CloudMine: Multi-Party Privacy-Preserving Data Analytics Service

Dinh Tien Tuan Anh, Quach Vinh Thanh, Anwitaman Datta
ug93tad@gmail.com, \{vtquach,anwitaman\}@ntu.edu.sg

School of Computer Engineering
Nanyang Technological University

Abstract. An increasing number of businesses are replacing their data storage and computation infrastructure with cloud services. Likewise, there is an increased emphasis on performing analytics based on multiple datasets obtained from different data sources. While ensuring security of data and computation outsourced to a third party cloud is in itself challenging, supporting analytics using data distributed across multiple, independent clouds is even further from trivial. In this paper we present CloudMine, a cloud-based service which allows multiple data owners to perform privacy-preserved computation over the joint data using their clouds as delegates. CloudMine protects data privacy with respect to semi-honest data owners and semi-honest clouds. It furthermore ensures the privacy of the computation outputs from the curious clouds. It allows data owners to reliably detect if their cloud delegates have been lazy when carrying out the delegated computation. CloudMine can run as a centralized service on a single cloud, or as a distributed service over multiple, independent clouds. CloudMine supports a set of basic computations that can be used to construct a variety of highly complex, distributed privacy-preserving data analytics. We demonstrate how a simple instance of CloudMine (secure sum service) is used to implement three classical data mining tasks (classification, association rule mining and clustering) in a cloud environment. We experiment with a prototype of the service, the results of which suggest its practicality for supporting privacy-preserving data analytics as a (multi) cloud-based service.

Keywords: delegated multiparty computation, privacy-preserving data analytics, multi-cloud, cloud service

1 Introduction

An enormous amount of data is being generated everyday from a plethora of computing devices. Traditionally, data is stored in the data owner’s in-house infrastructure, and access to outsiders is provided typically through web services. Data from multiple sources can be mashed-up or jointly analyzed to create new services and derive information that cannot be realized from individual datasets. However, it is often desirable or even required by law
to protect data privacy. Although numerous techniques for carrying out privacy-preserving data analytics exist ([26][14]), we believe that for wide-scale adoption of such techniques, it is essential to provide them as basic, out-of-the-box services which are flexible enough, so that individual users can freely choose their respective service providers, and yet be able to collaborate among each other.

Recent developments of cloud computing have materialized a concrete platform for rapid realization of the service-oriented computing paradigm [43]. Cloud providers (Google, Amazon, Salesforce, etc.) offer computing as a service, from which software services can be built, sold and integrated into complex applications. Migration of private IT infrastructures to the cloud is gathering momentum [24][38], as many companies and government agencies are moving most (or all) of their data, application logics and front-end services to the cloud. Recent advances in cloud computing have largely succeeded in accommodating the demand for cheap, elastic and scalable computing resources. However, security issues related to the outsourced data and computation remain a challenging obstacle to overcome [34][11].

Our work is motivated by the realization of these two trends, namely the need for a service for privacy-preserving analytics and the availability of cloud computing as a platform for service-oriented computing. More specifically, this work concerns the design space of a cloud-based service for carrying out distributed, privacy-preserving data analytics. We present CloudMine, a cloud-based, on-demand service that data owners can leverage to perform analytics over their joint data. CloudMine runs on the cloud (or delegate) and supports three basic functions: secure sum, secure set union and intersection, secure scalar product. CloudMine provides three security assurances. First, confidentiality of individual’s data is protected from other semi-honest data owners, as well as from colluding, semi-honest clouds. Second, outputs of the joint computations are protected from the semi-honest clouds. Third, data owners can reliably detect if their delegates have been lazy, i.e. if they have skipped the computations.

CloudMine can be used in a centralized manner when all the data owners use the same service on a single cloud. More importantly, it also works well in distributed settings where different data owners invoke different services on their delegates. In such setting, multiple instances of CloudMine participate in a distributed protocol in order to achieve the same functionality. The security properties of CloudMine are still guaranteed in this distributed environment, even when the clouds collude with each other.

A use case of CloudMine is illustrated in the following example. Suppose there is a number of supermarkets wishing to learn customer purchase behavior by performing association rule mining over their joint data. Each supermarket stores their customer transaction data in-house because the data contains sensitive information, while outsourcing the rest of its IT operation to the cloud. Suppose the supermarkets would like to outsourcing the computation (association rule mining) to their delegate clouds, without revealing their sensitive data to the clouds and to each other. Since the customer purchase behavior (output of the computation) is valuable to the participating supermarkets, they will like it
to be kept secret from the clouds (for otherwise, the latter can benefit from the information without contributing any data). They will also like to be able to detect if their clouds have been unscrupulous, i.e. skipping the delegated computations while still charging them for the same. Such lazy behavior could undermine accuracy of the final result. CloudMine meets these functionality and security requirements, and it can be readily invoked on the clouds. First, CloudMine supports set intersection and sum operations, which can be used to carry out association rule mining [13]. Next, CloudMine protects confidentiality of data owner’s input from other data owners and from the clouds, thus the supermarkets can be assured of the data privacy from each other and from the clouds. In addition, CloudMine protects output of the computation from the clouds, therefore the result from association rule mining is only learned by the participating supermarkets. Finally, CloudMine allows data owners to detect lazy clouds, thus the supermarkets can use CloudMine to verify if their clouds have been unscrupulous.

Privacy-preserving data analytics is an active area of research. Existing techniques are based on either a generic secure multi-party computation [45,44], or on using a semi-honest third party [15,26]. Our work distinguishes itself from the former in that the clouds are used as delegated computation units, hence it is more scalable. It differs from the latter in that we consider a stronger adversary model for the cloud delegates. Especially, we consider colluding adversaries who try to learn both the inputs and outputs of the computation while doing as little as possible. Furthermore, while previous works consider ad-hoc sets of data analytic tasks, each focusing on one primitive function (mostly secure sum function), CloudMine is designed as a service with a large set of analytic functions including secure sum, set operations and scalar products. We defer more detailed discussion to the next section.

The key enabling technique used in CloudMine is additive homomorphic encryption [32], which allows data owners to encrypt their private inputs before exporting it to the CloudMine cloud delegate. The ciphertexts contain additional information to allow for verification of computation. Secure set operations (intersection and union) are reduced to secure sum operations by encoding set membership into the plain-text inputs. Scalar product is computed by leveraging the homomorphic property of Paillier encryption and the secure sum function. In all cases, the keys are kept secret from the clouds, hence they are unable to decrypt the outputs.

Our contributions are as follows:

1. We present a model for cloud-based services for distributed, privacy-preserving data analytics. The model allows data owners to outsource their private computations to the cloud in a privacy-preserved manner.
2. We describe how the service can be implemented to support a number of cardinal data analytic functions, namely secure sum, secure set operations, and secure scalar product. We name the service CloudMine.
3. We demonstrate how CloudMine can be used for more complex data mining tasks — namely classification, association rule mining and clustering — in
a hybrid cloud setting. In particular, we show how CloudMine works when some parts of the data are stored in encrypted form in the public clouds.

4. We benchmark CloudMine on a cloud platform, both as a stand-alone service and as a part of more complex data mining applications. The results suggest that the overheads incurred because of the added security mechanism are reasonable and amortized as the workload increases. They indicate that it is practical to outsource distributed, privacy-preserving data analytics to a (multi) cloud service.

In the next section, we discuss in detail the system and adversary model of CloudMine. Section 3 delineate the CloudMine protocols for various analytic functions. Section 4 describes how three classic data mining tasks can be built using an instance of CloudMine in a hybrid cloud setting. Section 5 follows with experimental evaluation before related works are discussed in Section 6. We conclude and outline some planned future work in Section 7.

2 CloudMine Model

2.1 System Model

The system using CloudMine consists of two kinds of entities: data owners (or parties) and clouds (or delegates). The data owners \( P = \{P_0, P_1, \ldots, P_{n-1}\} \) wish to compute a function \( f(x_0, x_1, \ldots, x_{n-1}) \) where \( x_i \) is the input of \( P_i \), without revealing the input to each other. The clouds \( C = \{C_0, C_1, \ldots, C_{k-1}\} \) where \( k \leq n \) are the service providers. Each party uses one of these clouds, and each cloud is utilized by at least one party. Denote \( \delta(i) \in C \) as the delegate used by party \( P_i \).

At a high level, data owners use CloudMine in two steps in order to compute \( f(\cdot) \). First, they enter the setup phase, in which they agree on a function \( \phi \) (and \( \phi^{-1} \)) and a secret \( sk \). Next, each party \( P_i \) computes \( \phi_{sk}(x_i) \) and sends it to the delegate \( \delta(i) \). In turn, the delegates exchange messages among themselves and effectively compute \( \pi = \phi_{sk}(f(x_0, x_1, \ldots)) \). The data owners receive \( \pi \) from their respective delegates and compute \( \phi_{sk}^{-1}(\pi) = f(x_0, x_1, \ldots) \).

2.2 Adversary model

Data owners / parties are curious but honest. They follow the protocol for computing \( f(\cdot) \) correctly, but passively try to learn the private inputs of each other. They could collude with each other, but the number of colluding parties is less than \( n - 1 \).

Clouds/Delegates are curious and lazy. They are curious with respect to the parties’ private inputs as well as the output of \( f(\cdot) \). They do not actively subvert the computation, but are lazy in the sense that they try to do as little as possible while charging the data owners for the same. For example, they may skip some (or all) of the computations, replay results from the previous rounds, or even replace inputs from the data owners with other values in order to avoid computation. This model is justified by the economic incentives of the cloud
providers to over-charge customers without being detected [42], as well as the legal realities in which the clouds can sniff sensitive information without the liability of committing a criminal offense.

The collusion between parties and delegates is weak. In particular, the delegates may reveal the messages exchanged during the computation of $\phi(f(\cdot))$ to the parties, but the shared secret between the parties are not revealed to the delegates. If the shared secret is revealed, it is not possible to guarantee privacy of the computation output.

2.3 Security goals

Given the model above, CloudMine aims to provide the following security assurances:

1. Data owners cannot learn each other’s private inputs.
2. Delegates cannot learn the parties’ private inputs, nor can they learn the output $f(\cdot)$.
3. Delegates cannot skip, replay or replace inputs of the delegated computations without being detected by the parties.

2.4 Discussion

Existing works on multi-party private computation, which underlie privacy-preserving data analytics, can be grouped into two different approaches. The first is based on secure multi-party computation, in which data owners interact with each other directly to evaluate a function based on their private inputs. For example, [40,44] use generic multi-party computation circuits [45]. The second approach is based on a third party, in which data owners send their encrypted inputs to the third party which evaluates the function. [15,36,26], for instance, follow this approach.

Our model differs to the secure multi-party computation approach mainly in that the parties delegate their computations to the clouds. As a result, instead of interacting with each other, which does not scale well with the size of $n$, each party only interacts with its delegate. More importantly, this model allows for much more efficient implementation of the private computation than using generic, circuit evaluation (which takes in the order of seconds to compute a 2-party secure sum [15]).

Our model share some similarities with the second approach. On one hand, when $k = 1$, the system model of CloudMine is the same as in many other works which rely on a single third party. On the other hand, CloudMine distinguishes itself in a number of aspects. First, we consider the case when there are multiple, independent third parties that each data owner can individually choose to use as delegate. CloudMine is designed to resist collusion among these delegates. This is different from [15] which also supports multiple servers, but they are assumed to be non-colluding. Second, CloudMine adversary model considers the delegates trying passively to learn the output of $f(\cdot)$, which is not the case in previous
work. We believe such outputs may leak sensitive information. For example, the clouds may use the aggregate (sum) values together with off-line knowledge to derive sensitive information [16], or they may directly infer parts of the data owners’ private inputs from the output of the set intersection function. Third, we consider the clouds to be lazy which may skip the delegated computation and subsequently render the output \( f(\cdot) \) incorrect. This behavior presents a realistic threat to the utility and integrity of the analytics, yet it has not been addressed in existing works.

Finally, designing CloudMine as a service on the cloud has another benefit with respect to scalability. Since the cloud maintains the service, it can monitor the workload and automatically add more resources to deal with increases in workload. This automatic, seamless scaling is an essential practical improvement over systems such as [15] which require complete reconfiguration and re-run of the protocols to accommodate more servers.

3 CloudMine Implementation

We now describe how to implement the CloudMine service to support three analytic functions: secure sum, secure set operations (intersection and union), and secure scalar product. These primitives serve as a powerful toolbox for doing privacy-preserving analytics, ranging from database queries such as join [31] and aggregate [26] to complex mining algorithms such as collaborative filtering [15].

CloudMine relies on an additively homomorphic encryption scheme to protect privacy of the data owners’ inputs and to implement the basic secure sum function. In particular, we use Paillier [32], a randomized encryption scheme consisting of three algorithms \((\text{Gen, Enc, Dec})\) where \(\text{Enc} \) and \(\text{Dec} \) are encryption and decryption algorithms which use the key generated by \(\text{Gen}\). Paillier has the following property:

\[
\text{Enc}(eK, m_1) \cdot \text{Enc}(eK, m_2) = \text{Enc}(eK, m_1 + m_2)
\]

where \(eK\) is the encryption key. Compared to other additively homomorphic schemes based on Elgamal [39], Paillier requires longer bit-length. But we can overcome this by packing multiple inputs into a single plaintext so that they can be encrypted and decrypted at the same time [34]. Suppose the inputs are at most \(b\) bits and Paillier’s plaintexts are \(bl\) bits. Suppose further that any sum value is smaller than \(2^{t+b}\) for some values of \(t\), then we can pack \(c\) inputs into a single plaintext (where \(c \leq \left\lceil \frac{bl}{t+b} \right\rceil \)) as follows:

\[
\langle x_1 || x_2 || \ldots || x_c \rangle = z || x_1 || z' || x_2 \ldots || z' || x_c
\]

where \(z'\) contains \(t\) bits of 0 and \(z\) contains \((bl - c(t + b))\) bits of 0.

In the following, we describe the construction of three services that constitute CloudMine. The secure sum service implements the aggregate function, secure set service the set union and intersection function, and secure scalar service the scalar product function.
3.1 Secure Sum Service

The secure sum service, denoted as $S_{\text{sum}}$, consists of five protocols: $S_{\text{sum}} = (\text{Setup}, \text{KeyGen}, \text{MaskGen}, \text{ComputeSum}, \text{Verify})$. The first three protocols are performed once at the beginning, while \text{ComputeSum} and \text{Verify} are invoked for each round of computation.

- \text{Setup}(\kappa): generate public parameters with $\kappa$ being the security parameter. The result is the tuple:
  \[ PK = (\text{gid}, \text{RNG}, G_p(g), \text{Gen}, b, bl) \]
  where $\text{gid}$ identifies the group to which all parties belong, $\text{RNG}$ is a random number generator, $G_p(g)$ is an algebraic group of prime order $p$ and generator $g$, $\text{Gen}$ is an algorithm for generating Paillier keys. $b$ and $bl$ are bit lengths of the inputs and Paillier plaintexts respectively.

- \text{KeyGen}(PK): data owners execute this protocol to establish a share secret:
  \[ SK = (eK, dK, rId) \]
  where $(eK, dK)$ is a Paillier key pair and $rId$ is a random number identifying the initial round of computation. First, the parties follow the protocols as proposed in \cite{11} to generate a secret $x \in G_p$ using the clouds and without the latter learning $x$. Next, they use $x$ as the seed to initialize the random number generator $\text{RNG}$, which is then used by $\text{Gen}$ to generate $(eK, dK)$. Finally, the parties assign the next random number generated by $\text{RNG}$ as $rId$.

- \text{MaskGen}(i, PK): each party $P_i$ invokes this protocol to generate its own secret
  \[ MaK_i = (r_i, \eta_i) \]
  such that $r_i, \eta_i$ are random values from an algebraic group of specific size, $\eta_i \neq 0$ and the sum of $r_i$ and $\eta_i$ across all data owners are known, i.e. $\sum_i r_i = \sum_i \eta_i = 0$.
  First, $P_i$ creates random values $r_{ij}$ and $\eta_{ij}$ (using its private source of randomness) for all $P_j$ ($i \neq j$) belonging to the group $\text{gid}$. Next, $r_{ij}$ and $\eta_{ij}$ are encrypted with $P_j$’s public key and sent to $\delta(i)$ which subsequently forwards them to $P_j$. Having received the encrypted $r_{ji}, \eta_{ji}$ from its delegate, $P_i$ then computes
  \[
  r_i = \sum_{i \neq j} (r_{ij} - r_{ji}) \quad \eta_i = \sum_{i \neq j} (\eta_{ij} - \eta_{ji})
  \]
  It can be seen that $P_j$ cannot learn $r_i, \eta_i$ for $i \neq j$, and that $\sum_i r_i = \sum_i \eta_i = 0$.

- \text{ComputeSum}(i, PK, SK, MaK_i, x_i): each party $P_i$ constructs the ciphertext $c_i$ for its private input $x_i$ as follows:
  \[ c_i = \text{Enc}(eK, (\eta_i || rId || (x_i + r_i))) \]
It then sends $c_i$ to its delegate which then broadcasts it to the other delegates. Finally, each delegate computes:
\[
c = \prod c_i = \text{Enc}(eK, \langle \sum \eta_i || n.\text{rId} || \sum x_i \rangle)
\]
and forwards it to the party. Finally, $P_i$ invokes $\text{Verify}(i, PK, SK, MaKi, c)$. If the result of this verification protocol is $y \neq \perp$, the party returns $y$ as the final sum.

- **ComputeSum**(i, PK, SK, MaKi, $\mathbf{c}$): takes as parameter a vector of inputs $\mathbf{c}$ instead of a single input. Let $s = [\frac{b(l-2(b_r + \log_2 n))}{\log_2 r}]$ where $b_r$ is the bit length of $\eta_i$ and rId. For $0 \leq k < [\frac{b.\mathbf{c}}{s}]$, we construct message $m_k$ as follows:
\[
m_k = \langle x_{k.s} || x_{k.s+1} || . . . . || x_{(k+1).s-1} \rangle
\]
where $|m_k| = bl - 2(b_r + \log_2 n)$ bit. For each $m_k$, the party invokes $\text{ComputeSum}(i, PK, SK, MaK_i, m_k)$. If the result $y \neq \perp$, it extracts the sums $y_{k.s}, y_{k.s+1}, . . . , y_{(k+1).s-1}$ from $y$. After each invocation, the party increments rId and updates SK accordingly.

- **Verify**(i, PK, SK, MaKi, c): each party decrypts the ciphertext $c$ and checks that the result is of the following form:
\[
\text{Dec}(dK, c) = (0 || n.\text{rId} || y)
\]
If true, $y$ is returned as the final sum.

**Discussion.** We now discuss how the protocols above meet the security requirements listed in Section 3. First, data owners cannot learn each other’s inputs, because each input $x_i$ has been masked with a secret value $r_i$. Second, delegates cannot extract the sum $\sum x_i$ from the ciphertext $c$, because they do not have access to the decryption key $dK$. Third, delegates cannot replay old values without being detected, since each ciphertext is embedded with a fresh value of rId. They cannot replace $c_i$ with another valid ciphertext either, because they do not have access to $(eK, r_i, \eta_i)$, thus invalid ciphertexts will be detected by the verification protocol. Neither can they skip some (or all) of the inputs for the computation of $c = \prod c_i$, because it will cause verification to fail, since $\text{Dec}(c) \neq (0 || n.\text{rId} || y)$. Finally, each delegate can compute $c = c_i^n$ (raising to the power of $n$ may be cheaper than $n$ multiplications), which makes the second element of $\text{Dec}(c)$ to be the same as n.rId. However, verification will still fail, because $n.\eta_i \neq 0$.

Security of $S_{\text{sum}}$ depends on the fact that delegates do not know the shared secret SK or the data owner secret MaKi. Every party $i$ must protect MaKi from other parties. To ensure long-term security, it is important to refresh SK as well as MaKi, albeit refreshing the latter can be done after longer intervals. This can be achieved by invoking $\text{KeyGen}$ and $\text{MaskGen}$ again. Alternatively, if $P_i$ stores the original $\{r_{ij}, r_{ji}, \eta_{ij}, \eta_{ji} \ | j \neq i\}$, the new MaKi can be computed as:
\[
r'_i = \sum_{i \neq j} (H(r_{ij}) - H(r_{ji})) \quad \eta'_i = \sum_{i \neq j} (H(\eta_{ij}) - H(\eta_{ji}))
\]
where $H$ is a cryptographic hash function.

The verification of delegate behavior relies on the party encoding its secret $MaK_i$ to the ciphertexts. As a consequence, the memory overhead is $o = \frac{2(b_r + \log_2 n)}{bl}$, which decreases as the Paillier bit-length $bl$ increases. The Verify protocol is performed at the end of every ComputeSum protocol. This can become overhead when there are many rounds of computations. Hence, we extend $S_{\text{sum}}$ to allow parties to invoke Verify only with a probability $p$. The probability of successfully detecting consistent misbehavior, $p_v$, can be made arbitrarily high after a number of verification. Specifically, $p_v = 1 - (1 - p)^n \cdot k$ where $k$ is the number of random checks.

### 3.2 Secure Set Service

The service for secure set union and intersection can be built directly from the secure sum service. Intuitively, the input sets are encoded into plaintext messages which are used as inputs for $S_{\text{sum}}$. The union or intersection set is then decoded from the final sum values.

The secure set service, denoted as $S_{\text{set}}$, consists of five protocols:

$S_{\text{set}} = (\text{Setup}, \text{KeyGen}, \text{MaskGen}, \text{ComputeUnion}, \text{ComputeIntersect})$.

- **Setup, KeyGen, MaskGen** are the same as in the secure sum service, except that the public parameter $PK$ also contains a universal domain $U$.

- **ComputeUnion$(i, PK, SK, MaK_i, \pi_i)$**: each party inputs a vector $\pi_i \in U^*$ and computes the union set as follows. A vector $I = (a_0, ..., a_{|U|-1})$ is constructed, in which $a_i = 1$ if $U[i] \in \pi_i$ and $a_i = 0$ otherwise. The party then invokes ComputeSum$(i, PK, SK, MaK_i, I)$. When the secure sum service returns $s_0, s_1, ..., s_{|U|-1}$, it computes $\{U[i] | s_i \geq 1\}$ as the union set.

- **ComputeIntersect$(i, PK, SK, MaK_i, \pi_i)$** works in the same way as ComputeUnion, except that the intersection set is computed as $\{U[i] | s_i = n\}$.

**Discussion.** This service has the same security properties as for secure sum. The number of encryptions per set operation is $\lceil \frac{|U|((\log_2 n + 1))}{bl-2(b_r+\log_2 n)+b} \rceil$, which grows linearly with the size of $U$. Consequently, our protocols may not scale well when $U$ and $n$ are very large (for example, in orders of millions as in the case of large-scale collaborative filtering). Other protocols for private set operations which scale more gracefully ([25,31]) do not apply to our delegate model. In practice, many applications involving secure set operations have small- to medium-size $U$ (in orders of ten or hundred) [4], which renders our protocols practical. For example, for 100 data owners, $|U| = 1000$, $bl = 1024$, $b = 16$, the protocols need only 9 encryptions. We believe that for current applications, this cost is reasonable.
3.3 Secure Scalar Service

Data owners are divided into two disjoint groups $X, Y$. Let $\vec{x}$ and $\vec{y}$ be two vectors in which $x_i$ is the private input of party $X_i$, $y_i$ the private input of $Y_i$.

The secure scalar service, $S_{sp}$ allow the data owners in both groups to compute

$$p = \vec{x} \cdot \vec{y} = x_0 y_0 + x_1 y_1 + \ldots + x_{n-1} y_{n-1}$$

$S_{sp}$ consists of four protocols ($\text{Setup, KeyGen, MaskGen, ComputeScalar}$), and it also makes use the secure sum service.

- **Setup($\kappa$)** is the similar to that in $S_{sum}$. It outputs public the parameter:

$$PK = (\text{gid}_x, \text{gid}_y, \text{RNG}, \mathbb{G}_p(g), \text{Gen}, b, bl)$$

where $\text{gid}_x$ and $\text{gid}_y$ are identities of group $X$ and $Y$ respectively.

- **KeyGen($gid, PK$)**: parties that belong to the group $gid$ execute this protocol to establish a shared secret among them. The protocol is the same as in $S_{sum}$, and the result is

$$SK_{gid} = (eK_{gid}, dK_{gid}, rId_{gid})$$

- **MaskGen($i, gid, PK$)**: each party $i$ in group $gid$ first generates a Paillier key pair $(eK'_{gid,i}, dK'_{gid,i})$. Next, it generates two values $r_{gid,i}, \eta_{gid,i}$ in the same way as in $S_{sum}$, i.e. $\eta_{gid,i} \neq 0$ and $\sum_i \eta_{gid,i} = \sum_i r_{gid,i} = 0$. Denote

$$MaK_{gid,i} = (dK_{gid,i}, r_{gid,i}, \eta_{gid,i})$$

as the secret of party $i$ in group $gid$. Also, let

$$PK' = \{eK'_{gid,i} \} \cup \{eK'_{gid,y,i} \}$$

be another set of public parameters.

- **ComputeScalar($i, gid, PK, PK', SK_{gid}, MaK_{gid,i}, x_i$)**: party $i$ in group $gid$ executes this protocol to compute the global scalar product. Suppose the party is $X_i$ (belonging to group $X$, and $gid = \text{gid}_x$), the protocol proceeds as follows:

1. $X_i$ sends $m_i = \text{Enc}(eK'_{gid,x,i}, x_i)$ to its delegate which then forwards it to $Y_i$.

2. $Y_i$ computes $c_i = m_i^{y_i} \cdot \text{Enc}(eK'_{gid,y,i}, r_{gid,x,i})$ and sends it back to $X_i$ via its delegate.

3. $X_i$ computes $z = \text{Dec}(dK'_{gid,x,i}, c_i) = x_i y_i + r_{gid,y,i}$. It then invokes the service sum service $\text{ComputeSum}(i, PK, SK_{gid}, MaK_{gid,i}, z)$, the result of which is the scalar product.
**Discussion.** The intuition behind ComputeScalar protocol is for $X_i$ to compute the value $(x_i, y_i + r_{gid,i})$ without knowing $y_i$. This is then aggregated with the other values from $X_j$ $(i \neq j)$ to cancel out $r_{gid,j}$ and obtain the scalar product. This works because of the homomorphic property of Paillier and the fact that $\sum r_{gid,i} = 0$.

The party $X_i$ cannot learn input of $Y_i$, because the sum $x_i, y_i$ is masked by a random value $r_{gid,i}$. Neither can $X_i$ learn the input of $X_j$ $(i \neq j)$ due to the property of the secure sum service. The delegates can neither learn the intermediate sum $x_i, y_i$ because they are encrypted with data owners’ keys, nor the final scalar product because $S_{sum}$ does not reveal the final sum to the delegates. In ComputeScalar, the delegates play two roles: forwarding messages between parties and performing secure sum computations. For the former, the delegates cannot be lazy without being detected, because messages are acknowledged (so they cannot be skipped) and freshly signed (so they cannot be replayed). For the latter, the secure sum protocol ensures that lazy behavior will be reliably detected.

It can be seen that delegates are more involved in this service than in $S_{sum}$ or in $S_{set}$. In particular, they must keep track of the group to which each party belongs, and must forward messages to the correct delegates. This management task, if left to the data owners, may become impractical for large systems. Since CloudMine is a cloud-based service, such tasks can be performed by the cloud in a scalable way.

### 4 Data Mining in Hybrid Clouds

The hybrid cloud model, in which the user utilizes the combined resources of its private infrastructure (or private cloud) and a public cloud, helps ease the transition from in-house to public-cloud computing. This model is motivated by the need to optimize cost and performance, to cater for different demand patterns, or to mitigate risks [18]. In this section, we demonstrate how CloudMine’s secure sum service can be used to implement distributed, privacy-preserving data mining algorithms in this hybrid environment.

We consider data owners as hybrid-cloud users, who partition their data into two parts: the sensitive part maintained in the private cloud, and the less sensitive part stored in a public cloud [46]. For example, data generated by an intrusion detection system may consist of highly sensitive records associated with the internal system, whereas traffic to/from the front-end servers may be regarded as less sensitive. Another example is in large scale genomic sequencing: an individual’s DNA sequence is highly sensitive and must be handled in the private cloud, whereas a reference genome can be considered as less sensitive and therefore can be encrypted and outsourced to a public cloud [12]. Note that less sensitive is not the same as non-sensitive, in the sense that data owners still want to have some levels of privacy with the less sensitive data. We distinguish two logically separate delegates: a computation delegate which runs CloudMine service, and a data delegate which maintains the owner’s data. They may be-
long to the same cloud, or each to a different cloud. Adversary model for the computation delegates is the same as in the previous section. Adversary model for the data delegates adversary model is also curious-and-lazy. In particular, they try to learn the data stored on the public clouds, and try to do as little as possible when answering data queries from the owners. They may collude with each other, but they will not tamper with the data.

To protect the outsourced data from curious delegates, an encryption scheme must be used. In our design, we employ two encryptions scheme: AES and Order-Preserving Encryption (OPE) [10,33]. AES is a deterministic scheme that supports equality comparison of ciphertexts. OPE offers weaker security guarantees, but it supports inequality comparison of ciphertexts, which can be used for range queries. For the sake of simplicity, we store two encrypted copies of the data on the data delegates (a more elegant approach can be found in CryptDB [34]). We use the OPE scheme from [10], which is a stateless encryption and does not require a third-party server (as in [33]).

Untrusted data delegates necessitate protocols for ensuring query assurance. In the literature, techniques for query assurance are probabilistic which make use of redundant query execution (ringer schemes) [37,14,27]. In this work, we use a mechanism based on [37], in which the data owner maintains a random, small portion of the outsourced data in its private cloud. Queries to the delegates are extended with a number of fake queries, and the results are probabilistically checked by querying the local copy of the data. Our experiments show that maintaining as little as 15−20% of the outsourced data locally is sufficiently effective to detect lazy delegates after a small number of checks.

In the following, the data mining algorithms are run on the private cloud of each data owner. We assume, for simplicity, that data is in relational format and every attribute belongs to a non-negative integer domain. The algorithms consist of an iterative process of querying the public-cloud database, combining it with the local data, and using the result as inputs to the secure sum service. The fact that outputs from the interactions with the data delegates are used during the computations involving cloud delegates may appear to be a risk to privacy, especially when data and computation delegates collude (which is immediate when they belong to the same cloud provider). However, privacy is ensured for two reasons. First, the computation delegates cannot learn the data owners’ inputs to the private computation, because the inputs are obtained over both the data stored in the private cloud and data outsourced to the data delegate. Hence, results from querying the data delegates only contribute partly to the inputs. Second, and more importantly, even if all the data is outsourced, the delegates cannot collude and compute analytics by themselves, because both the data and the meta-data (column names, table names, etc.) are encrypted.

4.1 Classification (Naive Bayes).

A classification algorithm takes as input a set of labeled, training data and outputs a classifier that can be used to assign label to new data. Let $N$ be the number of data instances, $Y$ the set of labels, $A$ the set of attributes and $V_A$ the
Algorithm 1: Naive Bayes classification

1 **Input:** $Y, A, V, i$
2 **Output:** $N, \{N_y\}, \{N_{y,a,v}\}$
3 $PK \leftarrow \text{Setup}(\kappa); SK \leftarrow \text{KeyGen}(PK); MaK_i \leftarrow \text{MaskGen}(i, PK)$
4 foreach $y \in Y, a \in A, v \in V$
5   $N_y \leftarrow \text{QueryCount}(\text{label} = y)$
6   $N_{y,a,v} \leftarrow \text{QueryCount}(a = v, \text{label} = y)$
7 foreach $y \in Y, a \in A, v \in V$
8   $N_y \leftarrow \text{ComputeSum}(i, PK, SK, MaK_i, N_y)$
9   $N_{y,a,v} \leftarrow \text{ComputeSum}(i, PK, SK, MaK_i, N_{y,a,v})$

attribute domain for $a \in A$. The NaiveBayes algorithm shown in Algorithm 1 computes:

$$\text{classifier} = (N, \{N_y | y \in Y\}, \{N_{y,a,v} | y \in Y, a \in A, v \in V\})$$

The label for a new instance $x$ is:

$$\text{label}(x) = \arg\max_y (N_y \prod_i \frac{N_{y,i,x_i}}{N_y})$$

The protocol $\text{QueryCount}(a_1 = v_1, a_2 = v_2)$ encrypts $a_1, v_1$ with AES and issues a SQL query of the form

$$\text{select COUNT from Enc_{aes}(Data)}$$
$$\text{where Enc_{aes}(a_1) = Enc_{aes}(v_1)}$$
$$\text{AND Enc_{aes}(a_2) = Enc_{aes}(v_2)} \ldots$$

to the data delegate. The delegate executes the SQL query over the encrypted data and returns the result which is probabilistically verified by the owner.

4.2 Clustering (K-Mode).

A clustering algorithm partitions the data into separate clusters such that distance between members of the same cluster is smaller than that between members of different clusters. The K-Mode algorithm (Algorithm 2) finds $k$ clusters identified by their centroids (or modes) that minimizes the dissimilarity between members of the same cluster (the Mode function). The algorithm works in multiple rounds until the set of modes converges.

We use Manhattan distance to quantify the distance from a data instance $x$ to a mode $c$, i.e. $\Delta(x, c) = \sum_i |x_i - c_i|$. The protocol $\text{QueryGroupBy}(a, m_i, M)$ queries the data delegate for a list of frequencies for attribute $a$ in the portion
Algorithm 2: K-Mode Clustering

1. Input: $k, A, i$
2. Output: $M = \{m_1, \ldots, m_k\}$
3. $PK \leftarrow \text{Setup}(\kappa); SK \leftarrow \text{KeyGen}(PK); MaK_i \leftarrow \text{MaskGen}(i, PK)$
4. Initialize $m_j = \{j, j, \ldots, j\}$ for $m_j \in M$; $C^i = \emptyset$
5. foreach $m_j \in M$:
6. $C^p = \emptyset$
7. foreach $a \in A$:
8. $C^a_{m_j}(a) \leftarrow \text{QueryGroupBy}(a, m_j, M)$
9. $C^a_{m_j} = C^a_{m_j} \cup C^a_{m_j}(a)$
10. $C^i = C^p \cup C^a_{m_j}$
11. foreach $C^i[j] \in C^i$:
12. $C[i] \leftarrow \text{ComputeSum}(i, PK, SK, MaK_i, C'[i])$
13. $C' \leftarrow C$
14. foreach $m_j \in M, a \in A$:
15. $m_j(a) \leftarrow \text{Mode}(C^a_{m_j}(a))$
16. Repeat Step 5 until $M$ converges.

of data closest to the centroid $m_i \in M$. The query has the form:

\[
\text{select } \text{Enc}_{aes}(a), \text{COUNT from } \text{Enc}_{aes}(\text{Data}) \text{ as freq}
\]

\[
\text{where } \Delta(\text{Enc}_{opec}(a), \text{Enc}_{opec}(m_i)) < \Delta(\text{Enc}_{opec}(a), \text{Enc}_{opec}(m_0))
\]

\[
\text{AND } \Delta(\text{Enc}_{opec}(a), \text{Enc}_{opec}(m_i)) < \Delta(\text{Enc}_{opec}(a), \text{Enc}_{opec}(m_1))
\]

\[
\text{Group by } \text{Enc}_{aes}(a), \text{Order by } \text{Enc}_{opec}(a)
\]

Since $\Delta$ is computed over OPE ciphertext, the response from the cloud for $\text{QueryGroupBy}$ might not be accurate, as compared to the same query executed over the plaintext data. OPE’s only guarantee is $\text{Enc}_{opec}(x) < \text{Enc}_{opec}(y) \leftrightarrow x < y$, hence it does not always follow that $|\text{Enc}_{opec}(x) - \text{Enc}_{opec}(x')| < |\text{Enc}_{opec}(y) - \text{Enc}_{opec}(y')| \leftrightarrow |x - x'| < |y - y'|$. In the next section, we show that this phenomenon occurs frequently, yet the final clusters are very close to the clusters found using the unencrypted data.

4.3 Association rule mining (Apriori).

An association rule mining algorithm extracts the relationships between attributes that occur frequently in the data. An association rule has the form $(X \rightarrow Y)$ where $X, Y \subseteq A$. The Apriori algorithm (Algorithm 3) first determines frequent item-sets containing a single item using the $\text{GenerateFrequentItemset-Size1}$ protocol. The results are merged into larger item-sets (candidates) using $\text{GenerateCandidates}$. The threshold value $\text{minsup}$ specifies the lower bound for item-set frequency. These steps are repeated until there is no more item-set to be found. Finally, $\text{GenerateRules}$ generates the outputs by establishing rules whose
Algorithm 3: Apriori association rule mining

1. **Input**: minsup, minconf, i
2. **Output**: set of rules \{\(X \rightarrow Y\)\}

3. \(PK \leftarrow \text{Setup}(\kappa)\); \(SK \leftarrow \text{KeyGen}(PK)\); \(MaK_i \leftarrow \text{MaskGen}(i, PK)\)
4. \(L_1 \leftarrow \text{GenerateFrequentItemsetSize1}()\)
5. \(k = 2, B_i = \emptyset\)
6. \(C_k \leftarrow \text{GenerateCandidates}(L_{k-1})\)

7. foreach \(c \in C_k:\)
   8. \(t \leftarrow \text{QueryCount}(c)\)
   9. \(B_i = B_i \cup t\)

10. foreach \(j \in [1, k]:\)
   11. \(B[j] \leftarrow \text{ComputeSum}(i, PK, SK, MaK_i, B_i[j])\)
   12. extract \(c\.count\) from \(B[j]\)
   13. \(L_k \leftarrow \{c \in C_k | c\.count \geq \text{minsup}\}\)
   14. Increase \(k\) and repeat from line 6 until \(L_k = \emptyset\)
   15. \(\text{GenerateRules}(\bigcup_k L_k, \text{minconf})\)

Confidence values are above \(\text{minconf}\). The details of \(\text{GenerateFrequentItemsetSize1}\), \(\text{GenerateCandidates}\) and \(\text{GenerateRules}\) can be found in [7].

5 Evaluation

We have implemented the protocols described in the previous sections in order to demonstrate CloudMine’s functionality as well as to preliminarily assess its performance in a hybrid cloud environment. In particular, the prototype implements the secure sum service and three data mining algorithms built using this service. It is written in Java, with cryptographic operations provided by the Crypto++ library [1], OPE and Paillier encryptions by CryptDB library [34]. Data mining algorithms made use of the Weka library [29]. Communications between data owners and delegates are done via Java sockets. The source code is available at [https://code.google.com/p/cloudmine-sum/](https://code.google.com/p/cloudmine-sum/).

| Parameters | Description | Values |
|------------|-------------|--------|
| \(n\)      | number of parties | 2, 4, 8, 16 |
| \(k\)      | number of delegates | 1, 2, 4, 8, 16 |
| \(it\)     | EC2 instance types | small, medium, large |
| \(bl\)     | encryption bit length | 512, 1024 |
| \(r\)      | secure sum request rate | 10, 100, 200, 400, 700 |
| \(ds\)     | dataset | breast_cancer, x50_breast_cancer, mushroom, x50_mushroom, splice, x10_splice |
| \(alg\)    | data mining algorithm | NaiveBayes, Apriori, K-Mode |

Table 1: List of parameters used in experiments.
We first experimented with CloudMine as a stand-alone service. We used throughput — the number of secure sum operations completed per second measured at the party — as the metric. Next, we evaluated the performance of CloudMine when being used in complex data mining algorithms. For this, we measured the overall and detailed breakdown of the running time of each data mining algorithm. We ran all experiments on Amazon EC2 platform [9], using the parameters as listed in Table 1. Unless otherwise stated, each delegate runs on one large EC2 instance, and two parties share one large EC2 instance. In addition, $n = k = 8$ and $bl = 1024$. The results presented below are averaged over multiple runs.

5.1 Secure Sum Benchmark

![Graphs showing throughput comparison](image)

(a) Throughput, with varying request rate  
(b) Throughput at steady state, with varying $n$

Fig. 1: Secure sum throughput

To benchmark CloudMine service, we varied the frequency at which each party requests for the service from its delegate. We also varied the types of EC2 instances on which the party is run, and the number of parties sharing one instance. Figure 1[a] shows that throughput reaches its steady state at different values for different configurations of the party. In particular, the highest throughput is observed at 150 (sums/sec) when one party occupies one large instance. When two parties share the same instance, throughput dips to around 110 (sums/sec). When medium or small instances are used for the parties, throughput falls even further (the lowest is at 33 (sums/sec) with parties sharing small instances). These results indicate with fixed $n$ and $k$, throughput depends on the computation at the parties, i.e. the more powerful the parties are, the higher the overall throughput. Furthermore, considering that our prototype implementation has not been optimized for highly parallel workload, we believe these throughputs
are practical for many real-time applications in which data does not arrive at extremely high rates.

Figure 1[b] shows how throughput also depends on encryption bit-length $bl$, the ratio $\frac{k}{n}$ and the number of parties $n$. It can be easily seen that reducing the encryption bit-length from 1024 to 512 leads to substantial increase in throughput. This is because Paillier encryption and decryption operations take roughly 1ms when $bl = 512$, which rise to 7ms with $bl = 1024$. The ratio $\frac{k}{n}$ represents the level of decentralization. When $k = 1$, all parties communicate to one centralized delegate — the model adopted in [15,30,26]. When $k = n$, each party has one delegate and each delegate has one party. The results indicate that throughput is always slightly higher when $k = 1$ than when $k = n$. This means that throughput is mainly determined by the sum computation, as opposed to be affected by the communication overhead incurred when $k = n$. In other words, our service supports the decentralization of the multi-party computation with minimal cost to the overall performance. Thus, there is no substantial advantage, at least in terms of throughput, in using a centralized service for secure sum computation; whereas distributing this private computation over multiple delegates implies the decentralization of trust, which is a more acceptable model in practice. Finally, as $n$ increases, we can observe a drop in throughput. This is caused by the computation and communication overhead incurred at the delegates when $n$ gets larger. We will discuss this overhead in more detail shortly.

5.2 Data Mining Performance

We used three standard datasets: breast_cancer (small), mushroom (large, many rows) and splice (large, many columns) from [4], and synthesized larger
datasets by extending them with random values from similar distributions. For instance, \textit{x50_mushroom} represents the dataset 50-time the size of the original \textit{mushroom} dataset. The largest dataset consists of 91350 rows and 23 columns.

In our prototype, each data owner encrypts its data with AES and OPE and uploads it to the delegate which then stores it in a MySQL server. We let data owners outsource all of their data to the cloud, causing larger data query overhead than when parts of the data are stored locally. The encrypted datasets were as much as 23 times larger in size than the original, unencrypted ones (for the \textit{x10_splice} dataset). We quantify the costs for database encryption at the party and database loading at the delegate, which incur only once at the beginning, in terms of the time taken to complete the operations. Figure 2[a] illustrates these costs with varying datasets for the NaiveBayes algorithm. It can be seen that both encryption and loading time are proportional to the data size, and they remain below 8s even for the largest dataset. Figure 2[b] shows the loading time at the delegates when delegates are running on different types of EC2 instances. Across all datasets, using small instances results in longer loading time.

As explained in Section 4, a data mining application built using CloudMine consists of two iterative, interleaving processes: database query and secure sum. Figure 3 shows the breakdown costs of these processes — measured as the time taken to complete the process — for Apriori algorithm. One important observation is that database query time is always greater than secure sum time. For \textit{x50_mushroom} dataset, the former takes more than an order of magnitude longer to complete. The longest experiment (with \textit{x10_splice} dataset) took 12 minutes to complete, of which secure sum operation accounted for only 2 minutes. This suggests that when used in real data mining algorithms, the cost of the secure sum service has small effect on the overall performance. Figure 4 shows the effect

![Fig. 3: Overall running time, \textit{alg} = Apriori](image-url)
of increasing data size to the database query time for different algorithms. It can be observed that query time scales differently for different algorithms. Particularly, Apriori demonstrates the sharpest growth as compared to NaiveBayes and K-Mode. We attribute this to the intrinsic properties of the data mining algorithm. Specifically, we observe that in our experiments with Apriori, larger datasets led to more queries being performed by the delegate (from 132 with mushroom to 4214 with the x50_mushroom dataset).

Finally, we investigated the cost of the secure sum service as being used in data mining algorithms. Figure 5 shows this cost varies with $n$ for the Apriori algorithm. As the number of parties gets larger, the secure sum cost also increases, albeit at a sub-linear rate. This is consistent to what has been observed.
in Figure 1[b]. Recall that the cost of a secure sum operation comprises the encryption/decryption cost at the party and the computation and communication cost at the delegates. The former is shown in Figure 5 to be almost constant, meaning that the overhead incurred when \( n \) increases can be attributed to the overhead at the delegate. First, each delegate needs to perform more multiplications when \( n \) increases. Second, each will have to wait longer to receive all the messages from other delegates when \( n \) becomes bigger.

**Correctness of K-Mode.** As explained in Section 4, the QueryGroupBy protocol in K-Mode may return a different result as compared to performing the corresponding query locally on the plaintext data. We refer to this as mismatched query, whose error may affect the convergence rate of the algorithm as well as the final clusters. All of our experiments with K-Mode converged to final modes. To quantify the differences between clusters found by using CloudMine and what are found using standard K-Mode over plaintext data, we used an error metric \( \epsilon(C_i, C'_i) = \frac{|\Omega(C_i) - \Omega(C'_i)|}{\Omega(C'_i)} \) where \( C_i, C'_i \) denote the two clusters and \( \Omega(C_i) \) is the mean squared distance of the members of \( C_i \) to the mode. While the average number of mismatched queries ranges from 0 (for mushroom dataset) to 508.2 (for splice dataset), the maximum error is 0.03. This means our protocols yield nearly identical clusters to what obtained from the standard K-Mode.

5.3 Discussion.

The results above have demonstrated that there are overhead incurred by cryptographic operations when using CloudMine, as compared to when the data owners use their own infrastructure and directly take part in the multi-party protocol with each other. While these costs are necessary to provide security in the presence of the delegates, we also remark that when used in the context of data mining, they become less substantial, and can be more than offset by the benefits gained from using elastic cloud resources. In particular, let \( m \) be the number of secure sum messages sent and received by the data owners during a data mining algorithm. Let \( \alpha \) be the cryptographic cost for encrypting and decrypting a message (with additive homomorphic encryption schemes). Let \( q \) be the number of database queries and \( c_q \) the CPU cost for each query. The computation overhead at each data owner becomes \( O = (C_d - C) = (\alpha.m - q.c_q) \) where \( C \) is the cost when the data owner uses its own infrastructure. It can be seen that \( O \) diminishes quickly and becomes negative for larger workloads: more complex data mining algorithms with high value of \( q \) or larger datasets with high \( c_q \). It has been shown in Figure 3, for example, that the database query costs may be over an order of magnitude more than the costs incurred by the secure sum service.
6 Related Work

CloudMine shares common goals with many other works in the area of distributed, privacy-preserving data analytics. Our work is not based on randomization approach [8] which perturbs the inputs or differential privacy [16] approach which adds noise to the outputs. Instead, CloudMine follows the secure multi-party computation approach [45] in preserving data privacy during computation. It has been shown that any computation can be done in a private manner, by reducing the computation to a combination of circuits. Vaidya et al. [40] use generic circuits for evaluating 2-party comparison operation, which is then used for K-Means algorithm over vertically partitioned data. Yang et al [44] use generic circuits for computing Bayesian networks on vertically partitioned data. CloudMine does not rely on circuit evaluation, which is either expensive [15] or is restricted to two-party computation [25]. Instead, it shares similar model to what is proposed in [15,26,36,35] which rely on third-party servers. However, these works focus on specific functions for specific application domains. In contrast, CloudMine is designed in a service-oriented manner, that can be flexibly used by a wide range of applications. Furthermore, the adversary model of CloudMine is stronger than in the aforementioned previous works.

Our delegated computation model is a special case of verifiable computation, in which a client outsources its computations to a more powerful entity and is able to later verify the outputs. Theoretical results have shown that any computation can be outsourced with guaranteed input and output privacy [19]. However, a general protocol for outsourced computation is inefficient [41]. [22,21] propose to detect cheating and mis-computation at the expense of data privacy, but they rely on probabilistic checking and require the client to pre-compute the results or the delegate to commit certain values. Wang et al. [41,42] propose practical methods to outsource linear programming to the cloud. However, they consider a single data owner and delegate, as opposed to CloudMine’s multi-party model.

Finally, existing works on security of outsourced databases focus on data privacy [34], query freshness [30,23] and query completeness [28]. These works complement the protocols we described in Section 4 (which deal with data privacy and query completeness).

7 Conclusions and Future work

In this paper, we have described a cloud-based service, named CloudMine, which allows multiple data owners to carry out analytics over their joint data in a privacy-preserving manner. The computation is outsourced to a number of independent clouds (or delegates). CloudMine protects data privacy and ensures correctness of the computation against the standard semi-honest model of the data owners, and against the curious-and-lazy delegate model. CloudMine supports three analytic functions: secure sum, secure set union and intersection, and secure scalar product. These primitives can be used to implement a wide
range of complex data mining algorithms. We demonstrated this by showing how a simple instance of CloudMine (the secure sum service) can be used in a hybrid cloud environment for the classification, association rule mining and clustering algorithms. We discussed the mechanisms designed to ensure privacy when the data is stored in a public cloud. Finally, we implemented a prototype of CloudMine’s secure sum service and evaluated the performance of the service as a stand-alone application and as part of complex data mining applications. The results demonstrate the service’s practical performance, and show that it provides privacy with little cost to the overall performance for workloads that are inherently computationally intensive.

Our current prototype has not implemented the protocols for bootstrapping the CloudMine service. Dynamic group membership may affect the service and its applications in interesting ways. Incorporating and evaluating these protocols, and optimization of the overall implementation are parts of our immediate plan for future work. We also plan to implement the protocols for secure set and scalar product services. For the former, particularly, we intend to investigate how existing protocols for private set intersection (which scale better than our current protocol) can be modified to work in our delegate settings. Once being equipped with these higher-level primitives, we can start looking at more complex applications such as collaborative filtering. Additionally, we plan to explore if the automated scaling features offered by some cloud platforms could improve the performance of the service, especially under intensive workloads. Finally, we would like to incorporate differential privacy techniques into the service and investigate the maximum privacy budget needed to realize any given data mining algorithm.

References

1. Crypto++ library 5.6.1. www.cryptopp.com
2. National oceanic and atmospheric administration. www.noaa.gov
3. Resmap, earth image source. www.resmap.com
4. Uci machine learning repository. archive.ics.uci.edu/ml/datasets.html
5. xignite: on demand financial market data. xignite.com
6. Emmanuel A. Abbe, Amir E. Khandani, and Andrew W. Lo. Privacy-preserving methods for sharing financial risk exposures. http://arxiv.org/abs/1111.5228 Nov 2011.
7. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In 20th International conference on very large databases, pages 487–99, 1994.
8. Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. ACM SIGMOD Record, 2000.
9. Amazon. Amazon Elastic Compute Cloud (EC2). aws.amazon.com/ec2
10. Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. Order-preserving symmetric encryption. In EUROCRYPT’09, 2009.
11. Mike Burmester and Yvo Desmedt. A secure and scalable group key exchange system. Information Processing, 2005.
12. Yangyi Chen, Bo Peng, Xiaofeng Wang, and Haixu Tang. Large-scale privacy-preserving mappings of human genomic sequences on hybrid clouds. In NDSS, 2012.

13. Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. SIGKDD Explorations Newsletter, 4(2), 2002.

14. Wenliang Du, Jing Liu, Manish Mangal, and Mummooorthy Murugesan. Uncheatable grid computing. In ICDCS, 2004.

15. Yitao Duan, John Canny, and Justin Zhan. P4p: practical large-scale privacy-preserving distributed computation robust against malicious users. In USENIX Security Symposium, 2010.

16. Cynthia Dwork. Differential privacy. In ICALP, 2006.

17. Stephen Eubank, Hasan Guclu, V.S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, Zoltan Toroczkai, and Nan Wang. Modelling disease outbreaks in realistic urban social networks. Nature, (429):180–84, 2004.

18. Eucalyptus. Blending clouds: a blueprint for your hybrid future. http://www.eucalyptus.com/resources/whitepapers/blending-clouds-blueprint-your-hybrid-future

19. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In CRYPTO’10, August 2010.

20. Edward L. Glaeser and Matthew E. Kahn. Sprawl and Urban Growth, volume 4, chapter 56. Elsevier, 2003.

21. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Symposium of Theory of Computing, STOC’08, 2008.

22. Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In CT-RSA, pages 425–40.

23. Michael T. Goodrich, Roberto Tamassia, and Andrew Schwerin. Implementation of an authenticated dictionary with skip lists and communicative hashing. In DARPA Information survivability conference and exposition, pages 68–82, 2001.

24. Mohammad Hajjat, Xin Sun, Yu-Wei Eric Sung, David Maltz, Sanjay Rao, Kunwadee Sripanidkulchai, and Mohit Tawarmalani. Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In SIGCOMM, 2010.

25. Yang Huang, David Evans, and Jonathan Katz. Private set intersection: are garbled circuits better than custom protocols. In NDSS, 2012.

26. Klaus Kursawe, George Danezis, and Murkulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. Technical report, Microsoft Research, 2011.

27. Angchao Le and Feifei Li. Query access assurance in outsourced databases. IEEE Transactions on services computing, 2012.

28. Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Authenticated index structures for aggregation queries. Transactions on information and system security, 13(4), 2010.

29. Machine Learning Group, Uni of Waikato. Data mining software in java. www.cs.waikato.ac.nz/ml/weka

30. R. Merkle. Secrecy, authentication and public key systems. PhD thesis, Dept. of Electrical Engineering, Stanford University, 1979.

31. Arjun Narayan and Andreas Haeberlen. Djoin: Differentially private join queries over distributed databases. In OSDI, pages 149–62, 2012.

32. Pascal Paillier. Public-key cryptosystems based on composit degree residuosity classes. In EUROCRYPT, pages 223–38, 1999.
33. Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *34th IEEE Symposium on Security and Privacy*, 2013.

34. Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: a practical encrypted relational dbms. Technical Report MIT-CSAIL-TR-2011-005, CSAIL, MIT, 2011.

35. Vihbor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, pages 735–46, 2010.

36. Elaine Shi, T-H. Hubert Chan, Eleanor Rieffel FxPal, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *Network and Distributed System Security Symposium*, 2011.

37. Radu Sion. Query execution assurance for outsourced databases. In *31st VLDB Conference*, pages 601–12, 2005.

38. Byung Chul Tak, Bhuvan Urgaonkar, and Anand Sivasubramaniam. To cloud or not to cloud? musing on cost of cloud computing. In *3rd USENIX conference on Hot topics in cloud computing*, 2011.

39. Osman Ugus, Dirk Westhoff, Ralf Laue, Abdulhadi Shoufan, and S. A. Huss. Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor. *CoRR*, 2009.

40. Jaideep Vaidya and Chirs Clifton. Privacy-preserving k-means clustering over virtually partitioned data. In *SIGKDD*, 2003.

41. Cong Wang, Kui Ren, and Jia Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM’11*, 2011.

42. Cong Wang, Kui Ren, Jia Wang, and Karthik Mahendra Urs. Harnessing the cloud for securely solving large systems of linear equations. In *ICDCS*, 2011.

43. Yi Wei and M. Brian Blake. Service-oriented computing and cloud computing: challenges and opportunities. *Internet Computing*, 14(6):62–75, 2010.

44. Zhiqiang Yang and Rebecca N. Wright. Privacy-preserving computation of bayesian networks on vertically partitioned data. *IEEE Transaction on Data Knowledge Engineering*, 2006.

45. Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, 1982.

46. Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen, XiaoFeng Wang, and Yaoping Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *CCS*, 2011.