing providers's commodity (CPU cycles and disk space) is *orthogonal* to users' computation. Providers do not benefit directly from knowing the data or computation results, other than ensuring that they are correct.

2. The traditional image of client-server paradigm has changed. In particular, the users have much more control over the data and the computation. In fact in many cases the cloud servers will be running code written by the customers. This is to be contrasted with traditional e-commere where there is a tremendous power imbalance between the service provider, who possesses all the information and controls what computation to perform, and the client users.

3. The servers are now clusters of hundreds or even thousands of machines capable of handling huge amount of data. They are not bottlenecks anymore.

Discrepancy of incentives and power imbalance have been identified as two major obstacles for the adoption of privacy technology by researchers examining privacy issues from legal and economic perspectives [26, 1]. Interestingly, both are greatly mitigated with the dawn of cloud computing. While traditional e-commerce vendors are reluctant to adopt privacy technologies, cloud providers would happily comply with customers instructions regarding what computation to perform. And once a treasure for the traditional e-commerce vendors, user data is now almost a burden for the cloud computing providers: storing the data not only costs disk space, but also may entail certain liability such as hosting illegal information. Some cloud providers may even choose not to store the data. For example, with Amazon's EC2 service, user data only persists during the computation.

We believe that cloud computing offers an extremely valuable opportunity for developing a new paradigm of practical privacy-preserving distributed computation: the existence of highly available, highly reputable, legally bounded service providers also provides a very important source of security. In particular, they make it realistic to treat some participants as *passive* adversaries. (The rests are still handled as active adversaries. The model is therefore a heterogenous one.) By tapping into this resource, we can build a heterogeneous system that can have privacy, scalability and robustness all at once.

3.3 The Alternatives

Existing privacy solutions for distributed data mining can be classified into two models: distributed and server-based. The former is represented by a large amount of work in the area of secure multiparty computation (MPC) in cryptography. The latter includes mostly homomorphic encryption-based schemes such as [11, 22, 51].

Generic MPC: MPC allows n players to compute a function over their collective data without compromising the privacy of their inputs or the correctness of the outputs even when some players are corrupted by the same adversary. The problem dates back to Yao [52] and Goldreich et al. [31], and has been extensively studied in cryptography [6, 2, 33]. Recent years see some significant improvement in efficiency. Some protocols achieve nearly optimal asymptotic complexity [3, 16] while some work in small field [12].

From practitioners' perspective, however, these generic MPC protocols are mostly of theoretical interest. Reducing asymptotic complexity does not automatically make the schemes practical. These schemes tend to be complex which imposes a huge barrier for developers not familiar with this area. Trying to support generic computation, most of them compile an algorithm into a (boolean or arithmetic) circuit. Not only the depth of such a circuit can be huge for complex algorithms, it is also very difficult, if not entirely impossible, to incorporate existing infrastructures and tools (e.g., ARPACK, LAPACK, MapReduce, etc.), into such computation. These tools are indispensable part of our daily computing life and symbolize the work of many talents over many years. Re-building production-ready implementations is costly and error-prone and generally not an option for most companies in our fast-pacing modern world.

Recently there are several systems that implemented some of the MPC protocols. While this reflects a plausible attempt to bridge the gap between theory and practice, unfortunately, performance-wise none of the systems came close to providing satisfactory solutions for most large-scale real-world applications. Table 1 shows some representative benchmarks obtained by these implementations. Using FairplayMP [5] as an example, adding two 64-bit integers is compiled into a circuit of 628 gates and 756 wires using its SFDL compiler. According to [5]'s benchmark, evaluating such a circuit between two players takes about 7 seconds. With this performance, adding 10^6 vectors of dimensionality 10^6 each, which constitutes one iteration in our framework, takes 7×10^{12} seconds, or 221,969 years.

ECC and a single server: It has been shown that conventional client-server paradigm can be augmented with homomorphic encryption to perform some computations with privacy (e.g., [11, 22, 51]). Still, such schemes are only marginally feasible for small to medium scale problems due to the need to perform at least linear number of large field operations even in purely semi-honest model. Using elliptic curve cryptography (ECC) can mitigate the problem as ECC can reduce the size of the cryptographic field (e.g., a 160-bit ECC key provides the same level of security as a 1024-bit RSA key). ECC cryptosystems such as [44] are $(+,+)$-homomorphic which is ideal for private computation. However, ECC point addition requires 1 field inversion and several field multiplications. The operation is still orders of magnitude slower than adding 64-bit or 32-bit integers directly. According to our benchmark, inversion and multiplication in a 160-bit field take 0.0224 and 0.001 milliseconds, respectively. Adding 1 million 10^6-element vectors takes 260 days.

Lesson learned: For large-scale problems, privacy and security must be added with negligible cost. In particular, those steps that dominate the computation should *not* be burdened with public-key cryptographic operations (even those "efficient" ones such as ECC) simply because they have to be performed so many times. This is the major principle that guides our design. In our scheme, the main computation is always performed in small field, while verifications are done via random projection techniques to reduce the number of cryptographic operations. As our experiments show, this approach is effective. When the number of cryptographic operations are insignificant, even using the traditional ElGamal encryption (or commitment) with 1024-bit key the performance is adequate for large scale problems.

4 P4P's Architecture

Our approach is called Peers for Privacy, or P4P. The name comes from the feature that, during the computation, certain *aggregate* information is released. This is a very important technique that allows the private protocol to have high efficiency. We show that publishing such aggregate information does not harm privacy: individual traits are masked out in the aggregates and releasing them is safe. In other words, peers data mutually protects each other within the aggregates.

Let $\kappa > 1$ be a small integer. We assume that there are κ servers belonging to different service providers (e.g., Amazon's EC2 service and Microsoft's Azure Services Platform, http://www.microsoft.com/azure/default.mspx). We define a *server* as all the computation units under the control of a single entity. It can be a cluster of thousands of machines so that it has the capability to support a large number of users.

Threat Model *Let $\alpha \in [0, 0.5)$ be the upper bound on the fraction of the dishonest users in the system.* [1] *Our scheme is robust against a computationally bounded adversary whose capability of corrupting parties is modeled as follows:*

1. *The adversary may actively corrupt at most $\lfloor \alpha n \rfloor$ users where n is the number of users.*

2. *In addition to 1, we also allow the same adversary to passively corrupt $\kappa - 1$ server(s).*

Table 1: Performance Comparison of Existing MPC Implementations

System	Adversary Model	Benchmark	Run Time (sec)
Fairplay [40]	Semi-honest	Billionaires	1.25
FairplayMP [5]	Semi-honest	Binary Tree Circuit (512 Gates)	6.25
PSSW [46]	Semi-honest	AES Encryption of 128-bit block	7
LPS [39]	Malicious	16-bit Integer Comparison	135

This model was proposed in [21] and is a special case of the general adversary structure introduced in [28, 34, 35] in that some of the participants are actively corrupted while some others are passively corrupted by the same adversary *at the same time*. Our model does not satisfy the feasibility requirements of [34, 35] and [28]. We avoid the impossibility by considering addition only computation.

The model models realistic threats in our target applications. In general, users are not trustworthy. Some may be incentivized to bias the computation, some may have their machines corrupted. So we model them as active adversaries and our protocol ensures that active cheating of a small number of users will not exert large influence on the computation. This greatly improves over existing privacy-preserving data mining solutions (e.g. [38, 51, 49]) and many current MPC implementations which handle only purely passive adversary. The servers, on the other hand, are selling CPU cycles and disk space, something that is not related to user's computation or data. Deviating from the protocol causes them penalty (e.g., loss of revenue for incorrect results) but little benefit. Their threat is therefore passive. (Corrupted servers are allowed to share data with corrupted users)

Treating "large institutional" servers as semi-honest, non-colluding has already been established by various previous work [38, 51, 50, 49]. However, in most of the models, the servers are not only semi-honest, but also "trusted", in that some user data is exposed to at least one of the servers (vertical or horizontal partitioned database). Our model does not have this type of trust requirement as each server only holds a random share of the user data. This further reduces the server's incentive to try to benefit from user data (e.g., reselling it) because the information it has are just random numbers without the other shares. A compromise requires the collusion of *all* servers which is a much more difficult endeavor. This also works for the servers' benefit: they are relieved of the liability of hosting secret or illegal computation, a problem that someone [18] envisions cloud providers will be facing.

5 The P4P Framework

Let n be the number of users. Let ϕ be a small (e.g., 32- or 64-bit) integer. We write $\mathbb{Z}_\phi$ for the additive group of integers modulo ϕ. Let a_i be private user data for user i and I be public information. Both can be matrices of arbitrary dimensions with elements from arbitrary domains. Our scheme supports any iterative algorithms whose $(t+1)$-th update can be expressed as

$$I^{(t+1)} = f(\sum_{i=1}^{n} d_i^{(t)}, I^{(t)})$$

where $d_i^{(t)} = g(a_i, I^{(t)}) \in \mathbb{Z}_\phi^m$ is an m-dimensional data vector for user i computed locally. Typical values for both m and n can range from thousands to millions. Both f and g are in general non-linear. In the SVD example that we will present, $I^{(t)}$ is the vector returned by ARPACK, g is matrix-vector product, and f is the internal computation performed by ARPACK.

This simple primitive is a surprisingly powerful model supporting a large number of popular data mining and machine learning algorithms, including Linear Regression, Naive Bayes, PCA, k-means, ID3, and EM etc., as has been demonstrated by numerous previous work such as [11, 13, 17, 10, 22]. It has been shown that all algorithms in the statistical query model [36] can be expressed in this form. Moreover, addition is extremely easy to parallelize so aggregating a large amount of numbers on a cluster is straightforward.

5.1 Private Computation

In the following we only describe the protocol for one iteration since the entire algorithm is simply a sequential invocations of the same protocol. The superscript is thus dropped from the notation. For simplicity, we only describe the protocol for the case of $\kappa = 2$. It is straightforward to extend it to support $\kappa > 2$ servers (by substituting the $(2, 2)$-threshold secret sharing scheme with a (κ, κ) one). Using more servers strengthens the privacy protection but also incurs additional cost. We do not expect the scheme will be used with a large number of servers. This arrangement simplifies matters such as synchronization and agreement. Let S_1 and S_2 denote the two servers. Leaving out validity and consistency check which will be illustrated using the SVD example, the basic computation is carried out as follows:

1. User i generates a uniformly random vector $u_i \in \mathbb{Z}_\phi^m$ and computes $v_i = d_i - u_i \mod \phi$. She sends u_i to S_1 and v_i to S_2.
2. S_1 computes $\mu = \sum_{i=1}^n u_i \mod \phi$ and S_2 computes $\nu = \sum_{i=1}^n v_i \mod \phi$. S_2 sends ν to S_1.
3. S_1 updates I with $f((\mu + \nu) \mod \phi, I)$.

It is straightforward to verify that if both servers follow the protocol, then the final result is indeed the sum of the user data vectors mod ϕ. This result will be correct if every user's vector lies in the specified bounds for L2-norm, which is checked by the ZKP in [21].

5.2 Provable Privacy

Theorem 1 *P4P's computation protocol leaks no information beyond the intermediate and final aggregates, if no more than* $\kappa - 1$ *servers are corrupted.*

The proof follows easily the fact that both the secret sharing scheme (for the computation) and the Pedersen commitment scheme [45, 15] used in the ZK protocols are information-theoretic private, as the adversary's view of the protocol is uniformly random and contains no information about user data. We refer the readers to [30] for details and formal definition of information-theoretic privacy.

As for the leakage caused by the released sums, first, for SVD, and some other algorithms, we are able to show the sums can be approximated from the final result so they do not leak more information. For general computation, we draw on the works on differential privacy. [20] has shown that, using well-established results from statistical database privacy [7, 19, 25], under certain conditions, releasing the vector sums still maintains differential privacy.

In some situations verifying the conditions of [20] privately is non-trivial but this difficulty is not essential in our scheme. There are well-established results that prove that differential privacy, as well as adequate accuracy, can be maintained as long as the sums are perturbed by independent noise with variance calibrated to the number of iterations and the sensitivity of the function [7, 19, 25]. In our settings, it is trivial to introduce noise into our framework – each server, which is semi-honest, can add the appropriate amount of noise to their partial sums after all the vectors from users are aggregated. Calibrating noise level is also easy: All one needs are the parameters ϵ, δ, the total number of queries (mT in our case where T is the number of iterations), and the sensitivity of the function f, which is summation in our case, defined as [25]:

$$S(f) = \max_{D,D'} \|f(D) - f(D')\|_1$$

where D and D' are two data sets differing by a single record and $\|\cdot\|_1$ denotes the L1-norm of a vector. Cauchy's Inequality states that

$$(\sum_{i=1}^m x_i y_i)^2 \le (\sum_{i=1}^m x_i^2)(\sum_{i=1}^m y_i^2)$$

For a user vector $a = [a_1, \dots, a_m]$, let $x_i = |a_i|$, $y_i = 1$, we have

$$\|a\|_1^2 = (\sum_{i=1}^m |a_i|)^2 \le (\sum_{i=1}^m a_i^2)m = \|a\|_2^2 m$$

Since our framework bounds the L2-norm of a user's vector to below L, this means the sensitivity of the computation is at most $\sqrt{m}TL$.

Note that the perturbation does not interfere with our ZK verification protocols in any way, as the latter is performed between each user and the servers on the *original* data. Whether noise is necessary or not is dependent on the algorithm. For simplicity we will not describe the noise process in our protocol explicitly. We stress again that the SVD example we will present next does *not* need any noise at all. See section 6.6.

6 Private Large-Scale SVD

In the following we use a concrete example, a private SVD scheme, to demonstrate how the P4P framework can be used to support private computation of popular algorithms.

6.1 Basics

Recall that for a matrix $A \in \mathbb{R}^{n \times m}$, there exists a factorization of the form

$$A = U \Sigma V^T \tag{1}$$

where U and V are $n \times n$ and $m \times m$, respectively, and both have orthonormal columns. Σ is $n \times m$ with nonnegative real numbers on the diagonal sorted in descending order and zeros off the diagonal. Such a factorization is called a singular value decomposition of A. The diagonal entries of Σ are called the the singular values of A. The columns of U and V are left- resp. right-singular vectors for the corresponding singular values.

SVD is a very powerful technique that forms the core of many data mining and machine learning algorithms. Let $r = rank(A)$ and u_i, v_i be the column vectors of U and V, respectively. Equation 1 can be rewritten as

$A = U\Sigma V^T = \sum_{i=1}^r \sigma_i u_i v_i^T$ where σ_i is the ith singular value of A. Let $k \le r$ be an integer parameter, we can approximate A by $A_k = U_k \Sigma_k V_k^T = \sum_{i=1}^k \sigma_i u_i v_i^T$. It is known that of all rank-k approximations, A_k is optimal in Frobenius norm sense. The k columns of U_k (resp. V_k) give the optimal k-dimensional approximation to the columnspace (resp. rowspace) of A. This dimensionality reduction preserves the structure of original data while considers only essential components of the matrix. It usually filters out noise and improves the performance of data mining tasks.

Our implementation uses a popular eigensolver, ARPACK [37] (ARnoldi PACKage), and its parallel version PARPACK. ARPACK consists of a collection of Fortran77 subroutines for solving large-scale eigenvalue problems. The package implements the Implicitly Restarted Arnoldi Method (IRAM) and allows one to compute a few, say k, eigenvalues and eigenvectors with user specified features such as those of largest magnitude. Its storage complexity is $nO(k) + O(k^2)$ where n is the size of the matrix. ARPACK is a freely-available yet powerful tool. It is best suited for applications whose matrices are either sparse or not explicitly available: it only requires the user code to perform some "action" on a vector, supplied by the solver, at every IRAM iteration. This action is simply matrix-vector product in our case. Such a reverse communication interface works seamlessly with P4P's aggregation protocol.

6.2 The Private SVD Scheme

In our setting the rows of A are distributed across all users. We use $A_{i*} \in \mathbb{R}^m$ to denote the m-dimensional row vector owned by user i. From equation 1, and the fact that both U and V are orthonormal, it is clear that $A^TA = V\Sigma^2V^T$ which implies that $A^TAV = V\Sigma^2$. A straightforward way is then to compute $A^TA = \sum_{i=1}^n A_{i*}^T A_{i*}$ and solve for the eigenpairs of A^TA. The aggregate can be computed using our private vector addition framework. This is a distributed version of the method proposed in [7] and does not require the consistency protocol that we will introduce later. Unfortunately, this approach is not scalable as the cost for each user is $O(m^2)$. Suppose $m = 10^6$, and each element is a 64-bit integer, then $A_{i*}^T A_{i*}$ is 8×10^{12} bytes, or about 8 TB. The communication cost for each user is then 16 TB (she must send shares to two servers). This is a huge overhead, both communication- and computation-wise. Usually the data is very sparse and it is a common practice to reduce cost by utilizing the sparsity. Unfortunately, sparsity does not help in a privacy-respecting application: revealing which elements are non-zero is a huge privacy breach and the users are forced to use the dense format. We propose the following scheme which

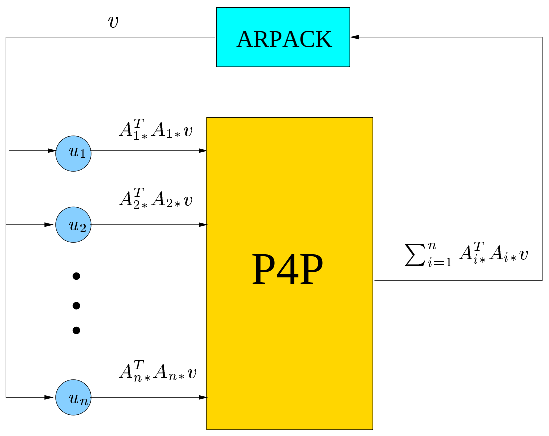

Figure 1: Private SVD with P4P

reduces the cost dramatically. We involve the users in the iteration and the total communication (and computation) cost per iteration is only $O(m)$ for each user. The number of iterations required ranges from tens to over a thousand. This translates to a maximum of a few GB data communicated for each user for the *entire* protocol which is much more manageable.

One server, say S_1, will host an ARPACK engine and interact with its reverse communication interface. In our case, since A^TA is symmetric, the server will use `dsaupd`, ARPACK's double precision routine for symmetric problems, and asks for k largest (in magnitude) eigenvalues. At each iteration, `dsaupd` returns a vector v to the server code and asks for the matrix-vector product A^TAv. Notice that

$$A^TAv = \sum_{i=1}^n A_{i*}^T A_{i*} v$$

Each term in the summation is computable by each user locally in $O(m)$ time (by computing the inner product $A_{i*} \cdot v$ first) and the result is an m-vector. The vector can then be input to the P4P computation which aggregates them across all users privately. The aggregate is the matrix-vector product which can be returned to ARPACK for another iteration. This process is illustrated in figure 1.

The above method is known to have sensitivity problem, i.e., a small perturbation to the input could cause large error in the output. In particular, the error is $O(\|A\|^2/\sigma_k)$ [48]. Fortunately, most applications (e.g., PCA) only need the k largest singular values (and their singular vectors). It is usually not a problem for those applications since for the principal components $O(\|A\|^2/\sigma_k)$ is small. There is no noticeable inaccuracy in our test applications (latent semantic analysis for document retrieval). For general problems the stable way is to compute the eigenpairs of the matrix

$$H = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$$

It is straightforward to adopt our private vector addition framework to compute matrix-vector product with H. For simplicity we will not elaborate on this.

6.3 Enforcing Data Consistency

During the iteration, user i should input $d_i = A_{i*}^T A_{i*} v$. However, a cheating user could input something completely different. This threat is different from inputting bogus (but in the allowable range) data at the beginning (and using it consistently throughout the iterations). The latter only introduces noise to the computation but generally does not affect the convergence. The L2-norm ZKP introduced in [21], which verifies that the L-2 norm of a user's vector is bounded by a public constant, is effective in bounding the noise but does not help in enforcing consistency. The former, on the other hand, may cause the computation not to converge at all. This generally is a problem for iterative algorithms and is more than simply testing the equality of vectors: The task is complicated by the local function that each user uses to evaluate on her data, i.e., she is not simply inputting her private data vector, but some (possibly non-linear) function of it. In the case of SVD, the system needs to ensure that user i uses the same A_{i*} (to compute $d_i = A_{i*}^T A_{i*} v$) in all the iterations, not that she inputs the same vector.

We provide a novel zero-knowledge tool that ensures that the correct data is used. The protocol is probabilistic and relies on random projection. That is, the user is asked to project her original vector and her result of the current round onto some random direction. It then tests the relation of the two projections. We will show that this method catches cheating with high probability but only involves very few expensive large field operations.

6.3.1 Tools

The consistency protocol uses some standard cryptographic primitives. Detailed construction and proofs can be found in [45, 15, 11]. We summarize only their key properties here. All values used in these primitives lie in the multiplicative group $\mathbb{Z}_q^*$, or in the additive group of exponents for this group, where q is a 1024 or 2048-bit prime. They rely on RSA or discrete log functions for cryptographic protection of information.

- **Homomorphic commitment**: A homomorphic commitment to an integer a with randomness r is written as $\mathcal{C}(a, r)$. It is homomorphic in the sense that $\mathcal{C}(a, r)\mathcal{C}(b, s) = \mathcal{C}(a + b, r + s)$. It is infeasible to determine a given $\mathcal{C}(a, r)$. We say that a prover "opens" the commitment if it reveals a and r.
- **ZKP of knowledge**: A prover who knows a and r (i.e., who knows how to open $\mathcal{A} = \mathcal{C}(a, r)$) can demonstrate that it has this knowledge to a verifier who knows only the commitment $\mathcal{A}$. The proof reveals nothing about a or r.
- **ZKP for equivalence**: Let $\mathcal{A} = \mathcal{C}(a, r)$ and $\mathcal{B} = \mathcal{C}(a, s)$ be two commitments to the same value a. A prover who knows how to open $\mathcal{A}$ and $\mathcal{B}$ can demonstrate to a verifier in zero knowledge that they commit to the same value.
- **ZKP for product**: Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be commitments to a, b, c respectively, where $c = ab$. A prover who knows how to open $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ can prove in zero knowledge to a verifier who has only the commitments that the relationship $c = ab$ holds among the values they commit to. If say a is made public, this primitive can be used to prove that $\mathcal{C}$ encodes a number that is multiple of a.

6.3.2 The Protocol

The consistency check protocol is summarized in the following. Since the protocol is identical for all users, we drop the user subscript for the rest of the paper whenever there is no confusion. Let $a \in \mathbb{Z}_\phi^m$ be a user's original vector (i.e., her row in the matrix A). The correct user input to this round should be $d = a^T a v$. For two vectors x and y, we use $x \cdot y$ to denote their scalar product.

1. After the user inputs her vector d, in the form of two random vectors $d^{(1)}$ and $d^{(2)}$ in $\mathbb{Z}_\phi^m$, one to each server, s.t. $d = d^{(1)} + d^{(2)} \mod \phi$, S_1 broadcasts a random number r. Using r as the seed and a public PRG (pseudo-random generator), all players generate a random vector $c \in_R \mathbb{Z}_\phi^m$.
2. For $j \in \{1, 2\}$, the user computes $x^{(j)} = c \cdot a^{(j)} \mod \phi$, $y^{(j)} = a^{(j)} \cdot v \mod \phi$. Let $x = x^{(1)} + x^{(2)}$, $y = y^{(1)} + y^{(2)}$, $z = xy$. Let $w = (c \cdot a)(a \cdot v) - xy$. The user commits $\mathcal{X}^{(j)}$ to $x^{(j)}$, $\mathcal{Y}^{(j)}$ to $y^{(j)}$, $\mathcal{Z}$ to z, and $\mathcal{W}$ to w. She also construct two ZKPs: (1) $\mathcal{W}$ encodes a number that is multiple of ϕ. (2) $\mathcal{Z}$ encodes a number that is the product of the two numbers encoded in $\mathcal{X}$ and $\mathcal{Y}$ where $\mathcal{X} = \mathcal{X}^{(1)}\mathcal{X}^{(2)}$ and $\mathcal{Y} = \mathcal{Y}^{(1)}\mathcal{Y}^{(2)}$. She sends all commitments and ZKPs to both servers.
3. The user opens $\mathcal{X}^{(j)}$ and $\mathcal{Y}^{(j)}$ to S_j who verifies that both are computed correctly. Both servers verify the ZKPs. If any of them fails, the user is marked as FAIL and the servers terminate the protocol with her.

4. For $j \in \{1, 2\}$, the user computes $\tilde{z}^{(j)} = c \cdot d^{(j)} \mod \phi$, $\tilde{z} = \tilde{z}^{(1)} + \tilde{z}^{(2)}$ and $\tilde{w} = c \cdot d - \tilde{z}$. She commits $\tilde{\mathcal{Z}}^{(1)}$ to $\tilde{z}^{(1)}$, $\tilde{\mathcal{Z}}^{(2)}$ to $\tilde{z}^{(2)}$, and $\tilde{\mathcal{W}}$ to $\tilde{w}$. She constructs the following two ZKPs: (1) $\tilde{\mathcal{W}}$ encodes a number that is multiple of ϕ and (2)$\tilde{\mathcal{Z}}\tilde{\mathcal{W}}$ and $\mathcal{Z}\mathcal{W}$ encode the same value. She sends all the commitments and ZKPs to both servers.

5. The user opens $\tilde{\mathcal{Z}}^{(j)}$ to S_j who verifies that it is computed correctly. Both servers verify the two ZKPs. They mark the user as FAIL if any of the verifications fails and terminate the protocol with her.

6. Both servers output PASS.

Group Sizes
There are three groups/fields involved in the protocol: the large, multiplicative group $\mathbb{Z}_q^*$ used for commitments and ZKPs, the "small" group $\mathbb{Z}_\phi$ used for additive secret-sharing, and the group of all integers. All the commitments such as $\mathcal{X}^{(j)}$ and $\mathcal{Y}^{(j)}$ are computed in $\mathbb{Z}_q^*$ so standard cryptographic tools can be used. The inputs to the commitments, which can be user's data or some intermediate results, are either in $\mathbb{Z}_\phi$ or in the integer group (without bounding their values). Restricting commitment inputs to small field/group does not compromise the security of the scheme since the outputs are still in the large field. Using Pederson's commitment as an example, the hiding property is guaranteed by the random numbers that are generated in the *large* field for each commitments. And breaking the binding property is still equivalent to solving the discrete logarithm problem in $\mathbb{Z}_q^*$. See [45].

The protocol makes it explicit which group a number is in using the $\mod \phi$ operator (i.e., $x = g(y) \mod \phi$ restricts x to be in $\mathbb{Z}_\phi$ while $x = g(y)$ means x can be in the whole integer range). The protocol assumes that $q \gg \phi$. This ensures that the numbers that are in the integer group (x, y, z, w in step 2 and $\tilde{z}$ and $\tilde{w}$ in step 4) are much less than q to avoid modular reduction when their commitments are produced. This is true for most realistic deployment, since ϕ is typically 64 bits or less while q is 1024 bits or more. Theorem 2 proves that the transition from $\mathbb{Z}_\phi$ to integer fields and $\mathbb{Z}_q^*$ only causes the protocol to fail with extremely low probability:

Theorem 2 *Let O be the output of the Consistency Check protocol. Then*

$$\Pr(O = \textit{PASS} | d = a^T av) = 1$$

and

$$\Pr(O = \textit{PASS} | d \neq a^T av) \leq \frac{1}{\phi}$$

Furthermore, the protocol is zero-knowledge.

Proof If computed correctly, both w and $\tilde{w}$ are multiples of ϕ due to modular reduction. Because of homomorphism, the equivalence ZKP that $\tilde{\mathcal{Z}}\tilde{\mathcal{W}}$ and $\mathcal{Z}\mathcal{W}$ encode the same value is to verify that $c \cdot d = c \cdot (a^T av)$.

Completeness: If the user performs the computation correctly, she should input $d = a^T av$ into this round of computation. All the verifications should pass. The protocol outputs PASS with probability 1.

Soundness: Suppose $d \neq aa^T v$. The user is forced to compute the commitments $\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \mathcal{Y}^{(1)}, \mathcal{Y}^{(2)}$, and $\tilde{\mathcal{Z}}^{(1)}, \tilde{\mathcal{Z}}^{(2)}$ faithfully since she has to open them to at least to one of the servers. The product ZKP at step 2 forces the number encoded in $\mathcal{Z}$ to be xy which differs from $c \cdot (a^T av)$ by w. Due to homomorphism, at step 4, $\tilde{\mathcal{Z}}$ encodes a number that differs from $c \cdot d$ by $\tilde{w}$. The user could cheat by lying about w or $\tilde{w}$, i.e., she could encode some other values in $\mathcal{W}$ and $\tilde{\mathcal{W}}$ to adjust for the difference between $c \cdot d$ and $c \cdot (a^T av)$, hoping to pass the equivalence ZKP. However, assuming the soundness of the ZKPs used, the protocol forces both to be multiple of ϕ (steps 2 and 4), so she could succeed only when the difference between $c \cdot d$, which she actually inputs to this round, and $c \cdot (a^T av)$, which she should input, is some multiple of ϕ. Since c is made known to her *after* she inputs d, the two numbers are totally unpredictable and random to her. The probability that $c \cdot d - c \cdot (a^T av)$ is a multiple of ϕ is only $1/\phi$ which is the probability of her success.

Finally, the protocol consists of a sequential invocation of some well-established ZKPs. By the sequential composition theorem of [32], the whole protocol is also zero-knowledge.

As a side note, all the ZKPs can be made non-interactive using the Fiat-Shamir paradigm [27]. The user could upload her data in a batch without further interaction. This makes it easier to deploy the scheme. It is also much more light-weight than the L2-norm ZKP [21]: the number of large field operations is *constant*, as opposed to $O(\log m)$ in the L2-norm ZKP. The private SVD computation thus involves only one L2-norm ZKP at first round, and one light verification for each of the subsequent rounds.

6.4 Dealing with Real Numbers

In their simplest forms, the cryptographic tools only support computation on integers. In most domains, however, applications typically have to handle real numbers. In the case of SVD, even if the original input matrix contains only integer entries, it is likely that real numbers appear in the intermediate (e.g., the vectors returned by ARPACK) and the final results.

Because of the linearity of the P4P computation, we can use a simple linear digitization scheme to convert between real numbers in the application domain and Z_ϕ, P4P's integer field. Let $R > 0$ be the bound of the maximum absolute value application data can take, i.e., all numbers produced by the application are between $[-R, R]$. The integer field provides $|\phi|$ bits resolution. This means the maximum quantization error for one variable is $R/\phi = 2^{|R|-|\phi|}$. Summing across all n users, the worst case absolute error is bounded by $n2^{|R|-|\phi|}$. In practice $|\phi|$ can be 64, and $|R|$ can be around e.g., 20 (this gives a range of $[-2^{20}, 2^{20}]$). With $n = 10^6$, this gives a maximum absolute error of under 1 over a million.

6.5 The Protocol

Let Q be the set of qualified users initialized to the set of all users. The entire private SVD method is summarized as follows:

1. **Input** The user first provides an L2-norm ZKP [21] on a with a bound L, i.e., she submits a ZKP that $\|a\|_2 < L$. This step also forces the user to commit to the vector a. Specifically, at the end of this step, S_1 and S_2 have $a^{(1)} \in \mathbb{Z}_\phi$ and $a^{(2)} \in \mathbb{Z}_\phi$, respectively, such that $a = a^{(1)} + a^{(2)} \mod \phi$. Users who fail this ZKP are excluded from subsequent computation.

2. Repeat the following steps until the ARPACK routine indicates convergence or stops after certain number of iterations:

   (a) **Consistency Check** When `dsaupd` returns control to S_1 with a vector, the server converts the vector to $v \in \mathbb{Z}_\phi^m$ and sends it to all users. The servers execute the consistency check protocol for each user.

   (b) **Aggregate** For any users who are marked as FAIL, or fail to respond, the servers simply ignore their data and exclude them from subsequent computation. Q is updated accordingly. For this round they compute $s = \sum_{i \in Q} d_i$ and S_1 returns it as the matrix-vector product to `dsaupd` which runs another iteration.

3. **Output** S_1 outputs

$$\begin{aligned}\Sigma_k &= diag(\sigma_1, \sigma_2, \ldots, \sigma_k) \in \mathbb{R}^{k\times k} \\ V_k &= [v_1, v_2, \ldots, v_k,] \in \mathbb{R}^{m\times k}\end{aligned}$$

   with $\sigma_i = \sqrt{\lambda_i}$ where λ_i is the ith eigenvalue and v_i the corresponding eigenvector computed by ARPACK, $i = 1, \ldots, k$, and $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_k$.

For accuracy of the result produced by this protocol in the presence of *actively* cheating users, we have

Theorem 3 *Let n_c be the number of cheating users. We use $\tilde{\cdot}$ to denote perturbed quantity and σ_i the i-th singular value of matrix A. Assuming that honest users vector L2-norms are uniformly random in $[0, L)$ and $n_c \ll n$, then*

$$\sqrt{\frac{\sum_i(\tilde{\sigma}_i - \sigma_i)^2}{\sum_i \sigma_i^2}} < 2\sqrt{\frac{n_c}{n}}$$

Proof The classic Weyl and Mirsky theorems [47] bound the perturbation to A's singular values in terms of the Frobenius norm $\|\cdot\|_F$ of $E := A - \tilde{A}$:

$$\sqrt{\sum_i(\tilde{\sigma}_i - \sigma_i)^2} \leq \|E\|_F$$

In our case each row a_i of A is held by a user, we have

$$\|E\|_F = \sqrt{\sum_{i=1}^{n} \|\tilde{a}_i - a_i\|_2^2}$$

Since the protocol ensures that $\|a_i\|_2 < L$ for all users,

$$\sqrt{\sum_i(\tilde{\sigma}_i - \sigma_i)^2} \leq \sqrt{\sum_{i=1}^{n} \|\tilde{a}_i - a_i\|_2^2} < \sqrt{n_c}L$$

Let $\xi = \sqrt{\sum_i(\tilde{\sigma}_i - \sigma_i)^2}/\sqrt{\sum_i \sigma_i^2}$, and assuming that honest users vector L2-norms are uniformly random in $[0, L)$ and $n_c \ll n$, then

$$\xi = \frac{\sqrt{\sum_i(\tilde{\sigma}_i - \sigma_i)^2}}{\|A\|_F} < \frac{\sqrt{n_c}L}{0.5\sqrt{(n-n_c)}L} \approx 2\sqrt{\frac{n_c}{n}}$$

The scheme is also quite robust against users failures. During our tests reported in section 7, we simulated a fraction of random users "dropping out" of each iteration. Even when up to 50% of the users dropped, for all our test sets, the computation still converged without noticeable loss of accuracy, measured by residual error (see section 7.1) using the final matrix with failed users data ignored. This allows us to handle malicious users who actively try to disrupt the computation and those who fail to response due to technical problems (e.g., network failure) in a uniform way.

6.6 Privacy Analysis

Note that the protocol does not compute U_k. This is intentional. U_k contains information about user data: the i^{th} row of U_k encodes user i's data in the k-dimensional subspace and should not be revealed at all in a privacy-respecting application. V_k, on the other hand, encodes "item" data in the k-dimensional subspace (e.g., if A is a user-by-movie rating matrix, the items will be movies).

In most applications the desired information can be computed from the singular values (Σ_k) and the right singular vectors (V_k^T) (e.g., [11])

At each iteration, the protocol reveals the matrix-vector product $A^T Av$ for some vectors v. This is not a problem because the final results Σ_k and V_k^T already give an approximation of $A^T A$ ($A^T A = V\Sigma^2 V^T$). A simulator with the final results can approximate the intermediate sums. Therefore the intermediate aggregates do not reveal more information.

7 Implementation and Evaluation

The P4P framework, including the SVD protocol, has been implemented in Java using JNI and a NativeBigInteger implementation from I2P (http://www.i2p2.de/). We run several experiments. The server is a 2.50GHz Xeon E5420 with 32GB memory, the clients are 2.00GHz Xeon E5405 with 800 MB memory allocated to the tests. In all the experiments, ϕ is set to be a 62-bit integer and q 1024-bit.

We evaluated our implementation on three data sets: the Enron Email Data set [14], EachMovie (EM), and a randomly generated dense matrix (RAND). The Enron corpus contains email data from 150 users, spanning a period of about 5 years (Jan. 1998 to Dec 2002). Our test was run on the social graph defined by the email communications. The graph is represented as a 150×150 matrix A with $A(i, j)$ being the number of emails sent by user i to user j. EachMovie is a well-known test data set for collaborative filtering. It comprises ratings of 1648 movies by 74424 users. Each rating is a number in the range $[0, 1]$. Both the Enron and EachMovie data sets are very sparse, with densities 0.0736 and 0.0229, respectively. To test the performance of our protocol on dense matrices, we generated randomly a 2000×2000 matrix with entries chosen in the range $[-2^{20}, 2^{20}]$.

7.1 Precision and Round Complexity

We measured two quantities: N, the number of IRAM iterations until ARPACK indicates convergence, and ϵ, the relative error. N is the number of matrix-vector computation that was required for the ARPACK to converge. It is also the number of times P4P aggregation is invoked. The error ϵ measures the maximum relative residual norm among all eigenpairs computed:

$$\epsilon = \max_{i=1,\ldots,k} \frac{\|A^T Av_i - \lambda_i v_i\|_2}{\|v_i\|_2}$$

Table 2 summarizes the results. In all these tests, we used machine precision as the tolerance input to ARPACK. The accuracy we obtained is very good: ϵ remains very small for all tests (10^{-12} to 10^{-8}). In terms of round complexity, N ranges from under 100 to a few hundreds. For comparison, we also measured the number of iterations required by ARPACK when we perform the matrix-vector multiplication directly without the P4P aggregation. In all experiments, we found no difference in N between this direct method and our private implementation.

7.2 Performance

We measured both running time and communication cost of our scheme. We focused on server load since each user only needs to handle her own data so is not a bottleneck. We first present the case with $\kappa = 2$ servers. We measured the work on the server hosting the ARPACK engine since it shares more load.

First, the implementation confirmed our observations about the difference in costs for manipulating large and small integers. With 1024-bit key length, one exponentiation within the multiplicative group $\mathbb{Z}_q^*$ takes 5.86 milliseconds. Addition and multiplication of two numbers, also within the group, take 0.024 and 0.062 milliseconds, respectively. In contrast, adding two 64-bit integers, which is the basic operations P4P framework performs, needs only 2.7×10^{-6} milliseconds. The product ZKP takes 35.7 ms verifier time and 24.3 ms prover time. The equivalence ZKP takes no time since it is simply revealing the difference of the two random numbers used in the commitments [45]. For each consistency check, the user needs to compute 9 commitments, 3 product ZKPs, 1 equivalence ZKP and 4 large integer multiplications. The total cost is 178.63 milliseconds for each user. For every user, each server needs to spend 212.83 milliseconds on verification.

For our test data sets, it takes 74.73 seconds of server time to validate and aggregate all 150 Enron users data on a *single* machine (each user needs to spend 726 milliseconds to prepare the zero-knowledge proofs). This translates into a total of 5000 seconds or 83 minutes spent on private P4P aggregation to compute $k = 10$ singular-pairs. To compute the same number of singular pairs for EachMovie, aggregating all users data takes about 6 hours (again on a single machine) and the total time for 70 rounds is 420 hours. Note that the total includes *both* verification and computation so it is the cost of a complete run. The server load appears large but actually is very inexpensive. The aggregation process is trivially parallelizable and using a cluster of, say 200 nodes, will reduce the running time to about 2 hours. This amounts to a very insignificant cost for most service providers: Using Amazon EC2's price as a benchmark, it costs \$0.80 per hour for 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each). Data transfer price is \$0.100 per GB. The total cost for comput-

Table 2: Round Complexity and Precision

Enron	k	10	20	30	40	50	60	70	80	90	100
	N	67	97	122	162	109	137	172	167	171	169
	$\epsilon(\times 10^{-8})$	0.00049	0.0021	0.0046	0.0084	0.0158	0.0452	0.121	0.266	0.520	1.232
EM	k	10	20	30	40	50	60	70	80	90	100
	N	70	140	254	222	276	371	322	356	434	508
	$\epsilon(\times 10^{-12})$	0.470	0.902	1.160	1.272	1.526	1.649	1.687	2.027	2.124	2.254
RAND	k	10	20	30	40	50	60	70	80	90	100
	N	304	404	450	480	550	700	770	720	810	800
	$\epsilon(\times 10^{-9})$	3.996	3.996	3.996	3.996	3.996	3.996	3.996	3.996	3.996	3.996

ing SVD for a system with 74424 users is merely about \$15, including data transfer and adjusted for difference in CPU performance between our experiments and EC2.

To compare with alternative solutions, we implemented a method based on homomorphic encryption which is a popular private data mining technique (see e.g., [11, 51]). We did not try other methods, such as the “add/subtract random” approach, with players adding their values to a running total, because they do not allow for verification of user data thus are insecure in our model. We tested both ElGamal and Paillier encryptions with the same security parameter as our P4P experiments (i.e., 1024-bit key). With the homomorphic encryption approach, it is almost impossible to execute the ZK verification (although there is a protocol [11]) as it takes hours to verify one user. So we only compared the time needed for computing the aggregates. Figure 2 shows the ratios of running time between homomorphic encryption and P4P for SVD on the three data sets. P4P is at least 8 orders of magnitude faster in all cases for both ElGamal and Paillier. And this translates to tens of millions of dollars of cost for the homomorphic encryption schemes if the computation is done using Amazon’s EC2 service not even counting data transfer expenses.

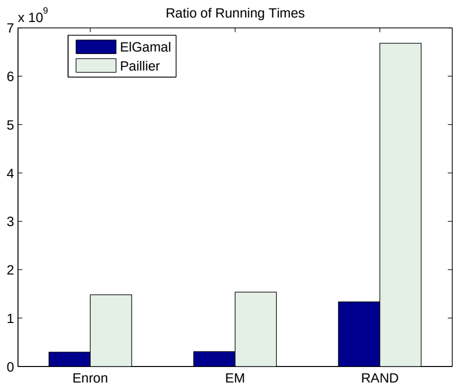

Figure 2: Running time ratios between homomorphic encryption based solutions and P4P.

The communication overhead is also very small since the protocol passes very few large integers. The extra communication per client for one L2-norm ZKP is under 50 kilobytes, and under 100 bytes for the consistency check, while other solutions require some hundreds of megabytes. This is significantly smaller than the size of an average web page. The additional workload for the server is less than serving an extra page to each user.

The case with $\kappa > 2$ servers: Although we do not expect the scheme to be deployed with a large number of servers, we provide some analysis here in case stronger protection is required. Each server’s work can be divided into two parts: processing clients and communicating with other servers. Most expensive interactions are with the clients (including verifying the ZKPs etc.), which can be performed on a single server and is independent of κ. The interaction among servers is simply data exchange and there is no complex computation involved.

Data exchange among the servers serves two purposes: reconstructing shared secrets when necessary (the final sum in the end of each iteration and the commitments during the verification) and reaching agreement regarding a user’s status (each server needs to verify that the user computes a share of the commitments correctly). And since each server is semi-honest, for the second part they only need pass the final conclusion, verification of the ZKPs can be done on only one of the servers.

For constructing the final sum, all servers must send their shares to the server hosting ARPACK. The later will receive a total of $8\kappa m$ bytes (assuming data is encoded using double precision) which is about 8κ MB if $m = 10^6$. For the consistency check, during each iteration, one server is selected as the “master”. All other servers sends their shares of the commitments to the master. This includes $3n$ large integers in $\mathbb{Z}_q$ (3 for each user) from each server. In addition, each non-master server also sends to the master an n-bit bitmap, encoding whether each user computes the commitments to the shares correctly. The master will reconstruct the complete commitments and verify the ZKPs. It then broadcasts an n-bit bitmap encoding whether each user passes the consistency check to all other servers. For the master, the total communication cost is receiving $3n(\kappa - 1)$ integers in $\mathbb{Z}_q$ and κn-bit strings and sending $(\kappa - 1)n$ bits. With $n = 10^6$ and $|q| = 1024$, these amount to

384 $(\kappa - 1)$ MB and approximately 0.1 $(\kappa - 1)$ MB, respectively. For other servers, the sending and receiving costs are approximately 384 MB and 0.1 MB, respectively. We believe such cost is practical for small κ (e.g., 3 or 4). Note that the master does not have to be collocated with the ARPACK engine so the servers can take turns to serve as the master to share the load.

As for the computation associated with using κ servers (the part that is independent of κ has been discussed earlier and omitted here), the master needs to perform $3n(\kappa - 1)$ multiplications in $\mathbb{Z}_q^*$. Using our benchmark, this amounts to $0.186(\kappa - 1)$ seconds for $n = 10^6$ users. Again we believe this is practical for small κ. The other servers do not need to do any extra work.

7.3 Scalability

We also experimented with a few very large matrices, with dimensionality ranging from tens of thousands to over a hundred million. They are document-term or user-query matrices that are used for latent semantic analysis. To facilitate the tests, we did not include the data verification ZKPs, as our previous benchmarks show they amount to an insignificant fraction of the cost. Due to space and resource limit we did not test how performance varies with dimensionality and other parameters. Rather, these results are meant to demonstrate the capability of our system, which we have shown to maintain privacy at very low cost, to handle large data sets at various configurations.

Table 3 summarizes some of the results. The running time measures the time of a complete run, i.e., from the start of the job till the results are safely written to disk. It includes both the computation time of the server (including the time spent on invoking the ARPACK engine) and the clients (which are running in parallel), and the communication time. In the table, frontend processors refer to the machines that interact with the users directly. Large-scale systems usually use multiple frontend machines, each serving a subset of the users. This is also a straightforward way to parallelize the aggregation process, i.e., each frontend machine receives data from a subset of users and aggregates them before forwarding to the server. On one hand, the more frontend machines the faster the sub-aggregates can be computed. On the other hand, the server's communication cost is linear in the number of frontend processors. The optimal solution must strike a balance between the two. Due to resource limitation, we were not able to use the optimal configuration for all our tests. The results are feasible even in these sub-optimal cases.

8 Conclusion

In this paper we present a new framework for privacy-preserving distributed data mining. Our protocol is based on secret sharing over *small* field, achieving orders of magnitude reduction in running time over alternative solutions with large-scale data. The framework also admits very efficient zero-knowledge tools that can be used to verify user data. They provide practical solutions for handling cheating users. P4P demonstrates that cryptographic building blocks can work harmoniously with existing tools, providing privacy without degrading their efficiency. Most components described in this paper have been implemented and the source code is available at http://bid.berkeley.edu/projects/p4p/. Our goal is to make it a useful tool for developers in data mining and others to build privacy preserving real-world applications.

References

[1] ALDERMAN, E., AND KENNEDY, C. *The Right to Privacy*. DIANE Publishing Co., 1995.

[2] BEAVER, D., AND GOLDWASSER, S. Multiparty computation with faulty majority. In *CRYPTO '89*.

[3] BEERLIOVÁ-TRUBÍNIOVÁ, Z., AND HIRT, M. Perfectly-secure mpc with linear communication complexity. In *TCC 2008* (2008), Springer-Verlag, pp. 213–230.

[4] BEIMEL, A., NISSIM1, K., AND OMRI, E. Distributed private data analysis: Simultaneously solving how and what. In *CRYPTO 2008*.

[5] BEN-DAVID, A., NISAN, N., AND PINKAS, B. Fairplaymp: a system for secure multi-party computation. In *CCS '08* (2008), ACM, pp. 257–266.

[6] BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC'88* (1988), ACM, pp. 1–10.

[7] BLUM, A., DWORK, C., MCSHERRY, F., AND NISSIM, K. Practical privacy: the SuLQ framework. In *PODS '05* (2005), ACM Press, pp. 128–138.

[8] BLUM, A., LIGETT, K., AND ROTH, A. A learning theory approach to non-interactive database privacy. In *STOC 08*.

[9] BOAZ BARAK, E. A. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS '07*.

[10] CANNY, J. Collaborative filtering with privacy via factor analysis. In *SIGIR '02*.

[11] CANNY, J. Collaborative filtering with privacy. In *IEEE Symposium on Security and Privacy* (2002), pp. 45–57.

[12] CHEN, H., AND CRAMER, R. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In *CRYPTO 2006*.

[13] CHU, C.-T., KIM, S. K., LIN, Y.-A., YU, Y., BRADSKI, G., NG, A. Y., AND OLUKOTUN, K. Map-reduce for machine learning on multicore. In *NIPS 2006* (2006).

[14] COHEN, W. W. Enron email dataset. http://www-2.cs.cmu.edu/˜enron/.

[15] CRAMER, R., AND DAMGÅRD, I. Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free? In *CRYPTO '98* (1998), Springer-Verlag.

Table 3: SVD of Large Matrices

n	m	k	No. Frontend Processors	Time (hours)	Iterations
100,443	176,573	200	32	1.4	1287
12,046,488	440,208	200	128	6.0	354
149,519,201	478,967	250	128	8.3	1579
37,389,030	366,881	300	128	9.1	1839
1,363,716	2,611,186	200	1	14.8	1260
33,193,487	1,949,789	200	128	28.0	1470

[16] DAMGÅRD, I., ISHAI, Y., KRØIGAARD, M., NIELSEN, J. B., AND SMITH, A. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO 2008* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 241–261.

[17] DAS, A. S., DATAR, M., GARG, A., AND RAJARAM, S. Google news personalization: scalable online collaborative filtering. In *WWW '07* (2007), ACM Press, pp. 271–280.

[18] DHANJANI, N. Amazon's elastic compute cloud [ec2]: Initial thoughts on security implications. http://www.dhanjani.com/archives/2008/04/.

[19] DINUR, I., AND NISSIM, K. Revealing information while preserving privacy. In *PODS '03* (2003), pp. 202–210.

[20] DUAN, Y. Privacy without noise. In *CIKM '09*.

[21] DUAN, Y., AND CANNY, J. Practical private computation and zero-knowledge tools for privacy-preserving distributed data mining. In *SDM '08* (2008).

[22] DUAN, Y., WANG, J., KAM, M., AND CANNY, J. A secure online algorithm for link analysis on weighted graph. In *Proc. of the Workshop on Link Analysis, Counterterrorism and Security, SDM 05*, pp. 71–81.

[23] DWORK, C. Ask a better question, get a better answer a new approach to private data analysis. In *ICDT 2007* (2007), Springer, pp. 18–27.

[24] DWORK, C., KENTHAPADI, K., MCSHERRY, F., MIRONOV, I., AND NAOR, M. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT 2006* (2006), Springer.

[25] DWORK, C., MCSHERRY, F., NISSIM, K., AND SMITH, A. Calibrating noise to sensitivity in private data analysis. In *TCC 2006* (2006), Springer, pp. 265–284.

[26] FEIGENBAUM, J., NISAN, N., RAMACHANDRAN, V., SAMI, R., AND SHENKER, S. Agents' privacy in distributed algorithmic mechanisms. In *Workshop on Economics and Information Securit* (Berkeley, CA, May 2002).

[27] FIAT, A., AND SHAMIR, A. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 86*.

[28] FITZI, M., HIRT, M., AND MAURER, U. General adversaries in unconditional multi-party computation. In *ASIACRYPT ' 99*.

[29] GENNARO, R., RABIN, M. O., AND RABIN, T. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *PODC '98*, pp. 101–111.

[30] GOLDREICH, O. *Foundations of Cryptography: Volume 2 – Basic Applications*. Cambridge University Press, 2004.

[31] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. How to play any mental game. In *STOC '87*.

[32] GOLDREICH, O., AND OREN, Y. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology 7*, 1 (1994), 1–32.

[33] GOLDWASSER, S., AND LEVIN, L. Fair computation of general functions in presence of immoral majority. In *CRYPTO '90* (1991), Springer-Verlag, pp. 77–93.

[34] HIRT, M., AND MAURER, U. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC '97*.

[35] HIRT, M., AND MAURER, U. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology 13*, 1 (2000), 31–60.

[36] KEARNS, M. Efficient noise-tolerant learning from statistical queries. In *STOC '93* (1993), pp. 392–401.

[37] LEHOUCQ, R. B., SORENSEN, D. C., AND YANG, C. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[38] LINDELL, Y., AND PINKAS, B. Privacy preserving data mining. *Journal of cryptology 15*, 3 (2002), 177–206.

[39] LINDELL, Y., PINKAS, B., AND SMART, N. P. Implementing two-party computation efficiently with security against malicious adversaries. In *SCN '08*.

[40] MALKHI, D., NISAN, N., PINKAS, B., AND SELLA, Y. Fairplay—a secure two-party computation system. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2004), USENIX Association, pp. 20–20.

[41] MCSHERRY, F., AND MIRONOV, I. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *KDD '09*.

[42] MCSHERRY, F., AND TALWAR, K. Mechanism design via differential privacy. In *FOCS '07*.

[43] NISSIM, K., RASKHODNIKOVA, S., AND SMITH, A. Smooth sensitivity and sampling in private data analysis. In *STOC '07* (2007), ACM, pp. 75–84.

[44] PAILLIER, P. Trapdooring discrete logarithms on elliptic curves over rings. In *ASIACRYPT '00*.

[45] PEDERSEN, T. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*.

[46] PINKAS, B., SCHNEIDER, T., SMART, N., AND WILLIAMS, S. Secure two-party computation is practical. Cryptology ePrint Archive, Report 2009/314, 2009.

[47] STEWART, G. W., AND SUN, J.-G. *Matrix Perturbation Theory*. Academic Press, 1990.

[48] TREFETHEN, L. N., AND III, D. B. *Numerical Linear Algebra*. SIAM, 1997.

[49] VAIDYA, J., AND CLIFTON, C. Privacy-preserving k-means clustering over vertically partitioned data. In *KDD '03*.

[50] WRIGHT, R., AND YANG, Z. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *KDD '04* (2004), pp. 713–718.

[51] YANG, Z., ZHONG, S., AND WRIGHT, R. N. Privacy-preserving classification of customer data without loss of accuracy. In *SDM 2005* (2005).

[52] YAO, A. C.-C. Protocols for secure computations. In *FOCS '82* (1982), IEEE, pp. 160–164.

Notes

[1]Most mining algorithms need to bound the amount of noise in the data to produce meaningful results. This means that the fraction of cheating users is usually below a much lower threshold (e.g. $\alpha < 20\%$).