Public Cryptography Linear and Non Linear Programming Solver in Cloud Computing

S. A. Phatangare, R.A. Deshmukh, P. K. Deshmukh
Rajarshri Shahu College of Engineering, Pune

Abstract
Public-key cryptography refers to a cryptographic system requiring two separate keys, one of which is secret and one of which is public. Cloud Computing has great potential of providing robust computational power to the society at reduced cost. It enables customers with limited computational resources to outsource their large computation workloads to the cloud, and economically enjoy the massive computational power, bandwidth, storage, and even appropriate software that can be shared in a pay-per-use manner. Despite the tremendous benefits, security is the primary obstacle that prevents the wide adoption of this promising computing model, especially for customers when their confidential data are consumed and produced during the computation. Treating the cloud as an intrinsically insecure computing platform from the viewpoint of the cloud customers, we must design mechanisms that not only protect sensitive information by enabling computations with encrypted data, but also protect customers from malicious behaviors by enabling the validation of the computation result. Focusing on engineering computing and optimization tasks, we investigates secure outsourcing of widely applicable linear programming (LP) computations. In order to achieve practical efficiency, our mechanism design explicitly decomposes the LP computation outsourcing into public LP solvers running on the cloud and private LP parameters owned by the customer. To validate the computation result, we explore the fundamental duality theorem of LP computation and derive the necessary and sufficient conditions that correct result must satisfy. In the proposed algorithm, the robustness preference of the output given by the server is checked by the client, whether the server is given the correct output. Whether the cloud is giving the correct result we also try to devise the robust algorithm for numerical stability.

Keywords: Public Cryptography, LP Parser, LP Definition Creator, RSA Algorithm, Robust Algorithm

1. Introduction
Public-key cryptography refers to a cryptographic system requiring two separate keys, one of which is secret and one of which is public. Although different, the two parts of the key pair are mathematically linked. One key locks or encrypts the plaintext and the other unlocks or decrypts the cipher text. Neither key can perform both functions (however, the private key can generate the public key). One of these keys is published or public, while the other is kept private. Public-key cryptography uses asymmetric key algorithms, and can also be referred to by the more generic term “asymmetric key cryptography.” Cloud Computing provides convenient on-demand network access to a shared pool of configurable computing resources that can be rapidly deployed with great efficiency and minimal management overhead [1]. Despite the tremendous benefits, outsourcing computation to the commercial public cloud is also depriving customers’ direct control over the systems that consume and produce their data during the computation, which inevitably brings in new security concerns and challenges towards this promising computing model [2].

- Public-key cryptography facilitates the following tasks:
  - Encryption and decryption allow two communicating parties to disguise information they send to each other. The sender encrypts, or scrambles, information before sending it. The receiver decrypts, or unscrambles, the information after receiving it. While in transit, the encrypted information is unintelligible to an intruder.
  - Tamper detection allows the recipient of information to verify that it has not been modified in transit. Any attempt to modify data or substitute a false message for a legitimate one will be detected.
  - Authentication allows the recipient of information to determine its origin—that is, to confirm the sender’s identity.
  - Nonrepudiation prevents the sender of information from claiming at a later date that the information was never sent. Specifically, we first formulate private data owned by the customer for LP problem a set of matrices and vectors. This higher level representation allows us to apply a set of efficient privacy-preserving problem transformation techniques, including matrix multiplication and affine mapping, to transform the original LP problem into some arbitrary one while protecting the sensitive input/output information. One crucial benefit of this higher level problem transformation method is that existing algorithms and tools for LP solvers can be directly reused by the cloud server. Although the generic mechanism defined at circuit level, e.g. [3], can even allow the customer to hide the fact that the outsourced computation is LP, we believe imposing this more stringent security measure than necessary would greatly affect the efficiency. To validate the computation result, we utilize the fact that the result is from cloud server solving the transformed LP problem. In particular, we explore the fundamental duality theorem together with the piece-wise construction of auxiliary LP problem to derive a set of necessary and sufficient conditions that the correct result must satisfy.
2. Problem Statement
We consider a computation outsourcing architecture involving two different entities, as illustrated in Fig. 1, the cloud customer, who has large amount of computationally Expensive LP problems to be outsourced to the cloud; the Cloud Server (CS), which has significant computation resources and provides utility computing services, such as hosting the public LP solvers in a pay-per-use manner. The customer has a large-scale linear programming problem Φ to be solved. However, due to the lack of computing resources, like processing power, memory, and storage etc., he cannot carry out such expensive computation locally. Thus, the customer resorts to CS for solving the LP computation and leverages its computation capacity in a pay-per-use manner.Instead of directly sending original problem Φ, the customer first uses a secret K to map Φ into some encrypted version Φk and outsources problem Φk to CS. CS then uses its public LP solver to get the answer of Φk and provides a correctness proof Γ, but it is supposed to learn nothing or little of the sensitive information contained in the original problem description Φ. After receiving the solution of encrypted problem Φk, the customer should be able to first verify the answer via the appended proof Γ. If it’s correct, he then uses the secret K to map the output into the desired answer for the original problem Φ. The security threats faced by the computation model primarily come from the malicious behavior of CS. We assume that the CS may behave beyond “honest-but-curious”, i.e. the semi-honest model that was assumed by many previous researches (e.g., [4], [5]), either because it intends to do so or because it is compromised. The CS may be persistently interested in analyzing the encrypted input sent by the customer and the encrypted output produced by the computation to learn the sensitive information as in the semi-honest model. In addition, CS can also behave unfaithfully or intentionally sabotage the computation, e.g. to lie about the result to save the computing resources, while hoping not to be caught at the same time. We also use the robustness for numerical stability.

3. Design Goals
To enable secure and practical outsourcing of LP under the aforementioned model, our mechanism design should achieve the following security and performance guarantees.
3.1. Correctness
Any cloud server that faithfully follows the mechanism must produce an output that can be decrypted and verified successfully by the customer.
3.2. Soundness
No cloud server can generate an incorrect output that can be decrypted and verified successfully by the customer with non-negligible probability.
3.3. Input/Output Privacy
No sensitive information from the customer’s private data can be derived by the cloud server during performing the LP computation.
3.4. Efficiency
The local computations done by customer should be substantially less than solving the original LP on his own. The computation burden on the cloud server should be within the comparable time complexity of existing practical algorithms solving LP problems.

4. Linear Programming
Linear programming is a mathematical method for determining a way to achieve the best outcome (such as maximum profit or lowest cost) in a given mathematical model for some list of requirements represented as linear relationships. Linear programming is a specific case of mathematical programming (mathematical optimization). More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Its feasible region is a convex polyhedron, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine function defined on this polyhedron. A linear programming algorithm finds a point in the polyhedron where this function has the smallest (or largest) value if such a point exists.
Linear programs are problems that can be expressed in canonical form:
where \( x \) represents the vector of variables (to be determined), \( c \) and \( b \) are vectors of (known) coefficients, \( A \) is a (known) matrix of coefficients, and \( \cdot ^T \) is the matrix transpose. The expression to be maximized or minimized is called the objective function (\( c^T x \) in this case). The inequalities \( A x \leq b \) are the constraints which specify a convex polytope over which the objective function is to be optimized. In this context, two vectors are comparable when they have the same dimensions. If every entry in the first is less-than or equal-to the corresponding entry in the second then we can say the first vector is less-than or equal-to the second vector.

5. The Proposed Schemes
5.1. Mechanism Design Framework
We propose to apply problem transformation for mechanism design. The general framework is adopted from a generic approach, while our instantiation is completely different and novel. In this framework, the process on cloud server can be represented by algorithm ProofGen and the process on customer can be organized into three algorithms (KeyGen, ProbEnc, ResultDec). These four algorithms are summarized below.

5.1.1. KeyGen(\( 1k \)) \rightarrow \{k\}
This is a randomized key generation algorithm which takes a system security parameter \( k \), and returns a secret key \( K \) that is used later by customer to encrypt the target LP problem.

5.1.2. ProbEnc(\( k, \Phi \)) \rightarrow \{\Phi k\}
This algorithm encrypts the input tuple \( \Phi \) into \( \Phi k \) with the secret key \( k \). According to problem transformation, the encrypted input \( \Phi k \) has the same form as \( \Phi \), and thus defines the problem to be solved in the cloud.

5.1.3. ProofGen(\( \Phi k \)) \rightarrow \{(y, \pi)\}
This algorithm augments a generic solver that solves the problem \( \Phi k \) to produce both the output \( y \) and a proof \( \pi \). The output \( y \) later decrypts to \( x \), and \( \pi \) is used later by the customer to verify the correctness of \( y \) or \( x \).

5.1.4. ResultDec(\( K, \Phi, y, \pi \)) \rightarrow \{x, \cdot \}
This algorithm may choose to verify either \( y \) or \( x \) via the proof \( \pi \). In any case, a correct output \( x \) is produced by decrypting \( y \) using the secret \( k \). The algorithm outputs \( \Phi k \) when the validation fails, indicating the cloud server was not performing the computation faithfully. Before presenting the details of our proposed mechanism, we study in this subsection a few basic techniques and show that the input encryption based on these techniques along may result in an unsatisfactory mechanism. However, the analysis will give insights on how a stronger mechanism should be designed. Note that to simplify the presentation, we assume that the cloud server honestly performs the computation, and defer the discussion on soundness.

5.2. Basic Techniques
5.2.1. Hiding Equality Constraints (A, b)
First of all, a randomly generated \( m \times m \) non singular matrix \( Q \) can be part of the secret key \( k \). The customer can apply the matrix to original constraint for the following constraints transformation,

\[
\begin{align*}
Ax &= b \\
A'x &= b'
\end{align*}
\]

where \( A' = QA \) and \( b' = Qb \).
Since we have assumed that \( A \) has full row rank, \( A' \) must have full row rank. Without knowing \( Q \), it is not possible for one to determine the exact elements of \( A \). However, the null spaces of \( A \) and \( A' \) remains the same, which may violate the security requirement of some applications. The vector \( b \) is encrypted in a perfect way since it can be mapped to an arbitrary \( b' \) with a proper choice of \( Q \).

5.2.2 Hiding Inequality Constraints (B)
The customer cannot transform the inequality constraints in the similar way as used for the equality constraints. This is because for an arbitrary invertible matrix \( Q \), \( Bx \geq 0 \) is not equivalent to \( QBx \geq 0 \) in general. To hide \( B \), we can leverage the fact that a feasible solution to our constraint must satisfy the equality constraints. To be more specific, the feasible regions defined by the following two groups of constraints are the same.

\[
\begin{align*}
Ax &= b \\
Bx &\geq 0 \\
(A - \lambda A)x &= B'x \geq 0
\end{align*}
\]

where \( \lambda \) is a randomly generated \( n \times m \) matrix in \( K \) satisfying that \( |B'| = |B - \lambda A| \neq 0 \) and \( \lambda b = 0 \). Since the condition \( \lambda b = 0 \) is largely underdetermined, it leaves great flexibility to choose \( \lambda \) in order to satisfy the above conditions.

5.2.3 Hiding Objective Functions \( c \) and Value \( c^T x \)
Given the widely application of LP, such as the estimation of business annul revenues or personal portfolio holdings etc., the information contained in objective function \( c \) and optimal objective value \( c^T x \) might be as sensitive as the constraints
of A, B, b. Thus, they should be protected, too. To achieve this, we apply constant scaling to the objective function, i.e. a real positive scalar is generated randomly as part of encryption key K and c is replaced by c. It is not possible to derive the original optimal objective value $c^T x$ without knowing first, since it can be mapped to any value with the same sign. While hiding the objective value well, this approach does leak structure-wise information of objective function c. Namely, the number and position of zero-elements in c are not protected. Besides, the ratio between the elements in c are also preserved after constant scaling.

6. Proposed Algorithm

Robust Algorithm Steps
Step 1: Extract the output from the cloud server say(a)
Step 2: Compute an output at client end say(b)
Step 3: if (a==b) then
  3.1: Cloud server will perform honestly
  3.2: Cloud will given an incorrect result
Step 4: If both the values will be match that is produce by client and server then our algorithm will give a robustness of output.

7. Experimental Results

7.1 Linear Results:

<table>
<thead>
<tr>
<th>Equation</th>
<th>Benchmark</th>
<th>Original Problem</th>
<th>Encrypted Problem</th>
<th>Cloud Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>N</td>
<td>M</td>
<td>N</td>
<td>M</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>0.67</td>
<td>0.7</td>
</tr>
<tr>
<td>2</td>
<td>109</td>
<td>200</td>
<td>0.127</td>
<td>0.134</td>
</tr>
<tr>
<td>3</td>
<td>45</td>
<td>198</td>
<td>0.2</td>
<td>0.27</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>200</td>
<td>0.321</td>
<td>0.324</td>
</tr>
<tr>
<td>5</td>
<td>233</td>
<td>700</td>
<td>0.544</td>
<td>0.566</td>
</tr>
<tr>
<td>6</td>
<td>1500</td>
<td>3</td>
<td>0.666</td>
<td>0.7</td>
</tr>
<tr>
<td>7</td>
<td>2344</td>
<td>2000</td>
<td>0.899</td>
<td>0.92</td>
</tr>
</tbody>
</table>

7.2 NonLinear Results:

<table>
<thead>
<tr>
<th>Equation</th>
<th>Benchmark</th>
<th>Original Problem</th>
<th>Encrypted Problem</th>
<th>Cloud Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>N</td>
<td>M</td>
<td>N</td>
<td>M</td>
</tr>
<tr>
<td>1</td>
<td>1.66</td>
<td>18.59</td>
<td>0.245</td>
<td>0.27</td>
</tr>
<tr>
<td>2</td>
<td>39.45</td>
<td>89.99</td>
<td>0.297</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>39.54</td>
<td>89.99</td>
<td>0.432</td>
<td>0.478</td>
</tr>
<tr>
<td>4</td>
<td>90.00037</td>
<td>90.00082</td>
<td>0.544</td>
<td>0.567</td>
</tr>
<tr>
<td>5</td>
<td>-116.11</td>
<td>106.43</td>
<td>0.788</td>
<td>0.798</td>
</tr>
<tr>
<td>6</td>
<td>1887.7</td>
<td>899</td>
<td>0.9</td>
<td>0.912</td>
</tr>
<tr>
<td>7</td>
<td>2787.7</td>
<td>89.99</td>
<td>1.2</td>
<td>1.233</td>
</tr>
</tbody>
</table>
8. Conclusion
In this paper, for the first time, we formalize the problem of securely outsourcing LP computations in cloud computing, and provide such a practical mechanism design which fulfills input/output privacy, cheating resilience, and efficiency. By explicitly decomposing LP computation outsourcing into public LP solvers and private data, our mechanism design is able to explore appropriate security/efficiency tradeoffs via higher level LP computation than the general circuit representation. We develop problem transformation techniques that enable customers to secretly transform the original LP into some arbitrary one while protecting sensitive input/output information. We also investigate duality theorem and derive a set of necessary and sufficient condition for result verification. Such a cheating resilience design can be bundled in the overall mechanism with close-to-zero additional overhead. Both security analysis and experiment results demonstrate the immediate practicality of the proposed mechanism. We also devise the robustness for the numerical stability and also a non-linear programming can be design.

References
Author

Sheetal Phatangare received his B.E. degree in Computer from Modern college of Engineering, India, in 2005, and now appearing the M.E. degree in Computer from Pune University, India. She was a lecturer with the Department of Computer, Pune University from 2009.

Dr. Pradeep K Deshmukh PhD in Computer Science and Engineering. He received M.E. & B.E. in Computer Science & Engineering. His key research interest include Cloud Computing, Network Security, ANN. He is currently working as Professor in Rajarshi Shahu college of Engg. Pune India (M.S.) Department of Post Graduate Computer engineering with the total experience of about 21 years.

Prof. Rushali A. Deshmukh completed M.E. & B.E. in Computer Science & Engineering. She is currently working as Professor in Rajarshi Shahu college of Engg. Pune India, Department of Post Graduate Computer engineering with the total experience of about 12 years.