Abstract—Nowadays industries are collecting a massive and exponentially growing amount of data that can potentially promote business innovations. However, it is challenging for resource-limited clients to analyze their data in a cost-effective and timely way as the data volume keeps growing. With cloud computing, one feasible solution is to analyze the massive data by outsourcing them to the cloud. Nonetheless, clients’ data may contain private information that needs to be kept secret. In this paper, we design a secure, efficient, and verifiable outsourcing protocol specifically for geometric programming, which is one of the most fundamental problems in data analysis with many applications. In particular, a secure and efficient transformation scheme is used to encrypt the original geometric programming problem at the client side and protect its privacy before offloading it, and the gradient projection method is employed to solve the encrypted geometric programming problem in the cloud side. Experiments are conducted on both Amazon Elastic Compute Cloud (EC2) and a laptop to evaluate performance of the designed outsourcing protocol, and the results show the feasibility and efficiency of the protocol.

I. INTRODUCTION

Mathematical optimization has found applications in various areas, such as computer science [1], signal processing [2], and economics [3]. It is defined as maximizing or minimizing an objective function by choosing input values from a feasible region. A broad class of mathematical optimization is convex optimization where the objective function is convex, and the feasible region is a convex set [4]. Particularly, if the objective function is geometric and the feasible region is constrained to a system of linear equalities and inequalities, it is called Geometric Programming (GP). GP has a variety of applications, e.g., circuit design [5], power control [6], and information theory [7]. However, solving GP requires many computing resources when the problem scale is large. Thus, it is attractive for a client with limited computing capability to outsource large-scale GP problems to the cloud.

Although outsourcing to the cloud allows a client to solve large-scale GP problems, it also brings some new issues [8], [9]. Privacy is the first issue to be handled, since both the outsourced problems and the solutions of these problems may contain sensitive information that the client does not want to expose to the cloud. To protect the input privacy (i.e., secrecy of the original GP problem), the client has to encrypt the original problem before sending it to the cloud. The output privacy should also be protected, which means the cloud should not be able to infer the solution to the original GP problem.

Verifiability is the second issue to be taken into consideration. The client needs to verify whether the result returned is correct or not. The cloud may return a random result to save computing resources when the outsourced task is highly resource-consuming. Even though the cloud performs faithfully, some inevitable hardware and software bugs in the cloud may also lead to an incorrect result. Thus, without verifiability, the correctness of the result returned by the cloud cannot be guaranteed.

Lastly, efficiency is also an issue that needs to be addressed. It requires that the overhead of the client should be substantially reduced when outsourcing is chosen. Furthermore, the amount of computation performed by the cloud should not be too high compared with the workload of solving the original problem.

In this paper, we develop an efficient and privacy-preserving protocol for outsourcing GP problems. Specifically, we consider a general GP problem. The GP is first converted to a convex dual geometric problem (DGP) by variable substitutions and the Lagrange dual method. Next, the client transforms (i.e., encrypts) the DGP through multiplying the decision variable and constraints by random sparse matrices. We show that the transformed DGP is computationally indistinguishable from the DGP both in value and in structure. Then based on the dual problem theory and the gradient projection method, the cloud solves the transformed DGP, and sends the result to the client, who can then efficiently derive the solution to its original GP and verify the solution. The scheme protects the client’s privacy by letting the cloud operate on the transformed DGP, rather than any original problem formulations.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, it is the first privacy-preserving solution for outsourcing GP problems to the cloud. It consists of a scheme that converts GP to DGP, a transformation scheme that encrypts and protects the DGP before offloading it to the cloud, and a scheme to solve the transformed DGP at the cloud side.
We formally prove that the transformed DGP problem can protect the client’s data privacy. In particular, the transformed DGP has the property of computationally indistinguishability.

We implement the proposed solution on the Amazon EC2 platform and a laptop. Experimental results show that the proposed secure outsourcing scheme can achieve significant time savings for the client.

The rest of the paper is organized as follows. Section II presents problem formulation and system overview. Section III introduces the secure transformation scheme. Section IV presents in detail the algorithm for solving the outsourced DGP. Performance evaluation is presented in Section V. Section VI reviews related work. Section VII concludes this paper.

II. PROBLEM FORMULATION

A. Geometric Problem Formulation

GP is a class of mathematical optimization and has the following form:

Minimize $f_0(x) = \sum_{j=1}^{N_0} c_j \prod_{i=1}^{n} x_i^{a_{ij}}$

subject to $g_k(x) = \sum_{j=N_{k-1}+1}^{N_k} c_j \prod_{i=1}^{n} x_i^{a_{ij}} \leq 1, \quad k = 1, \ldots, m$

$x > 0$ (1)

where $x = (x_1, x_2, \ldots, x_n)^t$ is the optimization variable, and $N_k$, for $k = 0, 1, \ldots, m$, and $c_j$, for $j = 1, 2, \ldots, N_m$ represent the number of terms in each function and term coefficients, respectively. Problem (1) is said to be a GP problem because both the objective function and the constraint functions can be expressed as the sum of posynomial terms [5].

GP problems arise frequently in engineering applications. For example, in a power control problem, each decision variable $x_i$ represents the positive transmitting power level, and the interface power of each transmitter/receiver pair can be expressed as posynomial terms. This problem can be formulated as a GP problem where decision variables are subject to practical constraint functions. Another example is semiconductor device operations. In particular, the objective function is to choose the doping profile to minimize the base transit time, while the doping profile value is bounded and consists of posynomial terms. This problem can also be formulated as a GP problem.

B. System Overview

As shown in Fig. 1, we consider an asymmetric two-party computing architecture, where a local client is resource-limited while a remote cloud server has abundant computing resources. The client is unable to solve the original GP problem with local computational resources in an acceptable amount of time. Thus, the client outsources the GP problem to the cloud after making certain transformations to it (it is called transformed GP after transformations). Then, the cloud server solves the transformed GP and sends the solution of the transformed GP back to the client, who will verify and decrypt the solution for the original GP problem. The transformation from the original GP to the transformed GP consists of two steps. The first step converts GP to DGP, and the second step transforms (or encrypts) the DGP to protect it from the cloud server.

C. Threat Model

We assume a malicious cloud server. In particular, the cloud attempts to learn the client’s original GP problem from the outsourced problem and the returned results of its own computations. Additionally, the cloud may not follow the proposed protocol and return incorrect results.

To securely outsource the computation of the GP problem, we adopt the concept of computational indistinguishability under a chosen plaintext attack (CPA) [10]. In a matrix, we notice that the elements’ values and positions both carry private information. In the following, we formally define computational indistinguishability under a CPA for these two types of private information, respectively.

We first present the definition of a pseudorandom function as follows, which will be employed to perform matrix transformations with CPA security.

**Definition 1.** Let $X = \{X_n\}_{n \in N}$ be a probability ensemble and $Y = \{Y_n\}_{n \in N}$ be a truly random function. We say $X$ is a
where $i$ and $j$ are variables.

This definition can be extended to the case where a distinguisher $D$ has access to multiple elements of the vectors $X$ and $Y$, e.g., when comparing two matrices.

**Definition 2.** Let $R \in \mathbb{R}^{m \times n}$ be a random matrix with elements in its $i$th row sampled from a uniform distribution with interval $[-R_j, R_j]$ for all $j \in [1, n]$. Matrices $R$ and $G \in \mathbb{R}^{m \times n}$ are computationally indistinguishable if for any probabilistic polynomial time distinguisher $D$ there exists a negligible function $\mu$ such that

$$|\Pr[D(X_n) = 1] - \Pr[D(Y_n) = 1]| \leq \mu$$  \hspace{1cm} (2)

where $i, j \in [1, m]$, $g_{ij}$ is the element in the $i$th row and $j$th column of $G$, and $r_{ij}$ is the element in the $i$th row and $j$th column of $R$. Distinguisher $D$ outputs 1 when it finds out $g_{ij}$ is not chosen from matrix $G$ and 0 otherwise.

### III. A Privacy-Preserving Transformation Scheme

To securely outsource a GP problem to the cloud, the client first converts it to a DGP problem as described in Section III-A. Then it transforms the DGP problem using three techniques and offloads the transformed DGP to the cloud. Section III-B, III-C, and III-D describe the three techniques, and Section III-E describes the transformation process based on the three techniques.

**A. The Lagrange Dual Problem**

Since the objective function $f_0$ and constraint functions $g_k$ in GP problem (1) are posynomials and are non-convex in general [4], GP problem (1) is a non-convex optimization problem and it can take exponential time to solve this problem, especially with large-scale decision variables and constraints. In this section, we identify an equivalent dual optimization problem that is convex with only linear constraints, i.e., the dual geometric problem.

In particular, first we transform the original non-convex GP problem with the following variable substitution:

$$y = \log x$$  \hspace{1cm} (4)

Next, from (4) we denote

$$\tau_j = \log a_{ij} = c_j e^{a_j y} \text{ for } j = 1, \ldots, N_m$$  \hspace{1cm} (5)

where $a_j = (a_{j1}, \ldots, a_{jn})^t$ for $j = 1, \ldots, N_m$. Taking a logarithmic transformation of the objective and constraint functions, the original GP problem (1) can be equivalently rewritten as:

Minimize $\log [F(y)]$

subject to $\log [G_k(y)] \leq 0$, $k = 1, \ldots, m$.  \hspace{1cm} (6)

where

$$F(y) = \sum_{j=1}^{N_2} \tau_j$$
$$G_k(y) = \sum_{j=N_k-1+1}^{N_k} \tau_j \text{ for } k = 1, \ldots, m$$  \hspace{1cm} (7)

According to [11], the problem (6) is now a convex programming problem.

In the following, we use the Lagrangian dual approach to solve problem (6). Since the interiority constraint qualification holds, there is no gap between problem (6) and its Lagrangian dual stated below:

$$LD : \text{Maximize } L(y, u)$$
$$\nabla_y L(y, u) = 0$$
$$u \geq 0, y \text{ unrestricted}$$

where the Lagrangian function is

$$L(y, u) = \log [F(y)] + \sum_{i=1}^{m} u_i \log [G_i(y)]$$  \hspace{1cm} (9)

We now define a new variable vector $\delta$ as follows:

$$\delta_k = \frac{\tau_k}{F} \text{ for all } k \in [1, N_0]$$
$$\delta_k = \frac{u_i \tau_k}{G_i} \text{ for all } k \in [N_0 + 1, N_m]$$  \hspace{1cm} (10)

Note that, from Eq. (7) and (10), we have

$$\sum_{k=1}^{N_0} \delta_k = 1$$
$$\sum_{k=N_0+1}^{N_m} \delta_k = u_i \text{ for } i \in [1, m]$$  \hspace{1cm} (11)

According to $\nabla_y L(y, u) = 0$ in the $LD$ problem (8) and Eq. (10), we have

$$\nabla_y L(y, u) = \nabla F(y) + \sum_{i=1}^{m} u_i \frac{\nabla G_i(y)}{G_i(y)}$$

$$= \frac{1}{F(y)} \sum_{k=1}^{N_0} \tau_k a_k + \sum_{i=1}^{m} u_i \frac{\sum_{k=N_0+1}^{N_m} \tau_k a_k}{G_i(y)}$$

$$= \sum_{k=1}^{N_m} \delta_k a_k = 0$$  \hspace{1cm} (12)
From Eq. (5), (10), and (11), the term \( u_i \log[G_i(y)] \) in Eq. (9) can also be rewritten as:

\[
\begin{align*}
    u_i \log[G_i(y)] &= u_i \log(u_i) + u_i \log \left( \frac{G_i}{u_i} \right) \\
    &= u_i \log(u_i) + \sum_{k=N_0+1}^{N_m} \delta_k \log \left( \frac{\tau_k}{\delta_k} \right) \\
    &= u_i \log(u_i) + \sum_{k=N_0+1}^{N_m} \delta_k \log \left( \frac{c_k e^{a_k y}}{\delta_k} \right) \\
    &= u_i \log(u_i) + \sum_{k=N_0+1}^{N_m} \delta_k a_k y
\end{align*}
\]

(13)

Similarly, we have

\[
\log[F(y)] = \sum_{k=0}^{N_0} \delta_k \log \left( \frac{c_k}{\delta_k} \right) + \sum_{k=0}^{N_0} \delta_k a_k y
\]

(14)

Thus, from Eq. (13) and (14), the objective function (9) can be rewritten as:

\[
L(\delta, u) = \sum_{k=1}^{N_m} \delta_k \log \left( \frac{c_k}{\delta_k} \right) + \sum_{i=1}^{m} u_i \log(u_i)
\]

(15)

We finally use Eq. (11), (12), and (15) to replace the LD problem (8) with the following dual geometric program (DGP) in the variables \((\delta, u)\):

**DGP**: Maximize \( \sum_{k=1}^{N_m} \delta_k \log \left( \frac{c_k}{\delta_k} \right) + \sum_{i=1}^{m} u_i \log(u_i) \)

subject to \( A \delta = 0 \)

\[
\begin{align*}
    &\sum_{k=0}^{N_0} \delta_k = 1 \\
    &\sum_{k=N_0+1}^{N_m} \delta_k - u_i = 0 \quad \text{for } i \in [1, m] \\
    &\delta \geq 0 \\
    &u \geq 0
\end{align*}
\]

(16)

where \( A = [a_1 \ a_2 \ \ldots \ a_k] \in R^{n \times N_m} \).

Note that the DGP problem (16) is a convex programming problem with linear constraints. We denote the optimal solution to (16) as \((\delta^*, u^*)\).

Since the DGP problem (16) is convex and the affine constraints are feasible, the strong duality holds [11] and according to Eq. (4), (5), and (10), we have that

\[
\begin{align*}
    y^* &= \left( A^T \right)^{-1} \log(\delta^*/u^*c) \\
    x^* &= e^{y^*}
\end{align*}
\]

(17)

where \( c = (c_1, c_2, \ldots, c_{N_m})^T \). That is, we can use the result of the DGP problem (16) to recover the result of the original GP problem (1).

**B. Privacy-Preserving Vector Addition**

The client can efficiently hide a private variable vector by adding a randomly generated vector to it. Specifically, a private variable vector \( x = (x_1, x_2, \ldots, x_n)^T \) can be encrypted as follows:

\[
y = x + r
\]

(18)

where \( y_i = x_i + r_i \) for any \( i \in [1, n] \), and \( y_i, x_i, \) and \( r_i \) are the \( i \)th element of vector \( y, x \) and \( r \), respectively. We assume that \( x_i \) is within the range \([-K, K] \), where \( K = 2^{l(i > 0)} \) is a positive constant. Additionally, vector \( r \in R^{n \times 1} \) is randomly generated with its elements subject to uniform distribution and the corresponding probability density function is expressed as:

\[
f(r_i) = \begin{cases} 
\frac{1}{2c} & -c \leq r_i \leq c \\
0 & \text{otherwise}
\end{cases}
\]

(19)

where \( c = 2^{l+p(p > 0)} \) is a positive constant, and \( r_i, i \in [1, n] \) is the \( i \)th element of vector \( r \). Next we will obtain the following theorem that vectors \( r \) and \( y \) are computationally indistinguishable.

**Theorem 1.** Let \( r \) be a random vector with elements sampled from a uniform distribution with interval \([-c, c] \). Then vectors \( r \) and \( y = x + r \) are computationally indistinguishable.

**Proof.** According to Definition 1, we need to prove that any probabilistic polynomial time distinguisher \( D \) cannot distinguish \( y_i \) from \( r_i \) for any \( i \in [1, n] \) except with negligible success probability, where \( y_i \) and \( r_i \) are the \( i \)th element of vector \( y \) and \( r \) respectively. The best strategy for a polynomial time distinguisher \( D \) when presented with a sample \( y_i \) is to return \( b \leftarrow \{0, 1\} \) with equal probability if \(-c \leq y_i \leq c \), and 1 if \( y_i < -c \) or \( y_i > c \). Therefore, the success probability of the distinguisher with the input being \( y_i = x_i + r_i \) is given by

\[
Pr[D(y_i) = 1] = \frac{1}{2} Pr[-c \leq x_i + r_i \leq c] + Pr[x_i + r_i < -c] + Pr[x_i + r_i > c]
\]

\[
= \frac{1}{2} (1 - Pr[x_i + r_i < -c] - Pr[x_i + r_i > c]) + Pr[x_i + r_i < -c] + Pr[x_i + r_i > c]
\]

(20)

Recall that \( x_i \) is within the range \([-K, K] \), and \( r_i \) is sampled from a uniform distribution specified by (19). We have that

\[
Pr[x_i + r_i > c] = Pr[r_i > c - x_i] \leq Pr[r_i > c - K] = \frac{K}{2c}
\]

(21)

Similarly, we find that \( Pr[x_i + r_i < -c] \leq \frac{K}{2c} \). Consequently, we have that the success probability of the distinguisher \( D \) is bounded as follows:

\[
Pr[D(y_i) = 1] = \frac{1}{2} + \frac{K}{2c}
\]

(22)

On the other hand, when the input is \( r_i \), obviously we can obtain that:

\[
Pr[D(r_i) = 1] = \frac{1}{2}
\]

(23)
According to Eq. (2), for any $i \in [1, n]$, we get that

$$|Pr[D(y_i) = 1] - Pr[D(r_i) = 1]| \leq \frac{K}{2c}$$  \hspace{1cm} (24)

Note that $K = 2^l$ and $c = 2^{l+p}$. Hence, we have

$$\mu(p) = \frac{K}{2c} \leq \frac{2^l}{2^{l+p+1}} = \frac{1}{2^{p+1}}$$  \hspace{1cm} (25)

which is a negligible function for large $p$. This concludes the proof. \hfill \square

C. Privacy-Preserving Matrix Multiplication

The client can efficiently encrypt the values of a private problem matrix by performing sparse random matrix multiplications. In particular, a private matrix $\mathbf{H} \in R^{m \times n}$ can be efficiently encrypted by performing the following multiplications:

$$\tilde{\mathbf{H}} = DHF$$  \hspace{1cm} (26)

Here $\mathbf{D} \in R^{m \times m}$ is a diagonal matrix defined as

$$d_{i,j} = \begin{cases} v_i & i = j \text{ for } i, j \in [1, m] \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (27)

where the value $v_i$, for $i \in [1, m]$, is generated based on the uniform distribution defined in (19). $\mathbf{F} \in R^{n \times n}$ is also a diagonal matrix with elements being arbitrary positive constant $M$. Consequently, the elements of $\tilde{\mathbf{H}}$ in (26) are given by

$$\tilde{h}_{i,j} = d_{i,j}h_{i,j}f_{j,j} = v_i h_{i,j}M$$  \hspace{1cm} (28)

Assume that the element values of $\mathbf{H}$ are within the range $[-T, T]$. Next we can arrive at Theorem 2 that the encrypted private matrix $\tilde{\mathbf{H}}$ and a random matrix $\mathbf{R}$ with elements sampled from a uniform distribution are computationally indistinguishable.

Theorem 2. Let $\mathbf{R} \in R^{m \times n}$ be a random matrix with elements in its $j$th column sampled from a uniform distribution with interval $[-c, c]$, for $j \in [1, n]$. Matrices $\mathbf{R}$ and $\tilde{\mathbf{H}}$ are computationally indistinguishable.

Proof. According to Definition 2, we need to prove that $r_{i,j}$ and $\tilde{h}_{i,j}$, for $i \in [1, m], j \in [1, n]$, are computationally indistinguishable for matrices $\mathbf{R}$ and $\tilde{\mathbf{H}}$ to be computationally indistinguishable. Specifically, we prove that any polynomial time distinguisher $D$ cannot distinguish $\tilde{h}_{i,j}$ from $r_{i,j}$, for $i \in [1, m], j \in [1, n]$, except with negligible success probability.

The distinguisher $D$ is defined in the same way as in Theorem 1. Therefore, the success probability of the distinguisher $D$ is given by

$$Pr[D(\tilde{h}_{i,j}) = 1] = \frac{1}{2}Pr[-c \leq \tilde{h}_{i,j} \leq c]$$
$$+ Pr[\tilde{h}_{i,j} < -c] + Pr[\tilde{h}_{i,j} > c]$$
$$= \frac{1}{2}(1 - Pr[\tilde{h}_{i,j} < -c] - Pr[\tilde{h}_{i,j} > c])$$
$$+ Pr[\tilde{h}_{i,j} < -c] + Pr[\tilde{h}_{i,j} > c]$$  \hspace{1cm} (29)

where

$$Pr[\tilde{h}_{i,j} > c] = Pr[v_i h_{i,j} M > c]$$
$$= Pr[v_i h_{i,j} > \frac{c}{M}]$$
$$\leq \gamma Pr[v_i > \frac{c}{MT}] + (1 - \gamma) Pr[v_i < \frac{c}{MT}]$$
$$= \frac{1}{2} - \frac{1}{2MT}$$  \hspace{1cm} (30)

the parameter $\gamma$ is the probability of the element $\tilde{h}_{i,j}$ being positive and $1 - \gamma$ is $\tilde{h}_{i,j}$ being negative. Similarly, we find that $Pr[\tilde{h}_{i,j} < -c] \leq \frac{1}{2} - \frac{1}{2MT}$. Consequently, we have that the success probability of distinguisher $D$ is bounded as follows:

$$Pr[D(\tilde{h}_{i,j}) = 1] = 1 - \frac{1}{2MT}$$  \hspace{1cm} (31)

On the other hand, we can easily obtain that $Pr[D(r_{i,j}) = 1] = \frac{1}{2}$.

According to Eq. (3), for $i \in [1, m], j \in [1, n]$, it follows that

$$|Pr[D(\tilde{h}_{i,j}) = 1] - Pr[D(r_{i,j}) = 1]| \leq \frac{MT - 1}{2MT}$$  \hspace{1cm} (32)

Note that $M$ is an arbitrary positive constant, we have

$$\beta(M) = \frac{MT - 1}{2MT}$$  \hspace{1cm} (33)

Thus, $\beta(M)$ can be guaranteed as a negligible function when $MT$ approaches 1. This concludes the proof. \hfill \square

D. Privacy-Preserving Matrix Permutation

Although the matrix transformation in Eq. (26) hides the values of the elements in $\mathbf{H}$, it still reveals the original positions of the non-zero elements, i.e., $\mathbf{H}$’s structure, which is also private. Next, we design secure permutations that can hide $\mathbf{H}$’s structure by randomly reordering the rows and columns of $\mathbf{H}$.

The client applies the random permutations as follows:

$$\tilde{\mathbf{H}} = E \tilde{\mathbf{H}} U$$  \hspace{1cm} (34)

where $\mathbf{E} \in R^{m \times m}$ and $\mathbf{U} \in R^{n \times n}$ are random permutation matrices, and their elements are defined by

$$e_{i,j} = \delta_{\pi(i),j} \forall i \in [1, m], j \in [1, m]$$
$$u_{i,j} = \delta_{\pi(i),j} \forall i \in [1, n], j \in [1, n]$$  \hspace{1cm} (35)

where $i$ and $j$ are the row and column indexes, respectively. The random permutation function $\pi(\cdot)$ maps an original index $i \in \{1, 2, \cdots, n\}$ to its permuted index within the same range. Besides, the Kronecker delta function as defined in [12] is given by

$$\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$  \hspace{1cm} (36)

The details of generating random permutation matrices in Eq. (34) are summarized in Algorithm 1.

In addition, the client is able to recover the original matrix $\mathbf{H}$ by applying the following inverse permutations:
Algorithm 1 Random permutation matrix generation

Input: Initial index set \( N = \{1, 2, \ldots, n\} \)

Output: Random permutation matrix \( P \)

1. Set \( \pi = I_n \); (identical permutation)
2. for \( i = n \) down to 2 do
3. Set \( j \) to a random integer with \( 1 \leq j \leq i \);
4. Swap \( \pi[i] \) and \( \pi[j] \) in set \( N \);
5. end for
6. for \( i = 1 \) to \( n \) do
7. for \( j = 1 \) to \( n \) do
8. \( \pi(i) \) outputs the \( i \)th element in set \( N \);
9. \( \delta_{\pi(i),j} \) outputs value based on Eq. (36);
10. Set \( P(i,j) = \delta_{\pi(i),j} \);
11. end for
12. end for
13. return \( P \);

\[ \hat{H} = E^T H U^T \]  

To get this result, the orthogonal property of permutation matrices are applied, i.e., \( E^T E = I \) and \( U^T U = I \), where \( I \) is the identity matrix.

E. The Overall Transformation

As shown in Section III-A, the client can transform the original GP problem (1) to the DGP problem (16). For convenience, we first rewrite the DGP problem in (16) as the following form:

Minimize \( D(z) \)
subject to \( Wz = b \)
\( z \geq 0 \)  

where

\[ D(z) = -L(\delta, u) = -\sum_{k=1}^{N_m} \delta_k \log \frac{c_k}{\delta_k} - \sum_{i=1}^{m} u_i \log(u_i) \]

\[ W = \begin{bmatrix} A & 0 \\ B & -I \end{bmatrix} \in R^{(n+m+1) \times (N_m+m)} \]

\[ B = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \]

\( b = (0, 1, 0, \ldots, 0)^T \in R^{(n+m+1) \times 1} \)

Thus, to protect the sensitive information of the coefficient matrix, the client applies the matrix transformations (26) and (34) to \( W \) in (38) as follows:

\[ \hat{W} = V W T \]  

where \( V \) is formed by a random permutation matrix and a random diagonal matrix, i.e., \( V = ED \), and \( T \) formed by a diagonal matrix of arbitrary positive constant and a random permutation matrix, i.e., \( T = FU \).

Furthermore, to protect the privacy of decision variable vector \( z \), the vector transformation (18) is applied in the following:

\[ z = T^{-1}(z + r) \]  

where \( r \in R^{(N_m+m) \times 1} \) is a random vector. Based on Eq. (39) and (40) we have

\[ \hat{b} = \hat{W}z = V(b + Wr) \]

Now we can transform the problem (38) into the following privacy-preserving problem:

Minimize \( D(\hat{z}) \)
subject to \( \hat{W}z = \hat{b} \)
\( z \geq T^{-1}r \)

Next, the encrypted problem (42) will be sent to the cloud server.

IV. SOLVING THE OUTSOURCED PROBLEM

In this section, we describe an efficient algorithm to solve the offloaded transformed DGP problem derived in Section III. Specifically, we employ the gradient projection method (GPM), an iterative method for nonlinear convex optimization problems to solve the transformed DGP problem.

For optimization problems, the search direction of fastest descent is the negative gradient of the objective function. However, moving along the negative gradient may lead to violating the constraint functions. The main idea of the GPM is to project the negative gradient in a way that improves the objective function while not violating any constraints.

In particular, let us first consider the following convex optimization problem

Minimize \( f(z) \)
subject to \( A'z = b' \)
\( z \geq 0 \)  

Given a feasible point \( z \), the moving direction of steep descent is \(-\nabla f(z)\). However, moving along \(-\nabla f(z)\) may violate feasibility of the solutions. To this end, the moving direction \( d \) is projected so that \( d = -P\nabla f(z) \), where \( P \) is a suitable projection matrix [11]. The following Theorem 3 [11], [13] will provide a way to find an improving feasible direction \( d \).

**Theorem 3.** Consider the optimization problem in (43), and suppose \( f(z) \) is differentiable at the point \( z \). Let projection matrix \( P \) be of the form \( P = I - A' (A' A'^T)^{-1} A \). Then \( d = -P \nabla f(z) \) is an improving feasible direction if \( d \neq 0 \), and \( z \) is a Karush–Kuhn–Tucker (KKT) point if \( d = 0 \).

**Proof.** First of all, it is noted that

\[ A'd = -A'P \nabla f(z) \]
\[ = -A'(I - A' (A' A'^T)^{-1} A') \nabla f(z) \]
\[ = 0 \]
Thus \( d \) is a feasible direction. In addition, according to the properties of projection matrix, i.e., \( P = P^T, P = P^2 \) [11], we have

\[
\nabla f(z)^T d = -\nabla f(z)^T P \nabla f(z) = -\nabla f(z)^T P^T P \nabla f(z) = -\|P \nabla f(z)\|^2 < 0
\]

Thus \( d \) is also an improving direction if \( d \neq 0 \). From Eq. (44), (45), we have that \( d \) is an improving feasible direction, which completes the proof.

Once an improving feasible direction \( d \) is found, the optimal point of the objective function \( f(z) \) can be approached in the following iterative way:

\[
z_{k+1} = z_k + \lambda_k d_k
\]

where \( k \) is the iteration index, and the upper bound of \( \lambda_k \) defined as \( \lambda_{\text{max}} \) is given by:

\[
\lambda_{\text{max}} = \left\{ \min_{1 \leq i \leq m} \left\{ \frac{-z_{ik}}{d_{ik}} : d_{ik} < 0 \right\} \right\} \quad \text{for all } k \geq 0
\]

where \( z_{ik} \) and \( d_{ik} \) are the \( i \)th elements of \( z_k \) and \( d_k \), respectively. Hence, the value of \( \lambda_k \) is determined by the following line search problem:

\[
\begin{align*}
\text{Minimize} & \quad f(z_k + \lambda d_k) \\
\text{subject to} & \quad 0 \leq \lambda \leq \lambda_{\text{max}}
\end{align*}
\]

Next, since problem (43) consists of a convex objective function with only linear constraints, the GPM converges. The proof for the convergence of the GPM is omitted here due to the space limitation.

The complete procedure is summarized in Algorithm 2.

**Algorithm 2** Secure algorithm for solving outsourced large-scale GP

**Input:** Starting point \( z_0 \) that \( Wz_0 = \bar{b} \)

**Output:** Optimal point \( \bar{z}^* \) for problem (42)

1: Initialize \( k = 0 \);
2: Compute \( P \) and \( d_0 \) from Theorem 3;
3: Let \( d = d_0 \);
4: while \( d \neq 0 \) do
5: \quad Compute \( \lambda_{\text{max}} \) using Eq. (47);
6: \quad Solve the line search for \( \lambda_k \):
7: \quad \lambda_k = \arg \min \{ f(z_k + \lambda d_k) \} ;
8: \quad 0 \leq \lambda \leq \lambda_{\text{max}}
9: \quad k = k + 1;
10: \quad Compute \( d_k \) from Theorem 3;
11: end while
12: Let \( \bar{z}^* = \bar{z}_k \), and \( \bar{z}^* \) is a KKT point;
13: return \( \bar{z}^* \);

As stated in Algorithm 2, the cloud server continues the iteration until the secure outsourcing algorithm converges to the KKT point \( \bar{z}^* \). Once the cloud server determines that the algorithm has converged, it sends \( \bar{z}^* \) back to the client, who verifies the correctness of the returned result based on KKT conditions. If the returned result satisfies the KKT conditions, the client determines the returned result is correct and compute the solution to the DGP problem (16) as follows:

\[
z^* = T \bar{z}^* - r
\]

From (38), we have \( z^* = (\delta^*, u^*)^T \). Thus the solution to the original GP problem (1) can be obtained by following Eq. (17).

**V. PERFORMANCE EVALUATION**

This section presents the computational complexity of the proposed solution and experiment results.

**A. Computational Complexity**

The computational cost at each step is analyzed as follows. First, the client transforms the original GP problem to the DGP problem, which takes \( \mathcal{O}(m^2) \) computational cost. Then, the client employs the privacy-preserving matrix transformation to encrypt the coefficient matrices, which induces a computational complexity of \( \mathcal{O}(2m^2 + 4m) \). Next, the cloud server solves the outsourced problem, which needs computation of \( \mathcal{O}(2m^2 n + m^3) \). Lastly, the client recovers the optimal solution based on the returned solution from the cloud server, resulting in a computational complexity of \( \mathcal{O}(2mn) \).

To summarize, the proposed privacy-preserving outsourcing protocol requires computational complexity of \( \mathcal{O}(\max\{mn, m^2\}) \) at the client side and \( \mathcal{O}(\max\{m^2 n, m^3\}) \) at the cloud side.

**B. Experiment Results**

In the following, we evaluate the performance of the proposed privacy-preserving outsourcing protocol for GP through experiments. We implemented the proposed protocol in a real-world scenario. The client side was implemented on a laptop with a dual-core 2.3 GHz CPU, 8GB RAM, and 256 GB solid state drive. The cloud side was implemented on the Amazon Elastic Compute Cloud (EC2) with a number of computing nodes each of 16GB memory. Both the client-side and the cloud-side computations were implemented by Matlab R2018a. The GP problems used in evaluations are randomly generated. Each data point presented below is the average of 20 runs with different randomness seeds.

We first measure the computing time of the proposed protocol at both the client and the cloud side. In these experiments, we only used 12 cloud nodes. Table I shows the results. It can be observed that the client can complete the needed computations very quickly, even for large-size GPs. For example, it only takes the client 2.2s to complete the computation for GP problem size 8000 (i.e., parameter \( m \)). The computing time of the cloud server to obtain the optimal solution is much longer than the client due to the complex nature of solving the problem. Not surprisingly, the computation time at both the client side and the cloud side increases when the problem size increases.
### TABLE I: Computing Time (12 cloud nodes, 16GB memory per node)

<table>
<thead>
<tr>
<th>Number of Variables</th>
<th>Solving GP by Client</th>
<th>Client’s Computing in Our Solution</th>
<th>Cloud’s Computing in Our Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>21.6s</td>
<td>0.31s</td>
<td>6.7s</td>
</tr>
<tr>
<td>2,000</td>
<td>43.9s</td>
<td>0.39s</td>
<td>13.1s</td>
</tr>
<tr>
<td>3,000</td>
<td>77.6s</td>
<td>0.52s</td>
<td>29.2s</td>
</tr>
<tr>
<td>4,000</td>
<td>137.7s</td>
<td>0.65s</td>
<td>48.1s</td>
</tr>
<tr>
<td>5,000</td>
<td>254.9s</td>
<td>0.86s</td>
<td>79.6s</td>
</tr>
<tr>
<td>6,000</td>
<td>355.7s</td>
<td>1.03s</td>
<td>98.4s</td>
</tr>
<tr>
<td>7,000</td>
<td>536.1s</td>
<td>1.34s</td>
<td>142.7s</td>
</tr>
<tr>
<td>8,000</td>
<td>1352.7s</td>
<td>2.17s</td>
<td>301.8s</td>
</tr>
</tbody>
</table>

Fig. 2: Computing time of cloud server with different node sizes

Subsequently, we examine the computing saved for the client by our outsourcing protocol. As shown in Table I, we compare the computing time of the client when it solves GP by itself with that when it outsources GP to the cloud. The saved computing increases dramatically as the problem size increases. For example, the saving can reach 624-fold for problem size 8000, indicating a 99.8% reduction in computing at the client. This validates the efficacy of our proposed protocol for the client.

The overall delay for solving GP is also much shorter when outsourcing it to the more powerful cloud. When the client solves GP by itself, the delay is the time shown in the second column of Table I. When the client outsources GP to the cloud using our protocol, the overall delay is the client’s computing time (the 3rd column of Table I) plus the cloud’s computing time (the 4th column of Table I). Here when delay is concerned the communication time between the client and the cloud is neglected since it is much shorter than the computing time. Then from Table I, it can be seen that the overall problem resolving delay is several times shorter in our solution.

Next, we investigate the computing time at the cloud server when a varying number of nodes are used. The results are shown in Fig. 2. It can be observed that the computing time of the cloud server decreases as the number of nodes used grows. For example, the computing time is as low as 302s when 12 nodes are used compared with about 520s when only 4 nodes are used. The computing time of the cloud server can be further shortened by using more cloud nodes.

Lastly, we measure the computing time of the cloud server with different node memory sizes. As it can be seen from Fig. 3, the computing time at the cloud server decreases as the node memory increases. For example, when node memory size increases from 8GB to 32GB, the computing time decreases from 372s to 287s for problem size 8000, indicating a huge time saving at the cloud side. We also noticed that when the memory size increases from 16GB to 32GB, the reduction in computing time is only a little compared with the reduction when the memory size increases from 8GB to 16GB. That is because 16GB node memory is already good for the maximum problem size experimented, i.e., 8000. When the problem size is larger, the reduction in computing time when node memory increases from 16GB to 32GB should be more.

### VI. RELATED WORK

In recent years, researchers have developed many protocols for privacy-preserving cloud computing.

Based on fully homomorphic encryption (FHE) [14], Gennaro et al. [15] proposed a privacy-preserving outsourcing algorithm by employing fully homomorphic encryption (FHE). Wang et al. [16] developed an iterative algorithm to solve linear systems of equations, where a client transforms and encrypts the coefficient matrix using homomorphic encryption, and the cloud carries out computations on ciphertexts. Hu et al. [17] designed secure interactive protocols to distribute the feature extraction computations to two independent cloud servers. However, these algorithms require the client to per-
form extensive data pre-processing and encryption/decryption operations.

Without resorting to homomorphic encryption, Wang et al. [18] presented an efficient algorithm to securely compute histogram of oriented gradients based on matrix transformations. Secure and efficient outsourcing protocols have been proposed to solve large scale non-linear programming problem [19], [20]. Shen et al. [21] developed a secure outsourcing scheme to solve linear algebraic equations. Some secure outsourcing protocols for matrix computation have also been developed, such as matrix inversion [22], matrix multiplication [23], and matrix determinant [24]. Besides, Zhang et al. [25] considered employing matrix digest techniques to securely outsource batch matrix multiplication. However, the privacy-preserving outsourcing algorithm for large-scale GPs has not been studied so far.

VII. CONCLUSION

In this paper, we investigated privacy-preserving outsourcing of large-scale geometric programming problems. To the best of our knowledge, this is the first work to solve GP in cloud computing with privacy protection. We converted the GP problem to the DGP problem, and then employed a transformation scheme to protect the client’s private problem formulation with formally proved security. The gradient projection method was used by the cloud server to solve the outsourced problem at the cloud. Experimental results based on Amazon EC2 showed that the proposed protocol can provide significant time savings for the client.

REFERENCES