# Efficient Privacy-preserving Outsourcing of Large-scale QR Factorization

Changqing Luo
*Dept. of Electrical Engineering and Computer Science*
*Case Western Reserve University*
*Cleveland, OH 44106*
*Email: cxl881@case.edu*

Kaijin Zhang
*Dept. of Electrical Engineering and Computer Science*
*Case Western Reserve University*
*Cleveland, OH 44106*
*Email: kxz138@case.edu*

Sergio Salinas
*Dept. of Electrical Engineering and Computer Science*
*Wichita State University*
*Wichita, KS 67260*
*Email: salinas@cs.wichita.edu*

Pan Li
*Dept. of Electrical Engineering and Computer Science*
*Case Western Reserve University*
*Cleveland, OH 44106*
*Email: lipan@case.edu*

***Abstract*—Modern organizations have collected vast amounts of data created by various systems and applications. Scientists and engineers have a strong desire to advance scientific and engineering knowledge from such massive data. QR factorization is one of the most fundamental mathematical tools for data analysis. However, conducting QR factorization of a matrix requires high computational complexity. This incurs a formidable challenge in efficiently analyzing large-scale data sets by normal users or small companies on traditional resource limited computers. To overcome this limitation, industry and academia propose to employ cloud computing that can offer abundant computing resources. This, however, raises privacy concerns because users' data may contain sensitive information that needs to be hidden for ethical, legal, or security reasons. To this end, we propose a privacy-preserving outsourcing algorithm for efficiently performing large-scale QR factorization. We implement the proposed algorithm on the Amazon Elastic Compute Cloud (EC2) platform and a laptop. The experiment results show significant time saving for the user.**

## 1. Introduction

In recent years, we have been creating enormous amounts of data in our daily life. International Data Corporation (IDC) has reported that the world created about 1.8 ZB ($= 10^{21}$ Bytes) of data in 2011 [1], and predicts that this figure will be at least doubled every other two years in the near future. These data are produced by various systems and applications, including power systems [2], intelligent transportation systems [3], and e-commerce companies [4].

By analyzing such large-scale data, scientists and engineers are able to better advance scientific and engineering knowledge. For example, power engineers can find the optimal power flow by performing real-time analysis of the huge amount of smart metering data in electric grids [5]; social scientists can study social interactions by analyzing large volumes of records from smart transportation systems [6]; and e-commerce companies are trying to provide customers with better recommendations by analyzing billions of product reviews [4].

We observe that QR factorization [7] is one of the most fundamental mathematical tools for big data analysis. For instance, in power systems, power engineers applied QR factorization to estimate power system state in the presence of bad data [8]. In e-commerce companies, marketing specialists employed QR factorization to analyze large data for finding business trends and predicting forthcoming social agendas [9]. More importantly, QR factorization is a very popular tool for conducting singular value decomposition (SVD) [10] and solving least squares problems [11] that widely exist in real-world data analysis problems. Therefore, we focus on large-scale QR factorization in this paper.

In the literature, researchers have developed several methods to conduct QR factorization like Householder Reflection [12]. It has been shown that conducting QR factorization on an $m \times n$ matrix by employing this method requires $\mathcal{O}(m^2 n)$ computational complexity. Consequently, users face a formidable challenge in computing large-scale QR factorization on general-purpose computers.

So far, industry and academia have proposed to employ cloud computing for large-scale computations. This is because the cloud is capable of providing an efficient and economical way to analyze such massive data. Specifically, a user with a general-purpose computer outsources his/her computational tasks to the resource-rich cloud, and thus enabling to solve the computation problems very quickly. More importantly, the cloud offers on-demand services, and requires no specific hardware and software from users. However, adopting cloud computing raises privacy concerns. To be more specific, the data implicitly contain users' private information. For example, we can find out customers' energy

*This work was partially supported by the U.S. National Science Foundation under grants CNS-1602172 and CNS-1566479.*

2324-9013/17 $31.00 © 2017 IEEE
DOI 10.1109/Trustcom/BigDataSE/ICESS.2017.331

consumption patterns by analyzing energy customers' meter readings [13] and learn people's daily live behaviors from people's intelligent transportation records [14]. Therefore, to enable large-scale QR factorization, we need to design a privacy-preserving outsourcing algorithm that is able to protect users' data privacy.

Some previous works have studied this privacy issue raised by using the cloud, and design privacy-preserving outsourcing algorithms. Some of them propose to protect users' data privacy by utilizing traditional cryptographic techniques like homomorphic encryption. For example, Gennaro *et al.* [15] develop a privacy-preserving outsourcing algorithm by employing fully homomorphic encryption (FHE). Wang *et al.* [16] develop a partial homomorphic encryption based privacy-preserving outsourcing scheme for solving a linear system of equations (LSEs). However, these schemes require clients to perform expensive data pre-processing. Besides, due to the expensive encryption/decryption operations, FHE is inefficient for practical big data applications. On the other hand, some other works consider employing linear algebra operations to transform the data for protecting data privacy. For instance, Salinas *et al.* devise privacy-preserving outsourcing schemes for solving LSEs [17] and quadratic programs (QP) [18], where the original problems are transformed by efficient arithmetic and linear algebra operations. However, the privacy-preserving outsourcing algorithm for large-scale QR factorization has not well been studied so far.

In this paper, we design an efficient and practical privacy-preserving outsourcing algorithm for performing large-scale QR factorization in the cloud environment. Specifically, a user first conceals the original data by employing the privacy-preserving orthogonal matrix transformation and the privacy-preserving upper triangular matrix transformation. Then, the cloud performs QR factorization on the transformed matrix in a non-interactive way, and outputs two factor matrices. At last, the user recovers the correct results from the returned factor matrices through multiplying them by the inverse of the random masking matrices, respectively. In our proposed algorithm, the user only performs operations with random sparse matrices when decomposing an $m \times n$ matrix, resulting in the computational complexity of $\mathcal{O}(\max\{m^2, mn\})$. Moreover, since the cloud conducts QR factorization on the transformed matrices without exchanging data with the user, the communication overhead is very small.

We summarize our key contributions as follows.

1) We develop an efficient privacy-preserving outsourcing algorithm for large-scale QR factorization.
2) We develop three privacy-preserving matrix transformations to transform an orthogonal matrix, an upper triangular matrix, and a general matrix, respectively.
3) The proposed privacy-preserving outsourcing QR factorization algorithm requires computational complexity of $\mathcal{O}(\max\{m^2, mn\})$ at the user side, and very low communication overhead between the user and the cloud.
4) We implement the proposed algorithm on the Amazon EC2 platform and a laptop. We find that our proposed algorithm can significantly save the user's computing time.

The rest of this paper is organized as follows. In Section 2, we introduce the QR factorization and the system architecture. Section 3 describes the proposed privacy-preserving matrix transformations. Section 4 details the proposed algorithm for privacy-preserving outsourcing large-scale QR factorization. Section 5 provides a thorough performance analysis. We present the experimental results in Section 6, and finally conclude the paper in Section 7.

## 2. Problem Formulation

In this section, we introduce the QR factorization and the considered system architecture.

### 2.1. Householder based QR Factorization

QR factorization is to decompose a matrix with linearly independent columns into a product of an orthogonal matrix and an upper triangular matrix, i.e.,

$$A = Q \times R, \tag{1}$$

where $A$ is an $m \times n$ ($m \geq n$) full column rank matrix, $Q$ is an $m \times m$ orthogonal matrix, and $R$ is an $m \times n$ upper triangular matrix. All the computations are conducted within the group $\mathbb{G}$, which is a sufficiently large multiplicative subgroup of $\mathbb{Z}_p^*$ where $\mathbb{Z}_p^* = \{z | 1 \leq z \leq p-1\}$ and $p$ is a large prime. Besides, we consider that all the variables are integer.

To find $Q$ and $R$, we perform orthogonal transformations on $A$. It is noteworthy that performing a QR factorization by Householder Reflection and Givens Rotation requires the same computational complexity. Here, we employ Householder Reflection whose process is as follows.

We denote matrix $A$ as $A = [u_1, u_2, \cdots, u_n]$ where $u_i$ $(1 \leq i \leq n)$ is the $i$th column vector. Then, based on $u_1$, we derive its Householder transformation $P_1$ as

$$P_1 = I - \frac{2u_1 u_1^T}{||u_1||_2^2}, \tag{2}$$

where $P_1$ is a $m \times m$ symmetric orthogonal matrix and $I$ is an identity matrix. Subsequently, multiplying $P_1$ by $A$, we can have a new matrix, denoted by $A1$, as follows:

$$A1 = P_1 \times A. \tag{3}$$

Following the same process, we set $u_2 = (0, a1_{2,2}, a1_{3,2}, \cdots, a1_{m,2})^T$, where $a1_{i,j}$ $(i, j \in [1, m])$ is the element in the $i$th row and $j$th column of $A1$. Subsequently, we obtain the Householder transformation $P_2$ of $u_2$, and then multiply $P_2$ by $A1$ to get a new matrix, denoted by $A2$, as follows:

$$A2 = P_2 \times A1 = P_2 P_1 A.$$

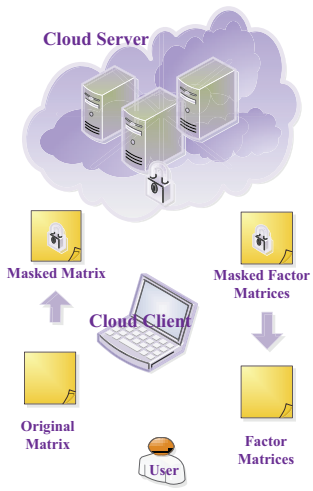



Figure 1. A privacy-preserving architecture for outsourcing large-scale QR factorization.

We continue this process until it reaches the Householder transformation $P_{m-1}$, which finally leads to $R$ as follows:

$$R = P_{m-1} \cdots P_2 P_1 A. \tag{4}$$

In the meantime, $Q$ can be obtained by

$$Q = P_1^{-1} P_2^{-1} \cdots P_{m-1}^{-1}. \tag{5}$$

### 2.2. System Architecture

We consider a two-party computing architecutre for conducting large-scale QR factorization, as shown in Fig. 1. One part is a resource-limited computer client who needs to perform the QR factorization, and the other is the resource-abundant cloud server which is able to offer on-demand services to help complete large-scale QR factorization.

## 3. Privacy-preserving Matrix Transformations

In this section, we describe three privacy-preserving matrix transformations used for protecting an upper triangular matrix, an orthogonal matrix, and a general matrix, respectively.

### 3.1. The Privacy-preserving Upper Triangular Matrix Transformation

To protect the privacy of an upper triangular matrix, we propose to multiply the matrix by a pseudorandom upper triangular matrix. Specifically, we conceal matrix $R = (r_{i,j})_{1 \le i \le m, 1 \le j \le n}$ as follows:

$$\hat{R} = R \times S, \tag{6}$$

where $S$ represents an $n \times n$ upper triangular matrix, and its elements are determined by

$$s_{i,j} = \begin{cases} d_i, & \text{for } 1 \le i = j \le n, \\ d_{i,j}, & \text{for } 1 \le i < j \le n \\ 0, & \text{otherwise} \end{cases}. \tag{7}$$

Each diagonal elements $d_i \in \mathbb{G}$ $(1 \le i \le n)$ is determined by a pseudorandom function $F_o : \{0,1\}^\omega \times \{0,1\}^\omega \to \{0,1\}^\omega$, i.e.,

$$d_i = F_o(r_i, g), \tag{8}$$

where $r_i$ is a random string and $g$ is a constant one. The off-diagonal values $d_{i,j} \leftarrow \{0,1\}^k \in \mathbb{G}$ $(1 \le i < j \le n)$ are set to arbitrary positive constants.

Thus, each non-zero element of matrix $\hat{R}$, denoted by $\hat{r}_{i,j}$ $(1 \le i \le j \le n)$, can be calculated as

$$\hat{r}_{i,j} = R(i,\cdot) \cdot S(\cdot, j) = r_{i,i} d_{i,j} + r_{i,i+1} d_{i+1,j} + \cdots + r_{i,j} d_j. \tag{9}$$

We denote the computation as

$$\hat{r}_{i,j} = Mask^{F_o}(r_i, d_{i,j}, d_{i+1,j}, \cdots, d_{j-1,j}, r_{i,i}, r_{i,i+1}, \cdots, r_{i,j}). \tag{10}$$

### 3.2. The Privacy-preserving Orthogonal Matrix Transformation

To protect the privacy of an orthogonal matrix, we propose to multiply it by a random orthogonal matrix. Specifically, we hide an $m \times m$ orthogonal matrix $Q = (q_{i,j})_{1 \le i,j \le m}$ by

$$\hat{Q} = O \times Q, \tag{11}$$

where $O = (o_{i,j})_{1 \le i,j \le m}$ denotes an $m \times m$ orthogonal matrix.

To build the matrix $O$, we first generate an $m \times 1$ random vector $u$ with its elements $u_i \in \mathbb{G}$ (for $i \in [1, m]$) is determined by a pseudorandom function $F_o : \{0,1\}^\omega \times \{0,1\}^\omega$, i.e.,

$$u_i = F_o(\bar{r}_i, g), \forall i \in [1, m], \tag{12}$$

where $\bar{r}_i$ is a random string and g is a constant one. Then, we find $O$ by employing the Householder transformation of the vector $u$ based on Eq. (2), i.e., . Thus, $O$ is given by

$$O = I - \frac{2uu^T}{||u||_2^2}. \tag{13}$$

Therefore, the elements of matrix $\hat{Q}$ are calculated as:

$$\hat{q}_{i,j} = O_{i,\cdot} \times Q_{\cdot,j}, \tag{14}$$

for $1 \le i \le m, 1 \le j \le m$. We denote such computations as

$$\hat{q}_{i,j} = Mask^{F_o}(\bar{r}_i, q_{1,j}, q_{2,j}, \cdots, q_{m,j}). \tag{15}$$

### 3.3. The Privacy-preserving Low-complexity Matrix Transformation

To efficiently conceal the private information of a general matrix, we propose to employ the following low-complexity matrix multiplication. Specifically, we preserve the privacy of an $m \times n$ general matrix $G = (g_{i,j})_{1 \le i \le m, 1 \le j \le n}$ by

$$\hat{G} = D_G G F_G, \tag{16}$$

where $F_G$ is an $n \times n$ diagonal matrix with arbitrary positive constants that are within the group $\mathbb{G}$ and $D_G$ is an $m \times m$ random diagonal matrix. Note that we will use $D_* * F_*$ and $F_* * D_*$ to represent this efficient general matrix transformation in the rest of this paper, where $*$ indicates a general matrix to be transformed.

Matrix $D_G$ is determined by

$$dg_{i,j} = \begin{cases} t_i, & \text{if } i = j \text{ for } i, j \in [1, m] \\ 0, & \text{otherwise} \end{cases}, \tag{17}$$

where diagonal elements are determined by a pseudorandom function $F_o : \{0,1\}^\omega \times \{0,1\}^\omega \to \{0,1\}^\omega$, i.e.,

$$t_i = F_o(r_i'', g), \forall i \in [1, m], \tag{18}$$

where $r_i''$ is a random string and g is a constant one.

Matrix $F_G$ is defined by

$$fg_{i,j} = \begin{cases} h_{i,j}, & \text{if } i = j \text{ for } i, j \in [1, n] \\ 0, & \text{otherwise,} \end{cases}. \tag{19}$$

where $h_{i,j} \leftarrow \{0,1\}^k$ is an arbitrary positive constant in $\mathbb{G}$.

As a result, based on Eq. (16), the elements in matrix $\hat{G}$ are given by

$$\hat{g}_{i,j} = t_i g_{i,j} h_j, \tag{20}$$

for $1 \le i \le m, 1 \le j \le n$. We denote such computations as

$$\hat{g}_{i,j} = Mask^{F_o}(r_i'', h_j, g_{i,j}). \tag{21}$$

## 4. Privacy-preserving Outsourcing QR Factorization

### 4.1. The Main Idea

Our privacy-preserving outsourcing algorithm for large-scale QR factorization consist of the following steps: transforming an original matrix at the user, decomposing the transformed matrix in the cloud, and recovering the correct results at the user. In the following, we explain each of the steps in detail.

First, to privately outsource QR factorization, a user employs simple linear algebra operations to transform the original matrix. Specifically, the user transforms matrix $A$ by multiplying it with a random orthogonal matrix and a random upper triangular matrix, i.e.,

$$\hat{A} = OAS. \tag{22}$$

Here, each element $\hat{a}_{i,j}$ of $\hat{A}$ can be computed as $\hat{a}_{i,j} = \sum_{j'=1}^{m} \hat{q}_{i,j'} \cdot \hat{r}_{j',j}$.

Second, the cloud conducts QR factorization on the transformed matrix $\hat{A}$ in a non-interactive way, which results in limited communication overhead. The cloud then directly sends the user the two factor matrices, i.e., $\hat{Q}$ and $\hat{R}$.

Finally, the user conducts the matrix recovery to find the correct decomposition results from the returned factor matrices. Intuitively, the recovery can be done by multiplying a factor matrix with the inverse of the corresponding mask matrix. For example, to recover matrix $R$ from matrix $\hat{R}$, we need to perform the following multiplication, i.e., $R = \hat{R}S^{-1}$. Since both matrix inverse and matrix multiplication involve high-complexity computations, the user needs to outsource both operations to the cloud.

### 4.2. The Privacy-preserving Outsourcing of Large-scale QR Factorization

We present the designed privacy-preserving outsourcing QR factorization algorithm step by step.

**Step 1: Privacy-preserving Transformation of the Original Matrix $A$:** At this step, the user needs to transform matrix $A$ into $\hat{A} = OAS$. Due to the high complexity, the user outsources the multiplications to the cloud.

The user first generates matrix $O$ from a random vector $u$ according to Eq. (13). Then, the user utilizes the privacy-preserving low-complexity matrix transformation to transform matrices $O$, $A$, and $S$ as follows:

$$O' = D_O O F_O, A' = F_A A D_A, \text{ and } S' = F_S S D_S, \tag{23}$$

where$F_A = F_O^{-1}$, and transmits $O'$, $A'$, and $S'$ to the cloud.

The cloud first performs matrix multiplication to obtain

$$\bar{OA} = O'A' = D_O O A D_A. \tag{24}$$

Subsequently, the cloud returns matrix $\bar{OA}$ to the user which performs the following computation:

$$\bar{OA}' = \bar{OA} D_A^{-1} F_{OA} = D_O O A F_{OA}, \tag{25}$$

where $F_{OA} = F_S^{-1}$.

Then, the user outsources $\bar{OA}'$ to the cloud which completes the matrix transformation as follows:

$$\hat{A}' = \hat{OA}' S' = D_O O A S D_S, \tag{26}$$

and returns $\hat{A}'$ to the user.

Finally, the user recovers matrix $\hat{A}$ from $\hat{A}'$ as follows:

$$\hat{A} = D_O^{-1} \hat{A}' D_S^{-1} = OAS. \tag{27}$$

**Step 2: Privacy-preserving Decomposition of the Matrix $\hat{A}$:** At this stage, the user first outsources matrix $\hat{A}$ to the cloud. Then, the cloud conducts QR factorization on $\hat{A}$, computes an orthogonal matrix $\hat{Q}$ and an upper triangular matrix $\hat{R}$, i.e.,

$$\hat{A} = \hat{Q}\hat{R}, \tag{28}$$

where $\hat{Q} = OQ$ and $\hat{R} = RS$, and returns $\hat{Q}$ and $\hat{R}$ to the user.

**Step 3: Privacy-preserving Recovery of the Orthogonal Matrix $Q$:** To recover matrix $Q$ from $\hat{Q}$, the user has to perform the multiplication, i.e., $Q = O^{-1}\hat{Q} = O^T\hat{Q}$, which needs to be outsourced to the cloud because of the high computational complexity.

Similarly, the user first employs the privacy-preserving low-complexity matrix transformation to transform matrix $\hat{Q}$, i.e.,

$$\hat{Q}' = F_{\hat{Q}}\hat{Q}D_{\hat{Q}}. \tag{29}$$

On the other hand, the user first makes matrix transpose to obtain $O^T = O^{-1}$, and then performs the multiplicaiton to get $O'' = D_{O^{-1}}O^{-1}F_{O^{-1}}$, where $F_{\hat{Q}}^{-1} = F_O^T$. As last, the user sends $\hat{Q}'$ and $O''$ to the cloud.

Then, the cloud performs the following matrix multiplication to get

$$Q' = O''\hat{Q}' = D_{O^{-1}}QD_{\hat{Q}}. \tag{30}$$

Finally, the cloud returns matrix $Q'$ to the user who can have the correct result as follows:

$$Q = D_{O^{-1}}^{-1}Q'D_{\hat{Q}}^{-1}. \tag{31}$$

**Step 4: Privacy-preserving Recovery of the Upper Triangular Matrix $R$:** The user can recover matrix $R$ by multiplying $\hat{R}$ by $S^{-1}$, i.e., $R = \hat{R}S^{-1}$. To find matrix $S^{-1}$, the user first transforms $S$ into $S'' = D_S S F_S$, and then outsources it to the cloud which call a matrix inverse algorithm to get $S''^{-1} = F_{S'}^{-1}S^{-1}D_{S'}^{-1}$.

The user employs the privacy-preserving low-complexity matrix transformation to transform matrix $\hat{R}$, i.e.,

$$\hat{R}' = D_{\hat{R}}\hat{R}F_{\hat{R}}, \tag{32}$$

where $F_{\hat{R}} = F_{S'}$, and sends $\hat{R}'$ to the cloud.

Subsequently, the cloud performs the following multiplication, and returns the result to the user:

$$R' = \hat{R}'S''^{-1} = D_{\hat{R}}RSF_{\hat{R}} \cdot F_{S'}^{-1}S^{-1}D_S^{-1} = D_{\hat{R}}RD_S^{-1}. \tag{33}$$

At last, the user can obtain the correct result as follows:

$$R = D_{\hat{R}}^{-1}R'D_S. \tag{34}$$

We summarize the proposed privacy-preserving outsourcing algorithm for QR factorization on matrix $A$ in **Algorithm** 1.

**Algorithm 1** : The privacy-preserving outsourcing algorithm for performing QR factorization on a matrix $A$

Initialization: The user generates a vector $u$, matrices $S$, $D_O$, $D_A$, $D_S$, $F_O$, $F_A$, $F_{OA}$, $F_S$, $D_{\hat{Q}}$, $F_{\hat{Q}}$, $D_{\hat{R}}$, and $F_{\hat{R}}$.
**Step 1: Transforming matrix $A$**
  The user employs Householder transformation to generate matrix $O$ from a vector $u$.
  The user transforms matrices $O$, $A$, and $S$ into matrices $O'$, $A'$, and $S'$, respectively, based on Eq. (23).
  The user outsources $O'$, $A'$, and $S'$ to the cloud.
  The cloud computes $\hat{OA}$ by using Eq. (24).
  The cloud returns $\hat{OA}$ to the The user.
  The user computes $\hat{OA}'$ by using Eq. (25).
  The user sends $\hat{OA}'$ to the cloud.
  The cloud computes $\hat{A}'$ by using Eq. (26).
  The cloud returns $\hat{A}'$ to the The user.
  The user recovers $\hat{A}$ from $\hat{A}'$ by using Eq. (27).
**Step 2: QR factorization of matrix $\hat{A}$**
  The user outsources $\hat{A}$ to the clodu.
  The cloud decomposes $\hat{A}$ into a product of $\hat{Q}$ and $\hat{R}$ by performing Householder based QR factorization.
  The cloud returns $\hat{Q}$ and $\hat{R}$ to the user.
**Step 3: Recovering matrix $Q$ from matrix $\hat{Q}$**
  The user computes $\hat{Q}'$ based on Eq. (29).
  The user transposes $O$ to obtain $O^{-1}$.
  The user computes $O''$.
  The user sends $\hat{Q}'$ and $O''$ to the cloud.
  The cloud computes $Q'$ through Eq. (30).
  The cloud returns $Q'$ to the user.
  The user obtains $Q$ by using Eq. (31).
**Step 4: Recovering matrix $R$ from matrix $\hat{R}$**
  The user transforms $S$ into $S''$, and sends it to the cloud.
  The cloud calls a matrix inverse algorithm to obtain $S''^{-1}$.
  The user transforms $\hat{R}$ into $\hat{R}'$ based on Eq. (32).
  The user outsources $\hat{R}'$ to the cloud.
  The cloud computes $R'$ through Eq. (33).
  The cloud returns $R'$ to the user.
  The user finds $R$ by using Eq. (34).
End;

## 5. Performance Analysis

In this section, we present the analysis of the computational complexity, communication overhead, and privacy, respectively.

### 5.1. Computational Complexity

We present the computational complexity analysis of the proposed privacy-preserving outsourcing algorithm for QR factorization. Particularly, we thoroughly analyze the computational complexity at each stage.

*At Step1*, the user first generates a random orthogonal matrix $O$, which takes $\mathcal{O}(m^2)$ computational complexity. Then, the user employs the privacy-preserving low-complexity general matrix transformation to respectively transform $O$, $A$, and $S$ into $O'$, $A'$, and $S'$, which requires computational complexity of $\mathcal{O}(2m^2 + 4mn)$. Then, the cloud performs the matrix multiplications, i.e., $\hat{OA} = O'A'$,

resulting in the computational complexity of $\mathcal{O}(m^2n)$. Afterwards, the user obtains $\bar{OA}'$ from $\bar{OA}$, which takes $\mathcal{O}(m^2)$ computational complexity. Subsequently, the cloud performs the computation, i.e., $\hat{A}' = \bar{OA}'S'$, resulting in $\mathcal{O}(m^2n)$ computational complexity. Finally, the user recovers $\hat{A}$ from $\hat{A}'$, which requires the computations with $\mathcal{O}(2mn)$ complexity.

*At Step 2*, the cloud performs QR factorization on $\hat{A}$, which generally requires to perform computations with $\mathcal{O}(2m^2n + m^3)$ complexity.

*At Step 3*, the user first transposes $O$ to get $O^{-1}$, which requires $\mathcal{O}(m)$ computational complexity. Afterwards, he/she employs the privacy-preserving low-complexity general matrix transformation to respectively transform $\hat{Q}$ and $O^{-1}$ into $\hat{Q}'$ and $O''$, which takes $\mathcal{O}(4m^2)$ computational complexity. Then, the cloud performs the matrix multiplication, i.e., $Q' = O''\hat{Q}'$, to get $Q'$, which requires computations with $\mathcal{O}(m^2n)$ complexity. Finally, the user recovers $Q$ from $Q'$, which costs computations with $\mathcal{O}(2m^2)$ complexity.

*At Step 4*, the user first applies the privacy-preserving low-complexity general matrix transformation to respectively transform $S$ and $\hat{R}$ into $S''$ and $\hat{R}'$, which takes $\mathcal{O}(2m^2 + 2mn)$ computational complexity. On the other hand, the cloud first find the inverse of $S''$, which cost the computations with $\mathcal{O}(Nn^3)$. Afterwards, the cloud performs the computation, i.e., $\hat{R}'S''^{-1}$, to obtain $R'$, which requies $\mathcal{O}(m^2n)$ computational complexity. At last, the user recovers $R$ from $R'$, resulting in the computational complexity of $\mathcal{O}(2mn)$.

To sum up, the proposed algorithm requires computational complexity of $\mathcal{O}(\max\{mn, m^2\})$ at the user and that of $\mathcal{O}(\max\{m^2n, m^3\})$ in the cloud.

### 5.2. Communication Overhead

We detail the analysis of the communication overhead at each stage of the proposed privacy-preserving outsourcing algorithm for QR factorization as follows.

*At Step 1*, the user outsources $O'$, $A'$, and $S'$ to the cloud, and then the cloud returns $\hat{A}'$ to the CC, which results in communication overhead of $\mathcal{C}(m^2 + 3mn)$. *At Step 2*, the cloud returns $\hat{Q}$ and $\hat{R}$ to the user, which results in communication overhead of $\mathcal{C}(m^2 + mn)$. *At Steps 3 and 4*, the user sends $\hat{Q}'$ and $\hat{R}'$ to the cloud. Afterwards, the cloud returns $Q'$ and $R'$ to the user, resulting in $\mathcal{C}(2m^2 + 2mn + m + n)$ communication overhead.

Therefore, the total communication overhead for performing QR factorization on matrix $A$ is $\mathcal{C}(4m^2 + 6mn + m + n)$.

### 5.3. Privacy Analysis

Inspecting the proposed privacy-preserving outsourcing algorithm for QR factorization, we observe that the cloud only has access to the transformed matrices, and thus is unable to learn private information from them. In the following, we offer a detailed analysis.

*At Step 1*, the cloud can access to $O'$, $A'$, $S'$, $\bar{OA}$, $\bar{OA}'$, and $\hat{A}'$. *At Step 2*, the cloud can access to $\hat{A}$, $\hat{Q}$, and $\hat{R}$. *At Step 3*, the cloud accesses to $\hat{Q}'$, $O''$, and $O'$. *At Step 4*, the cloud accesses to $S''$, $S''^{-1}$, $\hat{R}'$, and $R'$. We can see that the cloud only accesses to the transformed matrices, and thus the privacy is protected.

## 6. Experiment Results

To well evaluate the performance, we implement our proposed algorithm in a real-world scenario. In our experiment, the user is a laptop with a dual-core 2.7GHz CPU, 8GB RAM memory, and 256GB solid state drive. The cloud is the Amazon Elastic Compute Cloud (EC2) platform with 4 computing nodes. We implement the proposed algorithm by Java. Moreover, we apply AKKA computing framework [19] to manage the storage and computations at the EC2 platform. Additionally, we test the performance of our proposed algorithm with the synthetic data.

We first evaluate the running time of the proposed privacy-preserving outsourcing algorithms for QR factorization. Fig. 2 shows the time spent for different total number of elements in the tested matrices.

Specifically, Fig. 2(a) demonstrates the computing time at the user for performing QR factorization. We observe that the computer at the user completes the required computations very quick, even for large-scale matrices. For example, when the total number of elements in the matrix is $4 \times 10^6$, the user spends $1831s$ for performing QR factoriztion. Besides, we also find that the running time at the user is approximately linear regarding to the number of elements, which is practical for decomposing large-scale matrices.

On the other hand, Fig. 2(b) illustrates the running time taken by the cloud to perform matrix transformation, conduct QR factorization, and recover the correct decomposition result. We find that the computing time of the cloud is increasing with the growth of the matrix size.

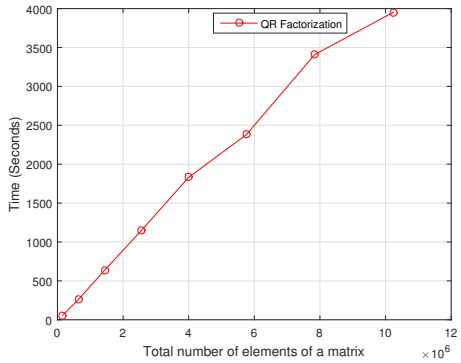
Subsequently, we examine the computing time saving provided by the proposed algorithms. Specifically, in Tables 1, we compare the computing time of a user conducting QR factorization by itself with that achieved by our proposed algorithm. We observe that the proposed algorithm has significant time saving compared to the one only locally performing QR factorization. For example, the user spends $1297549s$ to perform QR factorization on the matrix with $1.024 \times 10^7$ by itself. This validates the efficacy of our proposed algorithm.

## 7. Conclusions

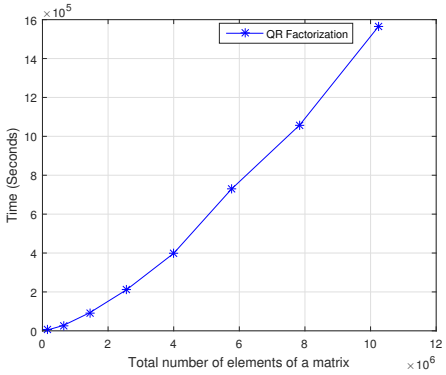
In this paper, we have investigated the problem of privacy-preserving outsourcing large-scale QR factorization to the cloud. Specifically, to protect data privacy, we have developed the privacy-preserving matrix transformations

TABLE 1. COMPUTING TIME COMPARISON OF LOCAL QR FACTORIZATION AND THE PROPOSED ALGORITHM FOR QR FACTORIZATION.

| Elements in a matrix $A$ | Local QR factorization | QR factorization at the user | User's speedup |
|---|---|---|---|
| $1.6\times10^5$ | 1003s | 58s | 17.3 |
| $6.4\times10^5$ | 14795s | 261s | 56.7 |
| $1.44\times10^6$ | 57426s | 638s | 90.01 |
| $2.56\times10^6$ | 140612s | 1149s | 121.97 |
| $4\times10^6$ | 297319s | 1831s | 162.4 |
| $5.76\times10^6$ | 542514s | 2381s | 227.9 |
| $7.84\times10^6$ | 873885s | 3409s | 256.4 |
| $1.024\times10^7$ | 1297549s | 3953s | 328.3 |




(a) Computing time running at the user for different data size.




(b) Computing time running at the cloud for different data size.

Figure 2. Computing time of the privacy-preserving outsourcing QR facotorization at the user and the cloud for different data sizes.

based on linear algebra operations. Employing the developed matrix transformations, we have proposed a privacy-preserving outsourcing QR factorization algorithm, where the user transforms his/her original matrix and the cloud performs QR factorization on the transformed matrix to output the factor matrices in a non-interactive way. Hence, the proposed algorithm requires the lower computational complexity at the user and the small communication overhead between the user and the cloud. We have implemented the proposed algorithm on the Amazon Elastic Compute Cloud (EC2) platform and a laptop, and the experimental results show that the proposed algorithm can offer significant time saving to the user.

## References


[1] J. Gantz and D. Reinsel, "Extracting value from chaos," *IDC iView*, Jun. 2011.

[2] M. Kezunovic, V. Vittal, S. Meliopoulos, and T. Mount, "The big picture: Smart research for large-scale integrated smart grid solutions," *IEEE Power and Energy Magazine*, vol. 10, pp. 22–34, Jul./Aug. 2012.

[3] Z. Khan, A. Anjum, and S. L. Kiani, "Cloud based big data analytics for smart future cities," in *Proc. IEEE/ACM UCC'09*, (Washington, DC, USA), pp. 381–386, Dec. 9 - 12 2013.

[4] L. Qi, X. Xu, X. Zhang, W. Dou, C. Hu, Y. Zhou, and J. Yu, "Structural balance theory-based e-commerce recommendation over big rating data," *IEEE Trans. Big Data*, vol. PP, no. 99, pp. 1–1, 2016.

[5] M. Kezunovic, L. Xie, and S. Grijalva, "The role of big data in improving power system operation and protection," in *Proc. IEEE IREP Symposium-Bulk Power System Dynamics and Control*, (Rethymnon, Greece), pp. 1–9, Aug. 25 - 30 2013.

[6] L. Liu, A. Hou, A. Biderman, C. Ratti, and J. Chen, "Understanding individual and collective mobility patterns from smart card records: A case study in shenzhen," in *Proc. IEEE ITSC'09*, (St. Louis, MO, USA), pp. 842–847, Oct. 4 - 7 2009.

[7] G. H. Golub and C. F. V. Loan, *Matrix Computations, 4th Edition*. Baltimore MD: Johns Hopkins University Press, 2013.

[8] S. Pajic and K. A. Clements, "Power system state estimation via globally convergent methods," *IEEE Trans. Power Syst.*, vol. 20, pp. 1683–1689, Nov. 2005.

[9] A. Pandey and S. Shrotriya, "Comparing the effect of matrix factorization techniques in reducing the time complexity for traversing the big data of recommendation systems," *Int'l Journal of Comput. and Commu. Eng.*, vol. 2, pp. 170–173, Mar. 2013.

[10] R. Mathias and G. W. Stewart, "A block QR algorithm and the singular value decomposition," *Linear Algebra and Its Applications*, vol. 182, pp. 91–100, Mar. 1993.

[11] A. R. Benson, D. F. Gleich, and J. Demmel, "Direct QR factorisations for tall-and-skinny matrices in MapReduce architectures," in *Proc. IEEE Int'l Conf. Big Data*, (Santa Clare, CA, USA), pp. 264–272, Oct. 6 - 9 2013.

[12] G. W. Stewart, *Introduction to Matrix Computations*. NJ: Academic Press, 1974.

[13] E. L. Quinn, "Privacy and the new energy infrastructure," *Social Science ResearchNetw.*, pp. 1995–2008, 2009 [Online]. Available: http://ssrn.com/pager=1370731.

[14] J. Zhang, F. Wang, K. Wang, W. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: A survey," *IEEE Trans. Intelligent Transportation Systems*, vol. 12, pp. 1624–1639, Jul. 2011.

[15] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. Advances in Cryptology (CRYPTO'10)*, (Santa Barbara, CA, USA), pp. 465–482, Aug. 15 - 19 2010.

[16] C. Wang, K. Ren, J. Wang, and Q. Wang, "Harnessing the cloud for securely outsourcing large-scale systems of linear equations," *IEEE Trans. Parallel and Distributed Systems*, vol. 24, pp. 1172–1181, Jun. 2013.

[17] S. Salinas, C. Luo, X. Chen, and P. Li, "Efficient secure outsourcing of large-scale linear systems of equations," in *Proc. IEEE INFOCOM'15*, (Hongkong, China), pp. 1035–1043, Apr. 26 - May 1 2015.

[18] S. Salinas, C. Luo, W. Liao, and P. Li, "Efficient secure outsourcing of large-scale quadratic programs," in *Proc. ACM Asia CCS'16*, (Xi'an, Shaanxi, China), 30 May – 1. Jun 2016.

[19] M. K. Gupta, "Akka essentials," *Packt Publishing*, vol. ISBN-13:978-1-84951-828-4, 2012.