1-2022

Novel secure outsourcing of modular inversion for arbitrary and variable modulus

Chengliang TIAN

Jia YU

Hanlin ZHANG

Haiyang XUE
Singapore Management University, haiyangxue@smu.edu.sg

Cong WANG

See next page for additional authors

Novel Secure Outsourcing of Modular Inversion for Arbitrary and Variable Modulus

Chengliang Tian, Jia Yu, Hanlin Zhang, Haiyang Xue, Cong Wang, *Senior Member, IEEE*, and Kui Ren, *Fellow, IEEE*

Abstract—In cryptography and algorithmic number theory, modular inversion is viewed as one of the most common and time-consuming operations. It is hard to be directly accomplished on resource-constrained clients (e.g., mobile devices and IC cards) since modular inversion involves a great amount of operations on large numbers in practice. To address the above problem, this paper proposes a novel unimodular matrix transformation technique to realize secure outsourcing of modular inversion. This technique makes our algorithm achieve several amazing properties. First, to the best of our knowledge, it is the first secure outsourcing computation algorithm that supports arbitrary and variable modulus, which eliminates the restriction in previous work that the protected modulus has to be a fixed composite number. Second, our algorithm is based on the single untrusted program model, which avoids the non-collusion assumption between multiple servers. Third, for each given instance of modular inversion, it only needs one round interaction between the client and the cloud server, and enables the client to verify the correctness of the results returned from the cloud server with the (optimal) probability 1. Furthermore, we propose an extended secure outsourcing algorithm that can solve modular inversion in multi-variable case. Theoretical analysis and experimental results show that our proposed algorithms achieve remarkable local-client's computational savings. At last, as two important and helpful applications of our algorithms, the outsourced implementations of the key generation of RSA algorithm and the Chinese Reminder Theorem are given.

Index Terms—Cloud computing, modular inversion, unimodular matrix transformation, efficiency, privacy

1 Introduction

Cloud computing is a type of Internet-based computing that offers processing resources and data to electronic devices on demand. It is a model that enables network access to a shared pool of configurable computing resources at any time, at any place [2], [21]. As one of the typical service delivery types in cloud computing, more and more cloud servers are providing alternative and economic platform services for resource-constrained clients [31]. By leveraging the cloud computing platforms, the resource-constrained clients can perform large-scale computations and data storage without the investment of purchasing and maintaining their own computing facilities. However, the client and the cloud server are not necessarily in the same trusted domain, which brings many security concerns and challenges towards this promising computation model [8], [23], [34]. First and foremost, the client's computation task outsourced to the cloud server often contains sensitive information, such as business financial records and client's private keys. Once the critical information is exposed to the cloud, it might bring huge lost to the client. Second, the result returned from the cloud might be invalid. For example, due to financial incentives, the cloud server may be lazy, curious, or even malicious, and therefore may return incorrect answers. Besides, some accidental reasons such as software bugs, hardware failures or outsider attacks can also result in wrong computational results. Last but not the least, the cost of delegating computation task must be less than that of doing the work locally from scratch. In summary, it has been widely established that a "well defined" secure outsourcing computation algorithm should at least satisfy the following requirements [15], [18]:

1) *Privacy*. The secure outsourcing computation algorithm must protect the secrecy of the input and the output of computation task. In other words, cloud server should not learn anything about what it is actually computing.
2) *Verifiability*. The client should be able to detect the misbehaviour of cloud server.
3) *Efficiency*. The cost of task generation and result verification for the client should be substantially cheaper than performing the computation on its own, otherwise the outsourcing will become meaningless.

- *C. Tian and J. Yu are with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China, and also with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. E-mail: tianchengliang@qdu.edu.cn, qduyujia@gmail.com.*
- *H. Zhang is with the College of Computer Science and Technology, Qingdao University, Qingdao 266071, China. E-mail: hanlin@qdu.edu.cn.*
- *H. Xue is with the State Key Laboratory of Information Security, Chinese Academy of Sciences, Beijing 100093, China, and also with the Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing 100093, China. E-mail: xuehaiyang@iie.ac.cn.*
- *C. Wang is with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: congwang@cityu.edu.hk.*
- *K. Ren is with the Institute of Cyberspace Research, Zhejiang University, Hangzhou 310007, China. E-mail: kuiren@buffalo.edu.*

Manuscript received 10 Aug. 2018; revised 13 July 2019; accepted 21 Aug. 2019. Date of publication 26 Aug. 2019; date of current version 4 Feb. 2022. (Corresponding author: Jia Yu.)
Digital Object Identifier no. 10.1109/TSC.2019.2937486

Modular inversion is one of the most common operations in cryptography and algorithmic number theory. Given two coprime integers a and n, modular inversion solves the linear congruence

$$ax \equiv 1 \mod n, \tag{1}$$

which in nature is the division operation over the residue class ring $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. It is well known that the division operation is the most time-consuming one among the four fundamental operations "addition", "subtraction", "multiplication" and "division". The classic algorithm of solving this problem is the famous extended Euclidean algorithm which computes two integers u, v such that $1 = \gcd(a, n) = au + nv$ with $O(\log^3 n)$-bit operations. Then the solution is $x = u \bmod n$. The total time cost is polynomial in the size of input which makes the algorithm feasible. However, it is far from being the whole story. In practice, with the fast development of information technology and mobile internet, more and more mobile intelligent clients and embedded devices, such as RFID tags, smart cards and cell phones, are connected with other components of their environment. These clients are usually limited in computing resources [19], [24], [25]. Meanwhile, the modular inversion, required by virtually any public-key algorithm, often involves handling large numbers. For example, during the key-generation phase of the celebrated RSA public-key scheme, one needs to randomly choose a large public key e satisfying $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$, and then computes the secret key d such that $ed \equiv 1 \bmod \phi(n)$, where $\phi(\cdot)$ is the Euler totient function, and n is a large number with bit length no less than 1,024. Moreover, because of the primitiveness of the division operation and the exponential growth quantity in data nowadays, it often needs to dispose a large number of modular inversion operations simultaneously in practice. It is difficult for resource-constrained clients to perform massive heavy computations. Fortunately, the cloud computing makes it possible for the resource-constrained clients to accomplish these operations.

1.1 Related Work

There are many researches on how to securely outsource various computation-expensive work. Currently, the researches mainly move towards two directions. One is to devise a generic model that outsources arbitrary scientific computations with fully homomorphic encryption (FHE) technology [7], [16]. This technology converts the computations on the plaintext into the computations on the ciphertext. Meanwhile, the encrypted result, when decrypted, matches the result of operations performed on the plaintext. Using FHE technology, the resource-constrained client can send the encrypted data to the cloud server, and then decrypt the computation result from the could server to the plaintext using his/her private key. Although great efforts have been made to optimize the FHE algorithms [3], [4], [11], [17], the existing algorithms still suffer from very high computation cost.

The other direction is to devise specific secure outsourcing computation algorithms for some concrete heavy computational tasks. For example, as a popular topic in the present, secure outsourcing of modular exponentiations involving large numbers has been extensively investigated. Hohenberger and Lysyanskaya [18] proposed the first outsource-secure algorithm for variable-exponent variable-base exponentiations with a strict proof of theoretical security. But the probability of verifiability was only $\frac{1}{2}$. Chen et al. [6] improved their secure outsourcing computation algorithm in both efficiency and verifiability. The above two algorithms were based on one-malicious version of two untrusted program model, which made too strong an assumption that the two untrusted programs are non-collude. In order to solve this problem, Wang et al. [28] presented the first algorithm to securely outsource variable-exponent variable-base exponentiations based on one untrusted program model, but the probability of verifiability was still $\frac{1}{2}$. Ding et al. [10] introduced a new secure outsourcing computation algorithm of modular exponentiation based on one untrusted program model which achieved higher checkability. However, all the aforementioned algorithms exposed the modulus completely while, in some applications such as the decryption of RSA cryptosystem [1], the modulus contains sensitive information which needs to be protected from the cloud server. Zhou et al. [33] designed a new secure outsourcing computation algorithm based on one untrusted program model, which not only concealed the exponent and the base, but also concealed the modulus. In terms of efficiency, without outsourcing, the local client would need $O(\log^3 n)$-bit operations to carry out a modular exponentiation for some modulus n and any exponent no bigger than n. Meanwhile, the client's computational burden is reduced to $O(\log^2 n)$ by outsourcing algorithms mentioned above. Besides, outsourcing other computation tasks such as polynomial evaluation [13], [14], large matrix operations [20], linear programming [27], large-scale system of linear equations solving [5], min-cut of graphs [32] and key updates in cloud storage auditing [29] have been studied. Compared with the generic model, secure outsourcing computation algorithm devised for specific operation usually achieves higher efficiency.

Recently, Su et al. [26] have studied the problem of outsourcing the inversion modulo a large composite number with two known factors, which is viewed as a special case of the Equation (1). By employing Chinese Reminder Theorem (CRT), they proposed an efficient secure outsourcing computation algorithm based on one-malicious version of two untrusted program model. However, the prior art is not sufficient in all contexts, and there still exist some critical challenges yet to be fully addressed. First, the previous work cannot be applied to the case where the modulus is not a composite number with known factors. This is indeed a common case in practice. For example, in the ElGamal encryption and signature schemes [12], we need to compute the inversion modulo a large prime. Second, in previous design, the preprocessing can only be executed under the condition that the modulus has been known in [26], and therefore the modulus must be fixed. Again, these preconditions somewhat limit the wide applicability of the design, and further demands solutions with more flexibility, such as the case of arbitrary and variable modulus. Third, the assumption of two non-colluding cloud servers in prior art in general is not easy to be met in practice [33]. Consequently, how to securely outsource modular inversion for arbitrary and variable modulus based on single untrusted program model, which is much more desired in practice, is left as an open problem.

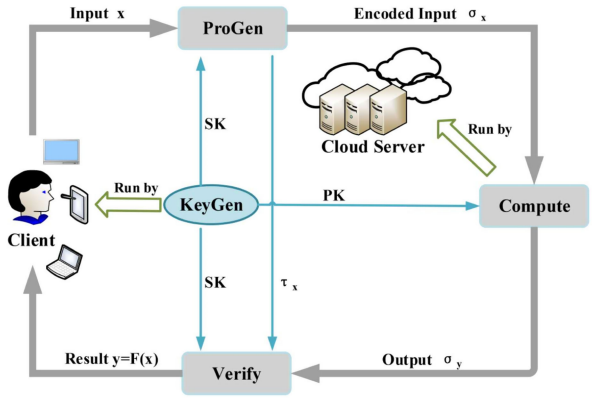


Fig. 1. The system model.

1.2 Our Contributions

To address the above problems, this paper further explores the properties of linear congruence (1), and presents a new secure outsourcing computation algorithm for modular inversion which essentially is an outsourcing implementation of the extended Euclidean algorithm. Further, we generalize our algorithm to outsource modular inversion in multi-variable case. Compared with previous work, our algorithms maintain the property that the client can detect the incorrect answer from the dishonest server with probability 1, and meanwhile are superior in the following aspects:

1) The proposed algorithms can support arbitrary and variable modulus. We give a novel unimodular matrix transformation technique, which takes advantage of a significant property that unimodular matrix transformation keeps the greatest common divisor of two integers invariant, to realize secure outsourcing of modular inversion for arbitrary and variable modulus.
2) The proposed algorithms are designed under single untrusted program model. According to the model defined in the paper, we give a rigorous theoretical analysis on the security of the proposed algorithms, which indicates our algorithms satisfying the verifiability and the one-way argument of input/output privacy.
3) Our algorithms achieve decent local-client's computational savings. They only need $O(l^2)$ bit operations for some given modulus of l bits in contrast with, without outsourcing, $O(l^3)$ bit operations on the client side. Specially, in the single-variable case, our experimental results demonstrate that the proposed algorithm performs about 9 to 14× faster than the algorithm without outsourcing and 5.8 to 8.3× faster than the algorithm in [26].
4) Our algorithms have found wide applications in some fundamental algorithms in cryptography and coding theory, e.g., the secure outsourcing for the key generation of RSA algorithm and the solving of linear congruences.

1.3 Road Map

The rest of this paper is organized as follows: In Section 2, we present the system model and the related security definitions. Section 3 reviews some necessary preliminaries of our algorithms. We propose two secure outsourcing computation algorithms and prove their correctness, security and efficiency in Section 4. The practical performance evaluation of the proposed algorithms is given in Section 5. Section 6 presents two important applications of our algorithms. Finally, we conclude our paper in Section 7.

2 System Model and Security Definitions

2.1 System Model

A secure outsourcing computation model involves two entities: the resource-constrained client C and the cloud server S with powerful computational resource yet maybe curious, or even malicious. With the help of the cloud server S, the client C intends to accomplish a computational task F with an input x. At a high level, the client C and the cloud server S perform a two-party protocol as follows: C sends an encoding of the computational task F and the blinded input σ_x to S. S is expected to complete the computational task associated with σ_x and return the output to C. The client C verifies correctness of the result given by S and recovers the desired value if the returned results passed the verification.

We show the system model in Fig. 1. Formally, following the model in [15], a secure outsourcing computation algorithm consists of the following four-tuple $\mathcal{SOC}_{\mathcal{F}} = (\textbf{KeyGen}, \textbf{ProbGen}, \textbf{Compute}, \textbf{Verify})$.

1) $\textbf{KeyGen}(F, \lambda) \rightarrow (PK, SK)$: Taken the inputs as the security parameter λ and the target function F, the *key generation* algorithm outputs a public key PK that encodes the target function F, and the corresponding private key SK which is kept private by the client C.
2) $\textbf{ProbGen}_{SK}(x) \rightarrow (\sigma_x, \tau_x)$: Given an input x to the function F, the *problem generation* algorithm uses the private key SK to encode x as a secret value τ_x and a public value σ_x, and then sends σ_x to the server S.
3) $\textbf{Compute}_{PK}(\sigma_x) \rightarrow \sigma_y$: Utilizing the public key PK and the encoded input σ_x given by C, the *computation* algorithm outputs σ_y, which is an encoded version of $y = F(x)$.
4) $\textbf{Verify}_{SK}(\tau_x, \sigma_y) \rightarrow y \cup \perp$: Utilizing the private key SK and the secret "decoding" τ_x, the *verification* algorithm verifies whether σ_y is valid. If the result is valid, the client converts σ_y into the real $y = F(x)$ as the output. Otherwise, the client outputs $\perp$.

2.2 Threat Model

The threats in our outsourcing system mainly come from three aspects: attacks from the (malicious) cloud server, attacks from external adversaries and occasional accidents from internal software or hardware failures. However, from the client's point of view, any extraneous attack will be attributed to the server's misbehaviors. Therefore, according to the server's behaviors and the number of servers, all the threats can be included into four models: the lazy single-server (LS) model, the honest but curious single-server (HCS) model, the malicious single-server (MS) model and the malicious multiple-server (MM) model.

Lazy Single-Server (LS) Model. In LS model, the cloud server still performs the protocol specification, but, for

saving computational resources, may return a random or intermediate result.

Honest but Curious Single-Server (HCS) Model. In HCS model (also known as the "semi-honest" single-server model), the cloud server is assumed to faithfully execute the delegated computation task and returns the correct result to the client. However, in order to profit from some sensitive computational tasks, the cloud may be curious about the intermediate data and try to learn or infer the protected information.

Malicious Single-Server (MS) Model. In MS model, the cloud server won't execute its advertised functionality, even arbitrarily deviates from the protocol specification and tries to learn sensitive information as much as it can. Worse still, it may intentionally return a random or forged result to fool the client.

Malicious Multiple-Servers (MM) Model. In MM model, the client assigns the computation task to multiple malicious cloud servers under the assumption of their non-collusion.

Clearly, under the LS model, a secure outsourcing algorithm should guarantee the verifiability of the returned result from, and, under the HCS model, a secure outsourcing algorithm should ensure the input/output privacy of client's data. While, under the MS model, a secure outsourcing algorithm should satisfy both of the above two requirements and, simultaneously, doesn't need the multiple non-collude servers in the MM model. Consequently, from the perspective of security, designing an outsourcing algorithm under the MS model for some given computation task is more significative than that of under LS, HCS or MM model.

2.3 Design Goals

In pursuit of an efficient and secure outsourcing algorithm for modular inversion under the MS model, we identify the following four design goals, which are consistent with the existing works [15], [30] except an adjustment on the input/output privacy.

2.3.1 Correctness

A secure outsourcing computation algorithm should first satisfy correctness. An algorithm is correct if, for any valid input, when the server correctly executes the algorithm **Compute**, the output of algorithm **Verify** is the evaluation of F on this input. More formally,

Definition 1 (Correctness [15]). *A secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{F}}$ of some computation task F is correct if the key generation algorithm produces key $(PK, SK) \leftarrow \mathbf{KeyGen}(F, \lambda)$ such that, for any $x \in \mathrm{Domain}(F)$, if $(\sigma_x, \tau_x) \leftarrow \mathbf{ProbGen}_{SK}(F, x)$ and $\sigma_y \leftarrow \mathbf{Compute}_{PK}(\sigma_x)$, then $y = F(x) \leftarrow \mathbf{Verify}_{SK}(\tau_x, \sigma_y)$.*

2.3.2 Verifiability

Verifiability means a malicious server cannot persuade the *verification* algorithm to accept an incorrect output. Namely, for any given computation F and randomly chosen input $x \in \mathrm{Domain}(F)$, the probability that a malicious server is able to convince the verification algorithm to output $\hat{y}$ such that $\hat{y} \notin \{F(x), \perp\}$ is negligible. Below, we formalize its notion with the following experiment.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{verif}[F, \lambda]$
$(PK, SK) \leftarrow \mathbf{KeyGen}(F, \lambda)$
Query and response:
$x_0 = \sigma_{x_0} = \beta_0 = \perp$.
For $i = 1, \ldots, \ell = \mathrm{poly}(\lambda)$
$x_i \leftarrow \mathcal{A}(PK, x_0, \sigma_{x_0}, \beta_0, \ldots, x_{i-1}, \sigma_{x_{i-1}}, \beta_{i-1})$.
$(\tau_{x_i}, \sigma_{x_i}) \leftarrow \mathbf{ProGen}_{SK}(x_i)$.
$\sigma_{y_i} \leftarrow \mathcal{A}(PK, x_0, \sigma_{x_0}, \beta_0, \ldots, x_{i-1}, \sigma_{x_{i-1}}, \beta_{i-1}, \sigma_{x_i})$.
$\beta_i \leftarrow \mathbf{Verify}_{SK}(\tau_{x_i}, \sigma_{y_i})$.
Challenge:
$x \leftarrow \mathcal{A}(PK, x_1, \sigma_{x_1}, \beta_1, \ldots, x_l, \sigma_{x_\ell}, \beta_\ell)$.
$(\tau_x, \sigma_x) \leftarrow \mathbf{ProGen}_{SK}(x)$.
$\sigma_y \leftarrow \mathcal{A}(PK, x_1, \sigma_{x_1}, \beta_1, \ldots, x_\ell, \sigma_{x_\ell}, \beta_\ell, \sigma_x)$.
$\hat{y} \leftarrow \mathbf{Verify}_{SK}(\tau_x, \sigma_y)$.
if $\hat{y} \neq F(x)$ and $\hat{y} \neq \perp$, output $'1'$;
else output $'0'$.

In the query and response phase, the adversary (e.g., malicious server) is given oracle access to the public output of **ProbGen** and the sub-algorithm **Verify**. In the challenge phase, the adversary is successful if it produces a forged output that convinces the verification algorithm to accept the wrong output. Now, we define the verifiability of a secure outsourcing computation algorithm based on the probability of the adversary's success.

Definition 2 (Verifiability). *A secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{F}}$ of computation task F is verifiable if for any adversary $\mathcal{A}$ running in probabilistic polynomial time, the probability of outputting 1 in Experiment $\mathbf{Exp}_{\mathcal{A}}^{verif}[F, \lambda]$ is negligible, i.e., $\Pr[\mathbf{Exp}_{\mathcal{A}}^{verif}[F, \lambda] = 1] \leq \mathtt{negli}(\lambda)$, where $\mathtt{negli}()$ is a negligible function of its input.*

2.3.3 Input/Output Privacy

The definition of input/output privacy in [15], [30] is based on a typical indistinguishability argument that ensures that any probabilistic polynomial-time adversary $\mathcal{A}$ cannot distinguish the ciphertexts of two different plaintexts. However, in most of our applications, one-way security is enough. That is, given the public information (e.g., the public key PK and the public value σ_x), the probabilistic polynomial-time adversary $\mathcal{A}$ can not recover the input x and the output $F(x)$. Here we formalize the one-way argument of privacy, including input privacy and output privacy, with the experiment as follows.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[F, \lambda]$
$(PK, SK) \leftarrow \mathbf{KeyGen}(F, \lambda)$
Query and response:
$x_0 = \sigma_{x_0} = \perp$.
For $i = 1, \ldots, \ell = \mathrm{poly}(\lambda)$
$x_i \leftarrow \mathcal{A}(PK, x_0, \sigma_{x_0}, \ldots, x_{i-1}, \sigma_{x_{i-1}})$.
$(\tau_{x_i}, \sigma_{x_i}) \leftarrow \mathbf{ProGen}_{SK}(x_i)$.
Challenge:
$x^\star \leftarrow \mathrm{Domain}(F)$.
$(\tau_{x^\star}, \sigma_{x^\star}) \leftarrow \mathbf{ProGen}_{SK}(x^\star)$.
$x' \leftarrow \mathcal{A}(PK, x_0, \sigma_{x_0}, \ldots, x_\ell, \sigma_{x_\ell}, \sigma_{x^\star})$.
if $x' = x^\star$, output $'1'$;
else output $'0'$.

In the query and response phase, the adversary is given the oracle access to obtain the public output of the *problem generation* algorithm, and allowed to invoke the algorithm polynomial times. In the challenge phase, based on the information gathered in the query and response phase and given the public key PK and a public value σ_{x^*}, the adversary tries to acquire the input, and succeeds if it can recover x^*.

Definition 3 (Input privacy). *A secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{F}}$ of computation task F is input-private if for any adversary $\mathcal{A}$ running in probabilistic polynomial time, the probability of outputting 1 in Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[F,\lambda]$ is negligible, i.e.,*

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[F,\lambda]=1]\leq \mathtt{negli}(\lambda),$$

where $\mathtt{negli}()$ *is a negligible function of its input.*

Similarly, the security requirement of output privacy can be defined by the following experiment.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[F,\lambda]$
$(PK,SK)\leftarrow \mathbf{KeyGen}(F,\lambda)$
Query and response:
$x_0=\sigma_{x_0}=\beta_0=\perp$.
For $i=1,\ldots,\ell=\mathrm{poly}(\lambda)$
$x_i\leftarrow \mathcal{A}(PK,x_0,\sigma_{x_0},\beta_0,\ldots,x_{i-1},\sigma_{x_{i-1}},\beta_{i-1})$.
$(\tau_{x_i},\sigma_{x_i})\leftarrow \mathbf{ProGen}_{SK}(x_i)$.
$\sigma_{y_i}\leftarrow \mathcal{A}(PK,x_0,\sigma_{x_0},\beta_0,\ldots,x_{i-1},\sigma_{x_{i-1}},\beta_{i-1},\sigma_{x_i})$.
$\beta_i\leftarrow \mathbf{Verify}_{SK}(\tau_{x_i},\sigma_{y_i})$.
Challenge:
$x^*\leftarrow \mathrm{Domain}(F)$.
$(\tau_{x^*},\sigma_{x^*})\leftarrow \mathbf{ProGen}_{SK}(x^*)$.
$\sigma_{y^*}\leftarrow \mathbf{Compute}_{PK}(\sigma_{x^*})$.
$f'\leftarrow \mathcal{A}(PK,\sigma_{x^*},\sigma_{y^*},(x_j,\sigma_{x_j},\beta_j)_{j=0,\ldots,\ell})$.
if $f'=F(x^*)$, output $'1'$;
else output $'0'$.

Definition 4 (Output privacy). *A secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{F}}$ of some computation task F is output-private if for any adversary $\mathcal{A}$ running in probabilistic time, the probability of outputting 1 in Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[F,\lambda]$ is negligible, i.e.,*

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[F,\lambda]=1]\leq \mathtt{negli}(\lambda),$$

where $\mathtt{negli}()$ *is a negligible function of its input.*

2.3.4 Efficiency

Efficiency is a basic requirement for a secure outsourcing computation algorithm, which requires that the client's cost of blinding the input and verifying the output must be substantially cheaper than that of computing the computational task from scratch locally. Since the **KeyGen** step is independent of the input, it can be preprocessed and its time overhead is not contained in the client's cost.

Definition 5 (α-Efficient [15]). *A secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{F}}$ of some computation task F is α-Efficient if the time required for $\mathbf{ProbGen}_{SK}(x)$ and $\mathbf{Verify}_{SK}(\tau_x,\sigma_y)$ is no more than an α multiplicative factor of T, where T is the fastest known time of computing $F(x)$.*

3 Preliminaries

To be self-contained, this section will briefly introduce some necessary notations and basic concepts used in the rest of the paper.

3.1 Notations and Terminologies

We use bold lower case letters to denote vectors, and use upper case letters to denote matrices. $\mathbf{M}^{\mathrm{T}}$ (resp. $\mathbf{a}^{\mathrm{T}}$) denotes the transposition of matrix $\mathbf{M}$ (resp. vector $\mathbf{a}$), and $\mathbb{Z}^{n\times m}$ denotes all of the $n\times m$ matrices whose entries are integers.

3.2 Unimodular Matrix

Unimodular matrix is a special invertible matrix which has numerous applications in matrix theory and lattice-based cryptography [22].

Definition 6 (Unimodular matrix). *An $n\times n$ matrix $\mathbf{U}$ is unimodular if and only if $\mathbf{U}\in\mathbb{Z}^{n\times n}$ and the absolute value of its determinant $|\det(\mathbf{U})|=1$.*

A simple property of unimodular matrix is that the inverse of a unimodular matrix is also unimodular. We list it as a lemma and omit its proof.

Lemma 1 ([22]). *If $\mathbf{U}\in\mathbb{Z}^{n\times n}$ is unimodular, then there exists a unique matrix $\mathbf{V}\in\mathbb{Z}^{n\times n}$ s. t. $|\det(\mathbf{V})|=1$ and $\mathbf{UV}=\mathbf{I}_{n\times n}$, where $\mathbf{I}_{n\times n}$ denotes the identity matrix.*

For example, let $\mathbf{U}=\begin{pmatrix}2&1\\5&2\end{pmatrix}$ be an unimodular matrix with $\det(\mathbf{U})=2\times 2-5\times 1=-1$. Then it is easy to verify that its inverse $\mathbf{V}=\begin{pmatrix}-2&1\\5&-2\end{pmatrix}$ is also unimodular.

3.3 Extended Euclidean Algorithm

The well-known extended Euclidean algorithm is an extension of the Euclidean algorithm. Given two integers a and n, the algorithm computes two integers x and y beside the greatest common divisor of integers a and n, such that $ax+ny=\gcd(a,n)$. It needs $2\log n$ divisions and $6\log n$ multiplications in the worst case, and therein the total time complexity is $O(\log^3 n)$. The pseudocode is shown in Algorithm 1.

Algorithm 1. $\mathtt{extended\_gcd}(a,n)$

Require: Integers a,n.
Ensure: Three integers d,x,y satisfying $d\,|\,a$, $d\,|\,n$ and $ax+ny=d$.
1: $x=0$, old_x = 1, $y=1$, old_y = 0, $r=n$, old_$r=a$
2: While ($r\neq 0$)
3: $q=\lfloor \text{old\_}r/r\rfloor$
4: $t=r, r=\text{old\_}r-q*t, \text{old\_}r=t$
5: $u=x, x=\text{old\_}x-q*u, \text{old\_}x=u$
6: $v=y, y=\text{old\_}y-q*v, \text{old\_}y=v$
7: Output $d=\text{old\_}r, \text{old\_}x, \text{old\_}y$

4 Secure Outsourcing Algorithms of Modular Inversion

4.1 Computation Task Description and Basic Idea

Given two coprime integers a and n, the client wants to find the unique $x \in \mathbb{Z}_n$ such that $ax \equiv 1 \mod n$. The classic method of solving this problem is to compute two integers u and v such that $au + nv = 1$ by the extended Euclid algorithm. Then, the solution is $x = u \mod n$. The procedure needs $O(\log^3 n)$ bit operations. Since the integers involved in the cryptographic community are usually very large. It is difficult for resource-limited client to carry out such expensive operations. Under this circumstance, the client may accomplish this task by secure outsourcing computation.

In order to keep the privacy of the input, we must blind the values of a and n. As a sequence, the result returned from the cloud server will not actually be the inversion of a modulo n. In order to ensure that the original result can be recovered from the blinded result, the input values and their blinded ones must satisfy certain equivalence relation. The unimodular matrix has a characteristic that the inversion of a unimodular matrix is still unimodular. This property implies an equivalence relation in the sense that the greatest common divisor of the input values equals that of their blinded ones. By taking advantage of this property, we propose a novel secure outsourcing computation algorithm of modular inversion. More precisely, the client first chooses a random unimodular matrix $\mathbf{U} \in \mathbb{Z}^{2\times 2}$, blinds the input by computing $(a'\ n')^T = \mathbf{U}(a\ n)^T$, and sends them to the cloud server. And then the cloud server performs the extended Euclid algorithm on the blinded values and returns the result to the client. At last, the client verifies the received values and recovers the solution. The security is based on the randomness of unimodular matrix $\mathbf{U}$, and the correctness follows the intrinsic property of unimodular matrix transform that keeps the greatest common factor of a and n invariant. Furthermore, we employ this novel technique to securely outsource the modular inversion in multi-variable case.

4.2 Outsourcing Algorithm of $ax \equiv 1 \mod n$

Given two coprime integers a and n, the proposed algorithm $\mathcal{SOC}_{\mathcal{MI}}(a, n)$ consists of four sub-algorithms as follows:

1) **Prepro**: Given a security parameter λ, the client C randomly generated an unimodular matrix

$$\mathbf{U} = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \in SK_{\mathcal{MI}},$$

where

$$SK_{\mathcal{MI}} = \{\mathbf{U}_i \mid \mathbf{U}_i \in K_\lambda^{2\times 2} \text{ and } |\det(\mathbf{U}_i)| = 1\},$$

and $K_\lambda = \mathbb{Z} \cap (-2^\lambda, 2^\lambda)$ denotes the set of integers with bit length no larger than λ. This step should be preprocessed before the following three steps.

2) **ProbGen**: Using the secret matrix $\mathbf{U}$, C blinds the input a and n by computing two integers a' and n'

$$a' = u_{11}a + u_{12}n, n' = u_{21}a + u_{22}n,$$

and sends (a', n') to the cloud server S.

3) **Compute**: Receiving the blinded values (a', n'), the cloud server S uses the extended euclidean algorithm to compute two integers x', y' such that $a'x' + n'y' = 1$. The cloud server returns (x', y') to C.

4) **Verify**: Receiving the encoded output (x', y'), the client C verifies whether the equation $a'x' + n'y' = 1$ holds. If it does, C computes

$$x = (u_{11}x' + u_{21}y') \mod n,$$

as the solution. Otherwise, C outputs "$\perp$" and claims the misbehavior of S.

Remark 1. Since the computation task F in our construction is modular inversion which is concrete, unlike the generic model for arbitrary target function in Section 2.1, there is no **KeyGen** step in our construction. To be consistent with the generic model, the public key PK can be seen as the algorithm that on input two coprime (a', n') outputs two integers x', y' subject to $a'x' + n'y' = 1$, and the secret key SK can be seen as the collection of the secret 2-by-2 unimodular matrices.

Remark 2. In the preprocessing stage, a feasible method of generating a random unimodular matrix is as follows: Randomly generate two coprime integers $u_{11}, u_{21} \in K_\lambda$. And then, run the extended euclidean algorithm to compute two integers u_{12} and u_{22} such that $u_{11}u_{22} - u_{12}u_{21} = 1$. Noteworthily, the randomness of $\mathbf{U}$ has great impact on the input/output privacy of the scheme. If the elements in $\mathbf{U}$ are too small, the privacy is offended. If the elements in $\mathbf{U}$ are too large, the efficiency of the scheme is affected. In practice, we should balance the security and the efficiency. To be immune against the brute-force attack, the security parameter λ should be at least 80. It will not reduce the performance gain too much, since, generally, outsourcing computation is employed when the integers are large, usually no less than 512 bit, and the local-client's computational savings obtained by outsourcing are about $O(\log n)$ muiltiplications/divisions with large numbers. For more details please refer to Section 5.

Remark 3. Noteworthily, if $n = p$ is a prime, our algorithm can be seen as an outsourcing implementation of Itoh-Tsujii inversion in the prime field $GF(p)$. To securely outsource Itoh-Tsujii inversion in a general large-scale finite field $GF(p^m)$, since $GF(p^m)$ is isomorphic with the residue class field $GF(p)[x]/(n(x))$, we can adapt the proposed algorithm to two large-scale polynomials $a(x)$, $n(x) \in GF(p)[x]$, where $n(x)$ is an irreducible polynomial with degree m. Nonetheless, the construction of random unimodular matrices should be different and carefully discussed. We leave it for our future work.

Remark 4. In practice, it is common to simultaneously or successively compute many instances of modular inversion, e.g., $a_i^{-1} \bmod n_i$ for m pairs of large integers (a_i, n_i), where $1 \le i \le m$ and m is a positive integer. In this case, we can precompute m unimodular matrices, and in parallel or serially invoke our proposed algorithm.

In order to make our algorithm more transparent, we give a toy example: Given two integers 13 and 15, the client

wants to find an integer x $(1 \le x < 15)$ such that $13 \cdot x \equiv 1 \bmod 15$. The secure outsourcing computation algorithm runs as follows (we omit the processing step):

(1) The client randomly chooses a unimodular matrix $\mathbf{U} = \begin{pmatrix} 5 & -2 \\ 7 & -3 \end{pmatrix}$. Obviously, $\det(\mathbf{U}) = -1$.
(2) The client blinds the input $(a, n) = (13, 15)$ as $(a', n') = (5 \cdot a + (-2) \cdot n, 7 \cdot a + (-3) \cdot n) = (5 \cdot 13 + (-2) \cdot 15, 7 \cdot 13 + (-3) \cdot 15) = (35, 46)$. The client sends (35,46) to the cloud server.
(3) The cloud server computes two integers $(x', y') = (-21, 16)$ such that $35 \cdot x' + 46 \cdot y' = 1$ by the extended euclidean algorithm. The cloud server returns (x', y') to C.
(4) The result is correct because $35 \cdot (-21) + 46 \cdot 16 = 1$. Finally, the client recovers the final solution $x = (5 \cdot (-21) + 7 \cdot 16) \bmod 15 = 7$.

Clearly, $x = 7$ is indeed the inversion of 13 modulo 15. For any valid input, we give a rigorous theoretical analysis on the correctness of the proposed algorithm in the following theorem.

Theorem 1. *For any input $(a, n) \in \mathbb{Z}^2$ satisfying $\gcd(a, n) = 1$, the proposed secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MI}}(a, n)$ is correct according to Definition 1.*

Proof. First, we need to prove that a' and n' generated in sub-algorithm **ProbGen** are coprime. In other words, an honest server can indeed find two integers x' and y' such that $a'x' + n'y' = 1$ in sub-algorithm **Compute**. In fact, since

$$\begin{cases} a' = u_{11}a + u_{12}n \\ n' = u_{21}a + u_{22}n \end{cases} \Rightarrow \begin{cases} a = \frac{1}{\det(\mathbf{U})}(u_{22}a' - u_{12}n') \\ n = -\frac{1}{\det(\mathbf{U})}(u_{21}a' - u_{11}n'), \end{cases}$$

and $|\det(\mathbf{U})| = 1$, it can be easily deduced that each common divisor of a' and n' is a common divisor of a and n which implies $\gcd(a', n') = 1$.

Second, if an honest server returns two integers a' and n' satisfying $a'x' + n'y' = 1$, by substituting a' and n' with $u_{11}a + u_{12}n$ and $u_{21}a + u_{22}n$ respectively, we have $a(u_{11}x' + u_{21}y') + n(u_{12}x' + u_{22}y') = 1$. Hence $x = (u_{11}x' + u_{21}y') \bmod n$ is the inverse of a modulo n. □

4.3 Outsourcing Algorithm of $\sum_{i=1}^{k} a_i x_i \equiv 1 \mod n$

More generally, given $k+1$ nonzero integers $a_1, \ldots, a_k, n$ with $\gcd(a_1, \ldots, a_k, n) = 1$, the client tries to find a solution $x = (x_1, \ldots, x_k) \in \mathbb{Z}_n^k$ such that $\sum_{i=1}^{k} a_i x_i \equiv 1 \bmod n$. Similarly, the proposed algorithm is $\mathcal{SOC}_{\mathcal{MMI}}(a_1, \ldots, a_k, n)$ designed as follows:

1) **Prepro**: Given a security parameter λ, this preprocessing step generates a secret and random unimodular matrix

$$\mathbf{U} = \begin{pmatrix} u_{1,1} & \cdots & u_{1,(k+1)} \\ \vdots & \ddots & \vdots \\ u_{k+1,1} & \cdots & u_{k+1,k+1} \end{pmatrix} \in SK_{\mathcal{MMI}},$$

where $SK_{\mathcal{MMI}} =$

$$\left\{ \mathbf{U}_i \mid \mathbf{U}_i \in K_\lambda^{(k+1)\times(k+1)} \text{ and } |\det(\mathbf{U}_i)| = 1 \right\},$$

and $K_\lambda = \mathbb{Z} \cap (-2^\lambda, 2^\lambda)$ denotes the set of integers with bit length no larger than λ.

2) **ProbGen**: Using the secret matrix $\mathbf{U}$, C computes

$$\begin{cases} a'_1 = u_{1,1}a_1 + \cdots + u_{1,k}a_k + u_{1,k+1}n, \\ a'_2 = u_{2,1}a_1 + \cdots + u_{2,k}a_k + u_{2,k+1}n, \\ \quad \vdots \\ n' = u_{k+1,1}a_1 + \cdots + u_{k+1,k}a_k + u_{k+1,k+1}n. \end{cases} \tag{2}$$

and sends $(a'_1, \ldots, a'_k, n')$ to the cloud server S.

3) **Compute**: Receiving the blinded values $(a'_1, \ldots, a'_k, n')$, the cloud server S computes $k+1$ integers $x'_1, \ldots, x'_k, x'_{k+1}$ such that $\sum_{i=1}^{k} a'_i x'_i + n'x'_{k+1} = 1$ by invoking the extended Euclidean algorithm k times, then returns $(x'_1, \ldots, x'_k, x'_{k+1})$ to C.

4) **Verify**: Receiving the encoded output $(x'_1, \ldots, x'_k, x'_{k+1})$, the client C verifies whether the equation $\sum_{i=1}^{k} a'_i x'_i + n'x'_{k+1} = 1$ holds. If it does, C computes

$$x_i = \sum_{j=1}^{k+1} u_{ji} x'_j \mod n, \quad i = 1, \ldots, k, \tag{3}$$

as the solution. Otherwise, C outputs "$\perp$" and claims the misbehavior of S.

Remark 5. Similarly, in the preprocessing stage, C can generate the random unimodular matrix according to the parity of k. If $k+1$ is even, for $i = 1, \ldots, (k+1)/2$, C produces

$$\begin{pmatrix} u_{2i-1,2i-1} & u_{2i-1,2i} \\ u_{2i,2i-1} & u_{2i,2i} \end{pmatrix}, \tag{4}$$

using the method in Remark 2 and sets other $u_{i,j} = 0$. If $k+1$ is odd, for $i = 1, \ldots, (k-2)/2$, one first produces (4) using the method in Remark 2. Second, C generates a random 3×3 unimodular matrix (5) as follows:

$$\begin{pmatrix} u_{n-2,n-2} & u_{n-2,n-1} & u_{n-2,n} \\ u_{n-1,n-2} & u_{n-1,n-1} & u_{n-1,n} \\ u_{n,n-2} & u_{n,n-1} & u_{n,n} \end{pmatrix}. \tag{5}$$

1) C randomly generates three integers $u_{n-2,n-2}$, $u_{n-1,n-2}$, $u_{n,n-2} \in K_\lambda$ with $\gcd(u_{n-2,n-2}, u_{n-1,n-2}, u_{n,n-2}) = 1$.
2) Using the extended euclidean algorithm, C computes integers x_1, x_2, x_3, d_1, y_1 such that $u_{n-2,n-2}x_1 + u_{n-1,n-2}x_2 = d_1$, $d_1 y_1 + u_{n,n-2}x_3 = 1$.
3) C Returns $u_{n-2,n-1} = -x_2$, $u_{n-2,n} = -x_3 \cdot \frac{u_{n-2,n-2}}{d_1}$, $u_{n-1,n-1} = x_1$, $u_{n-1,n} = -x_3 \cdot \frac{u_{n-1,n-2}}{d_1}$, $u_{n,n-1} = 0$, $u_{n,n} = y_1$.

Finally, C sets other $u_{i,j} = 0$.

Remark 6. Since the generated unimodular matrix in Remark 5 is sparse and every unimodular submatrix of order 2 or 3 is independent, the **ProbGen** algorithm will be extremely efficient by parallel processing. However, the unimodular matrix transformation doesn't change the

greatest common divisor of any given integers (see Lemma 2). It may result in that this kind of unimodular matrix reveals the greatest common divisor of the two or three adjacent integers corresponding to the unimodular submatrix of order 2 or 3.

In order to prove the correctness of the secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MMI}}()$, we need to utilize the following significant property of unimodular matrix transformation.

Lemma 2. *The unimodular matrix transformation in the sub-algorithm* **ProbGen** *keeps the greatest common divisor of* $a'_1, \ldots, a'_k$ *invariant, namely,* $\gcd(a'_1, \ldots, a'_k, n') = \gcd(a_1, \ldots, a_k, n)$.

Proof. Let $d = \gcd(a_1, \ldots, a_k, n)$, $d' = \gcd(a'_1, \ldots, a'_k, n')$. Denote $\mathbf{a} = (a_1 \ \ldots \ a_k \ n)^{\mathrm{T}}$, $\mathbf{a}' = (a'_1 \ \ldots \ a'_k \ n')^{\mathrm{T}}$ as two $k+1$-dimensional column vectors.

On one hand, for any $1 \le i \le k$, by Equation (2), we have

$$a'_i = u_{i,1}a_1 + \cdots + u_{i,k}a_k + u_{i,k+1}n,$$
$$n' = u_{k+1,1}a_1 + \cdots + u_{k+1,k}a_k + u_{k+1,k+1}n.$$

Also, since $d \mid a_1$ and $d \mid n$, we have $d \mid a'_i$ and $d \mid n'$. Therefore, d is a common factor of $a'_1, \ldots, a'_k$ and n' which results in $d \le d'$.

On the other hand, by Lemma 1, there exists a unimodular matrix $\mathbf{V} \in \mathbf{Z}^{(k+1)\times(k+1)}$ such that $\mathbf{V} = \mathbf{U}^{-1}$. Let

$$\mathbf{V} = \begin{pmatrix} v_{1,1} & \cdots & v_{1,(k+1)} \\ \vdots & \ddots & \vdots \\ v_{k+1,1} & \cdots & v_{k+1,k+1} \end{pmatrix}.$$

From Equation (2), we have

$$\mathbf{a}' = \mathbf{U}\mathbf{a} \Leftrightarrow \mathbf{a} = \mathbf{V}\mathbf{a}'.$$

In other words, for any $1 \le i \le k$

$$a_i = v_{i,1}a'_1 + \cdots + v_{i,k}a'_k + v_{i,k+1}n',$$
$$n = v_{k+1,1}a'_1 + \cdots + v_{k+1,k}a'_k + v_{k+1,k+1}n'.$$

Since $d' \mid a'_i$ and $d' \mid n'$, d' is a common factor of $a_1, \ldots, a_k$ and n, Consequently, $d' \le d$. □

Based on the lemma mentioned above, we can deduce the correctness of our algorithm easily.

Theorem 2. *For any input* $(a_1, \ldots, a_k, n) \in (\mathbb{Z} \setminus \{0\})^{k+1}$ *with* $\gcd(a_1, \ldots, a_k, n) = 1$, *the proposed secure outsourcing computation algorithm* $\mathcal{SOC}_{\mathcal{MMI}}(a_1, \ldots, a_k, n)$ *is correct according to Definition 1.*

Proof. According to Definition 1, our proof is split into two steps.

(1) For any blind value $(a'_1, \ldots, a'_k, n')$ generated by the client C, there indeed exist $k+1$ integers $x'_1, \ldots, x'_k, x'_{k+1}$, such that $\sum_{i=1}^{k} a'_i x'_i + n'x'_{k+1} = 1$. In fact, by Lemma 2, $\gcd(a'_1, \ldots, a'_k, n') = \gcd(a_1, \ldots, a_k, n) = 1$ which implicates there exist $k+1$ integers $x'_1, \ldots, x'_k, x'_{k+1}$ such that $\sum_{i=1}^{k} a'_i x'_i + n'x'_{k+1} = 1$. Further, $x'_1, \ldots, x'_k, x'_{k+1}$ can be found by invoking the extended euclidean algorithm k times.

(2) For the integers $x'_1, \ldots, x'_k, x'_{k+1}$ received by the client C in sub-algorithm **Verify**, if $\sum_{i=1}^{k} a'_i x'_i + n'x'_{k+1} = 1$, then x_j calculated by Equation (3) satisfies $\sum_{i=1}^{k} a_i x_i \equiv 1 \mod n$. In fact

$$\begin{aligned} &\sum_{i=1}^{k} a'_i x'_i + n'x'_{k+1} = 1 \\ \Leftrightarrow\ &(\mathbf{a}')^{\mathrm{T}}\mathbf{x}' = 1 \Leftrightarrow (\mathbf{a}\mathbf{U})^{\mathrm{T}}\mathbf{x}' = 1 \\ \Leftrightarrow\ &(\mathbf{a}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}})\mathbf{x}' = 1 \Leftrightarrow \mathbf{a}^{\mathrm{T}}(\mathbf{U}^{\mathrm{T}}\mathbf{x}') = 1 \\ \Leftrightarrow\ &\sum_{i=1}^{k} a_i \left(\sum_{j=1}^{k+1} u_{j,i} x'_j \right) + n \sum_{j=1}^{k+1} u_{j,k+1} x'_j = 1 \\ \Rightarrow\ &\sum_{i=1}^{k} a_i \left(\sum_{j=1}^{k+1} u_{j,i} x'_j \right) \equiv 1 \mod n, \end{aligned}$$

where $\mathbf{a} = (a_1, \ldots, a_k, n)^{\mathrm{T}}, \mathbf{a}' = (a'_1, \ldots, a'_k, n)^{\mathrm{T}}, \mathbf{x}' = (x'_1, \ldots, x'_k, x'_{k+1})^{\mathrm{T}}$ denote $k+1$-dimensional column vectors.

From (1) and (2), we know the proposed secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MMI}}(a_1, \ldots, a_k, n)$ is correct. □

4.4 Security Analysis

In this section, we will analyze the security of the proposed algorithms under the MS model. For the sake of brevity and transparency, we only prove the verifiability and the input/output privacy of the algorithm $\mathcal{SOC}_{\mathcal{MI}}()$, and the proof for the algorithm $\mathcal{SOC}_{\mathcal{MMI}}()$ is similar.

4.4.1 Verifiability

According to the formalized definition in Section 2.3, we have the following result.

Theorem 3. *For any input* $(a, n) \in (\mathbb{Z} \setminus \{0\})^2$ *satisfying* $\gcd(a, n) = 1$, *the algorithm* $\mathcal{SOC}_{\mathcal{MI}}$ (a, n) *is verifiable according to Definition 2.*

Proof. Consider the experiment $\mathbf{Exp}_{\mathcal{A}}^{verif}[F, \lambda]$ defined in Section 2. In the query and response phase, $x_i = (a_i, n_i)$ is the ith input of sub-algorithm **ProGen**, $\tau_{x_i} = \mathbf{U}_i$ is the ith secret unimodular matrix randomly chosen from $SK_{\mathcal{MI}}$, $\sigma_{x_i} = (a'_i, n'_i)$ is the blinded value of x_i, and β_i is the ith output of sub-algorithm **Verify**.

Based on the information gathered in the query phase, the adversary $\mathcal{A}$ produces a challenge $x = (a, n)$. Using a random unimodular matrix $\tau_x = \mathbf{U}$ randomly chosen from $SK_{\mathcal{MI}}$, sub-algorithm **ProbGen** encodes x as $\sigma_x = (a', n')$. For any output $\hat{\sigma}(y) = (x', y')$ returned from the adversary $\mathcal{A}$, if $a'x' + n'y' = 1$, then, by the proof of correctness in Theorem 2, the sub-algorithm **Verify** obviously outputs $y = F(x)$. Else, the sub-algorithm **Verify** outputs $y = \perp$. Therefore

$$\Pr[\mathbf{Exp}_{\mathcal{A}}^{verif}[F, \lambda] = 1] = 0,$$

and the probability is regardless of the number of queries from $\mathcal{A}$. □

4.4.2 Input/Output Privacy

To prove the input/output privacy, we need the following two useful lemmas. The first lemma estimates the size of key space $SK_{\mathcal{MI}}$, and the second lemma estimates the number of input instances with the same ciphertext (a', n'). Due to limited space, we leave their strict proofs in Appendix A and Appendix B respectively, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2019.2937486.

Lemma 3. *Let $\#SK_{\mathcal{MI}}$ denote the size of the set SK, we have*

$$\#SK_{\mathcal{MI}} \geq \frac{48}{\pi^2}(2^\lambda - 1)^2.$$

Lemma 4. *For any given pair of integer $(a', n') \in \mathbb{Z}^2$ with $\gcd(a', n') = 1$ and $\max\{|a'|, |n'|\} \geq 2^{\lambda+1}$, define the sets*

$$\mathcal{P}_{\mathcal{I}} = \Big\{(a, n) \in \mathbb{Z}^2 \mid \exists \mathbf{U} \in SK_{\mathcal{MI}}\Big(\mathbf{U}(a, n)^{\mathrm{T}} = (a', n')^{\mathrm{T}}\Big)\Big\},$$
$$\mathcal{P}_{\mathcal{O}} = \big\{(x, y) \in \mathbb{Z}^2 \mid \exists \mathbf{U} \in SK_{\mathcal{MI}}((x, y) = (x', y')\mathbf{U})\big\},$$

where (x', y') satisfies $a'x' + n'y' = 1$. Then the size of $\mathcal{P}_{\mathcal{I}}$ and $\mathcal{P}_{\mathcal{O}}$ satisfies

$$\#\mathcal{P}_{\mathcal{I}} = \#\mathcal{P}_{\mathcal{O}} = \#SK \geq \frac{48}{\pi^2}(2^\lambda - 1)^2.$$

Employing the above two lemmas, we can easily argue the one-way security of the input/ouput information.

Theorem 4. *For any input $(a, n) \in (\mathbb{Z} \setminus \{0\})^2$ with $\gcd(a, n) = 1$, the algorithm $\mathcal{SOC}_{\mathcal{MI}}(a, n)$ is input-private and output-private according to Definitions 3 and 4 respectively.*

Proof. According to Definitions 3 and 4, we need to prove that the probability of the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[F, \lambda]$ (resp. $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[F, \lambda]$) outputting "1" is negligible.

In these two experiments, the computation task F denotes the modular inversion operation. In the query and response phase, the input $x_i = (a_i, n_i)$ satisfies $\gcd(a_i, n_i) = 1$, $\sigma_{x_i} = (a_i', n_i')$ is obtained by encrypting x_i with the ith secret unimodular matrix $\tau_{x_i} = \mathbf{U}_i$ randomly chosen from $SK_{\mathcal{MI}}$, $\sigma_{y_i} = (x_i', y_i')$ is the result evaluated by the adversary $\mathcal{A}$, and β_i denotes the ith output of sub-algorithm **Verify**.

In the challenge phase, $x^* = (a^*, n^*)$ with $\gcd(a^*, n^*) = 1$ is a challenge instance chosen from the domain of F. $\sigma_{x^*} = (a^{*\prime}, n^{*\prime})$ is obtained by encrypting x^* with a secret unimodular matrix $\tau_{x^*} = \mathbf{U}^*$ which is chosen uniformly from $SK_{\mathcal{MI}}$. For the input privacy, the adversary $\mathcal{A}$ is given σ_{x^*} and the information gathered in the query and response phase, and tries to recover x^*. For the output privacy, given σ_{x^*}, $\sigma_{y^*} = (x', y')$ satisfying $a^{*\prime}x' + n^{*\prime}y' = 1$ and the information gathered in the query and response phase, the adversary $\mathcal{A}$ is asked to recover $(a^*)^{-1} \bmod n^*$.

In the query and response phase of the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[F, \lambda]$, since, by Lemma 3, $\tau_{x_i} = \mathbf{U}_i$ is chosen independently and randomly from a sufficient large set $SK_{\mathcal{MI}}$, σ_{x_i} can be seen as a one-time pad encryption of x_i with a random key $\mathbf{U}_i$. Consequently, no information will be leaked to the adversary. For the input privacy, the adversary $\mathcal{A}$ knows $\sigma_{x^*} = (a^{*\prime}, n^{*\prime})$ and tries to recover x^*. By Lemma 4, the number of input instances with the same ciphertext σ_{x^*} is $\#\mathcal{P}_{\mathcal{I}}$. Since τ_{x^*} is uniformly and randomly chosen from $SK_{\mathcal{MI}}$, these instances are with the same probability. Therefore

$$\Big|\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{I}priv}[F, \lambda] = 1]\Big| = \frac{1}{\#\mathcal{P}_{\mathcal{I}}} \leq \frac{1}{\frac{48}{\pi^2}(2^\lambda - 1)^2},$$

which is a negligible function of λ. For the output privacy, the adversary $\mathcal{A}$ knows $(a^{*\prime}, n^{*\prime})$ and (x', y'), and tries to recover $(a^*)^{-1} \bmod n^*$, where $a^{*\prime}x' + n^{*\prime}y' = 1$. There are two possible ways. The adversary either recovers (a^*, n^*) from $(a^{*\prime}, n^{*\prime})$ or directly recovers $(a^*)^{-1} \bmod n^*$ from (x', y'). By Lemma 4, no matter in which way, the adversary's success probability satisfies

$$\Big|\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathrm{O}priv}[F, \lambda] = 1]\Big| \leq \frac{1}{\#\mathcal{P}_{\mathcal{I}}} = \frac{1}{\#\mathcal{P}_{\mathcal{O}}} \leq \frac{1}{\frac{48}{\pi^2}(2^\lambda - 1)^2},$$

which is also a negligible function of λ. □

4.5 Efficiency

On the client side, for any given constant k, the cost of blinding the input and verifying the output is substantially lower than the cost of performing the computation task from scratch locally.

Theorem 5. *Given an integer $k > 0$, for any input $(a_1, \ldots, a_k, n) \in \mathbb{Z}^{k+1}$ with $\gcd(a_1, \ldots, a_k, n) = 1$, the algorithm $\mathcal{SOC}_{\mathcal{MMI}}(a_1, \ldots, a_k, n)$ is $O(1/\log n)$-efficient according to Definition 5.*

Proof. During the execution of the algorithm $\mathcal{SOC}_{\mathcal{MMI}}(a_1, \ldots, a_k, n)$, sub-algorithm **ProbGen** needs $(k+1)^2$ multiplications, sub-algorithm **Verify** needs $k+1$ multiplications and $k(k+1)$ modular multiplications, where we omit other efficient operations such as modular addition. Hence the total cost on the client side is $O(k^2)$ multiplications. Since k is a constant and the complexity of multiplication operation is $O(\log^2 n)$, the complexity of $\mathcal{SOC}_{\mathcal{MMI}}(a_1, \ldots, a_k, n)$ is $O(\log^2 n)$. Without outsourcing, the client needs to execute the extended euclidean algorithm k times, which is an $O(\log^3 n)$ bit operation. Consequently, according to Definition 5, the proposed algorithm is $O(1/\log n)$-efficient. □

5 Performance Evaluation

In this section, we provide theoretical and experimental analysis of the proposed algorithms $\mathcal{SOC}_{\mathcal{MI}}()$ and $\mathcal{SOC}_{\mathcal{MMI}}()$ by (1) comparing our secure outsourcing computation algorithms with the corresponding algorithms without outsourcing and (2) comparing our secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MI}}()$ with the secure outsourcing computation algorithm presented in [26]. It is noteworthy that, since the sub-algorithm **Prepro** is a preprocessing step, the client-side time cost in our proposed algorithms consists of only two parts: the cost of sub-algorithm **ProbGen** and the cost of sub-algorithm **Verify**.

On the theoretical side, we provide an elaborate description on the computation overhead of algorithms in different cases. Let M, D, MM, MA and Mod denote the operation of

TABLE 1
The Computation Overhead of Our Algorithms in Different Phases

	ProbGen	Verify
$\mathcal{SOC}_{\mathcal{MI}}(a,n)$	4M	2M+2MM+1MA
$\mathcal{SOC}_{\mathcal{MMI}}(a_1,\ldots,a_k,n)$	$(k+1)^2$M	$(k+1)$M+$k(k+1)$MM+k^2MA

TABLE 2
Efficiency Comparison in Single-Variable Case

Algorithm	Computation Overhead
SMInv(a,n)	$2\log n$ M+$2\log n$ D+$2\log n$ MM+$2\log n$ MA
$\mathcal{SOC}_{\mathcal{MI}}(a,n)$	6M+2MM+1MA

TABLE 3
Efficiency Comparison in Multi-Variable Case

Algorithm	Computation Overhead
MMInv$(a_1,\ldots,a_k,n)$	$k\cdot 6\log n$ M+$k\cdot 2\log n$ D+$k(k-1)/2$ MM+1Mod
$\mathcal{SOC}_{\mathcal{MMI}}(a_1,\ldots,a_k,n)$	$(k+1)(k+2)$M+$k(k+1)$MM+k^2MA

multiplication, division, modular multiplication, modular addition and the modular operation, respectively. We omit other operations such as addition and substraction. Let "SMInv(a,n)", "MMInv$(a_1,\ldots,a_k,n)$" denote modular inversion without outsourcing in single-variable case and multi-variable case, respectively. The pseudocode is shown in Algorithms 2 and 3. Table 1 shows the computation overhead of algorithm $\mathcal{SOC}_{\mathcal{MI}}(a,n)$ and algorithm $\mathcal{SOC}_{\mathcal{MMI}}(a_1,\ldots,a_k,n)$ in different phases. Table 2 compares the computation overhead of the proposed algorithm with that of $\mathcal{SOC}_{\mathcal{MI}}(a,n)$ algorithm SMInv(a,n). The efficiency comparison in multi-variable case is given in Table 3. Table 4 shows the comparison results between our algorithm $\mathcal{SOC}_{\mathcal{MI}}(a,n)$ and the secure outsourcing computation algorithm presented in [26], where "UP" denotes the untrusted program model. The results in the tables indicate that, theoretically, the proposed algorithms have great efficiency advantages compared with the algorithms without outsourcing and the algorithm presented in [26].

Algorithm 2. SMInv(a,n)

Require: Integers a,n with $\gcd(a,n)=1$.
Ensure: An integers x $(1\le x<n)$ such that $ax\equiv 1 \bmod n$.
1: $x=0$, old_$x=1$, $y=1$, old_$y=0$, $r=n$, old_$r=a$
2: While $(r\neq 0)$
3: $q=\lfloor$ old_$r/r\rfloor$
4: $t=r, r=$ old_$r-q*t$, old_$r=t$
5: $u=x \mod n, x=($old_$x-q*u) \mod n$, old_$x=u$
6: Output old_x

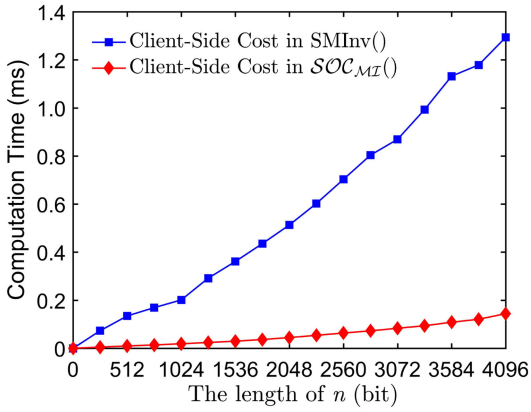


Fig. 2. Comparison of experimental results between algorithm $\mathcal{SOC}_{\mathcal{MI}}()$ and algorithm SMInv().

On the empirical side, we give the experimental evaluation of the proposed secure outsourcing computation algorithms. Our experiments are carried out on Ubuntu machine with 2.70 GHz Intel Pentium processor and 4 GB memory. We use C programming language with the GNU Multiple Precision Arithmetic (GMP) library. We set the modulus to be $256\cdot z$ bits, where z is an integer from 1 to 16. Fig. 2 compares the time cost on the client side in secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MI}}()$ with that in algorithm SMInv(). It can be observed that the cost on the client side is about 7-11 percent of that of the algorithm without outsourcing. Fig. 3 compares the time cost on the client side in secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MI}}()$ with that in the algorithm in [26]. The result shows that the cost on the client side in our algorithm is about 12-17 percent of that in the algorithm in [26]. For multi-variable scenarios, we compare the client-side cost between the secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MMI}}()$ and the algorithm MMInv(). In Fig. 4, the bit length of the modulus n is set as 1,024, while k varies from 5 to 100. In Fig. 5, k is set as 20 while the bit length of n varies from 256 to 4096. The above results demonstrate the great efficiency of our proposed algorithms.

Algorithm 3. MMInv$(a_1,\ldots,a_k,n)$

Require: Integers $a_1,\ldots,a_k,n$ with $\gcd(a_1,\ldots,a_k,n)=1$.
Ensure: k integers $x_1,\ldots,x_k$ such that $\sum_{i=1}^{k}a_ix_i\equiv 1 \bmod n$.
1: $(d_1,x_1,x_{k+1})\leftarrow$ `extended_gcd`(a_1,n)
2: For $i=2$ to k
3: $(d_i,x_i,y_{i-1})\leftarrow$ `extended_gcd`(a_i,d_{i-1})
4: For $j=1$ to $i-1$
5: $x_j=x_jy_{i-1} \mod n$
6: $x_k=x_k \mod n$
7: Output $x_1,\ldots,x_k$

TABLE 4
Comparison between $\mathcal{SOC}_{\mathcal{MI}}(a,n)$ and Proposed Algorithm in [26]

	ProbGen	Verify	Total overhead	Verifiability	UP	Modulus
$\mathcal{SOC}_{\mathcal{MI}}(a,n)$	4M	2M+2MM+1MA	6M+2MM+1MA	1	single	variable
algorithm in [26]	4MM	12MM+1MA	16MM+1MA	1	two	fixed

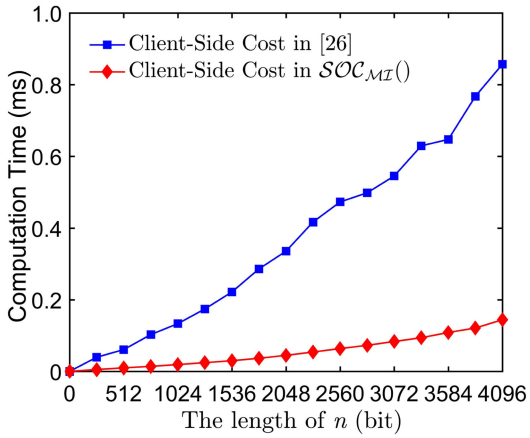


Fig. 3. Comparison of experimental results between algorithm $\mathcal{SOC}_{\mathcal{MI}}()$ and algorithm in [26].

6 Applications

In this section, we show two applications for the proposed outsourcing algorithms $\mathcal{SOC}_{\mathcal{MI}}()$ and $\mathcal{SOC}_{\mathcal{MMI}}()$.

6.1 Secure Outsourcing for the Secret Key Generation of RSA Algorithm

As one of the first practical public-key cryptosystems, RSA is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and the decryption key is secret. The theoretical security is based on the practical difficulty of factoring the product of two large primes. Concretely, RSA algorithm consists of three steps: *Key generation*, *Encryption* and *Decryption*.

1) *Key generation*:
   - Choose two distinct secure prime numbers p and q, which are at least 512 bits.
   - Compute $n = pq$, $\phi(n) = (p-1)(q-1)$.
   - Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$
   - Compute $d = e^{-1} \bmod \phi(n)$; i.e., d is the modular inverse of e (modulo $\phi(n)$).
   - The public key is $\{e, n\}$. The secret key is $\{d, n\}$
2) *Encryption*: For a plaintext m, compute $c = m^e \bmod n$
3) *Decryption*: For a ciphertext c, compute $m = c^d \bmod n$

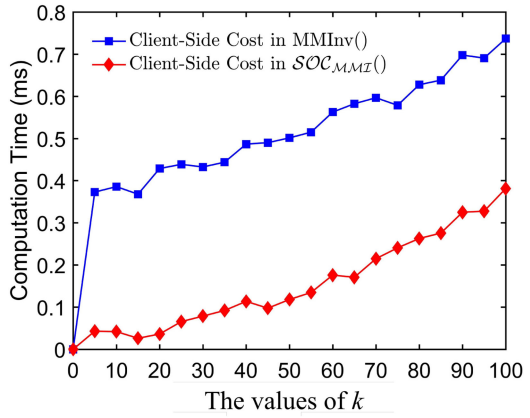


Fig. 4. Comparison of experimental results between algorithm $\mathcal{SOC}_{\mathcal{MMI}}()$ and algorithm MMInv() when the bit length of n is 1,024.

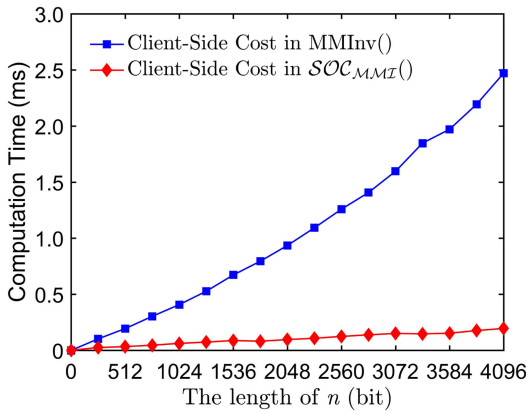


Fig. 5. Comparison of experimental results between algorithm $\mathcal{SOC}_{\mathcal{MMI}}()$ and algorithm MMInv() when $k = 20$.

Since p and q are large primes, $\phi(n) = (p-1)(q-1)$ is also a large number. Instead of computing the inverse by the user himself in step 1, we give an outsourcing implementation for *Key generation* of RSA by invoking the algorithm $\mathcal{SOC}_{\mathcal{MI}}()$ proposed in Section 4. The details are shown in Algorithm 4.

Algorithm 4. $\mathcal{SOC}_{\mathcal{RSAKG}}(\kappa)$

Require: A secure parameter κ.
Ensure: Public key $\{e, n\}$ and secret key $\{d, n\}$
1: Choose two distinct secure prime numbers p and q, which are at least κ bits.
2: Compute $n = pq$, $\phi(n) = (p-1)(q-1)$.
3: Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
4: Compute $d \leftarrow \mathcal{SOC}_{\mathcal{MI}}(e, \phi(n))$.
5: Return the public key $\{e, n\}$ and the secret key $\{d, n\}$.

6.2 Secure Outsourcing for the Chinese Reminder Theorem

Besides being significant in the number theory, the well-known Chinese reminder theorem is pervasive also in coding (e.g., redundant residue codes, Reed-Solomon codes) and cryptography (e.g., secret sharing) [9]. It is especially suited for computing with large integers, as it allows replacing one operation in which one knows a bound on the size of the result by several similar operations on small integers.

Given k pairwise coprime integers $m_1, \ldots, m_k$, the Chinese remainder theorem states that, for an integer n, if one knows the remainders of the divisions by $m_1 \ldots, m_k$, he can determine the unique remainder of the division by $\prod_{i=1}^{k} m_i$. More precisely,

Theorem 6 (Chinese Reminder Theorem [9]). *Assume $m_1, \ldots, m_k$ are positive integers satisfying $\gcd(m_i, m_j) = 1, \forall i \neq j$. Let $b_1, \ldots, b_k$ be arbitrary integers. Then the system of congruences*

$$\begin{cases} n \equiv b_1 \mod m_1 \\ \quad \vdots \\ n \equiv b_k \mod m_k, \end{cases} \tag{6}$$

has exactly one solution modulo the product $M = \prod_{i=1}^{k} m_i$, *namely,* $n = \sum_{i=1}^{k} M_i M_i' b_i \mod M$, *where, for* $1 \le i \le k$, $M_i = M/m_i, M_i M_i' \equiv 1 \mod m_i$.

In order to find the unique solution modulo M of linear congruences (6), we need to perform modular inversion k times. It is very time-consuming when m_i is large. Utilizing the proposed secure outsourcing computation algorithm $\mathcal{SOC}_{\mathcal{MMI}}()$, Algorithm 5 presents an outsourcing implement of solving the linear congruences (6).

The correctness of Algorithm 5 is described as follows: for any $i \in \{1, \ldots, k\}$, since $\sum_{i=1}^{k} M_i x_i \equiv 1 \mod M$ and $m_j \mid M_i \ (\forall j \neq i)$, we have;

$$M_i x_i \equiv \begin{cases} 1 & \mod m_i \\ 0 & \mod m_j. \end{cases}$$

Therefore, $n = \sum_{i=1}^{k} M_i x_i b_i \mod M \equiv b_i \mod m_i$.

Algorithm 5. $\mathcal{SOC}_{\mathcal{CRT}}(b_1, \ldots, b_k, , m_1, \ldots, m_k)$

Require: Integers $m_1, \ldots, m_k$ with $\gcd(m_i, m_j) = 1, \forall i \neq j$ and integers $b_1, \ldots, b_k$.
Ensure: Integers $n \ (1 \le n \le M)$ such that $n \equiv b_i \mod m_i$, where $1 \le i \le n$, $M = \prod_{i=1}^{k} m_i$.
1: Compute $M = \prod_{i=1}^{k} m_i$.
2: For $i = 1$ to k
3: $M_i = M/m_i$.
4: $(x_1, \ldots, x_k) \leftarrow \mathcal{SOC}_{\mathcal{MMI}}(M_1, \ldots, M_k, M)$.
5: Output $n = \sum_{i=1}^{k} M_i x_i b_i \mod M$.

7 Conclusion

This paper investigates how to securely outsource the modular inversion for arbitrary and variable modulus. By taking advantages of the unimodular matrix transformation, we present a novel secure outsourcing computation algorithm for modular inversion with high-efficiency and robust cheating resistance. Compared with prior art that demands two non-colluding servers, our algorithms remove the strong assumption and only require one single cloud server. To demonstrate the wide applicability of our algorithms, we implement the secure outsourcing for the key generation of RSA algorithm and the Chinese Reminder Theorem, and show the results with efficiency and effectiveness.

Acknowledgments

This research is supported by National Natural Science Foundation of China (61702294, 61572267), Natural Science Foundation of Shandong Province (ZR2016FQ02), National Development Foundation of Cryptography (MMJJ20170126,MMJJ20170118), the Open Research Project (2016-MS-23, 2019-MS-03) of State Key Laboratory of Information Security in Institute of Information Engineering, Chinese Academy of Sciences, Key Research and Development Project of Shandong Province, Applied Basic Research Project of Qingdao City (17-1-1-10-jch).

References

[1] J. Jonsson, K. Moriarty, B. Kaliski, and A. Rusch, PKCS# 1: RSA Cryptography Specifications Version 2.2. RFC8017, Nov. 2016. [Online]. Available: https://tools.ietf.org/html/rfc8017
[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
[3] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. 32nd Annu. Cryptology Conf. Advances Cryptology*, 2012, pp. 868–886.
[4] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," in *Proc. IEEE 52nd Annu. Symp. Found. Comput. Sci.*, 2011, pp. 97–106.
[5] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. S. Wong, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 1, pp. 69–78, Jan. 2015.
[6] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2386–2396, Sep. 2014.
[7] K.-M. Chung, Y. Kalai, and S. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *Proc. 30th Annu. Conf. Advances Cryptology*, 2010, pp. 483–501.
[8] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Apr. 2010, pp. 27–33.
[9] C. Ding, D. Pei, and A. Salomaa, *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. Singapore: World Scientific, 1996.
[10] Y. Ding, Z. Xu, J. Ye, and K.-K. R. Choo, "Secure outsourcing of modular exponentiations under single untrusted programme model," *J. Comput. Syst. Sci.*, vol. 90, pp. 1–13, 2017.
[11] L. Ducas and D. Stehlé, "Sanitization of FHE ciphertexts," in *Proc. 35th Annu. Int. Conf. Advances Cryptology*, 2016, pp. 294–310.
[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proc. 4th Annu. Int. Conf. Advances Cryptology*, 1985, pp. 10–18.
[13] K. Elkhiyaoui, M. Onen, M. Azraoui, and R. Molva, "Efficient techniques for publicly verifiable delegation of computation," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 119–128.
[14] D. Fiore and R. Gennaro, "Publicly verifiable delegation of large polynomials and matrix computations, with applications," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 501–512.
[15] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. 30th Annu. Cryptology Conf. Advances Cryptology*, 2010, pp. 465–482.
[16] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. ACM Symp. Theory Comput.*, 2009, pp. 169–178.
[17] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. 33rd Annu. Cryptology Conf. Advances Cryptology*, 2013, pp. 75–92.
[18] S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," in *Proc. 2nd Int. Conf. Theory Cryptography*, 2005, pp. 264–282.
[19] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp. 129–140, Feb. 2013.
[20] X. Lei, X. Liao, T. Huang, and H. Li, "Cloud computing service: The case of large matrix determinant computation," *IEEE Trans. Serv. Comput.*, vol. 8, no. 5, pp. 688–700, Sep. 2015.
[21] P. M. Mell and T. Grance, "The NIST definition of cloud computing," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. Sp 800-145, 2011.
[22] M. Newman, *Integral Matrices*. New York, NY, USA: Academic Press, 1972.
[23] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Comput.*, vol. 16, no. 1, pp. 69–73, Jan. 2012.
[24] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Proc. Conf. Theory Appl. Cryptology*, 1990, pp. 239–252.
[25] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.
[26] Q. Su, J. Yu, C. Tian, H. Zhang, and R. Hao, "How to securely outsource the inversion modulo a large composite number," *J. Syst. Softw.*, vol. 129, pp. 26–34, 2017.

[27] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 820–828.
[28] Y. Wang, Q. Wu, D. S. Wong, B. Qin, S. S. M. Chow, Z. Liu, and X. Tan, "Securely outsourcing exponentiations with single untrusted program for cloud storage," in *Proc. 19th Eur. Symp. Res. Comput. Secur.*, 2014, pp. 326–343.
[29] J. Yu, K. Ren, and C. Wang, "Enabling cloud storage auditing with verifiable outsourcing of key updates," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 6, pp. 1362–1375, Jun. 2016.
[30] F. Zhang, X. Ma, and S. Liu, "Efficient computation outsourcing for inverting a class of homomorphic functions," *Inf. Sci.*, vol. 286, pp. 19–28, 2014.
[31] L. Zhang, "Editorial: Big services era: Global trends of cloud computing and big data," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 467–468, Oct.–Dec. 2012.
[32] P. Zhao, J. Yu, H. Zhang, Z. Qin, and C. Wang, "How to securely outsource finding the min-cut of undirected edge-weighted graphs," *IEEE Trans. Inf. Forensics Secur.*, 2019, doi: 10.1109/TIFS.2019.2922277.
[33] K. Zhou, M. H. Afifi, and J. Ren, "ExpSOS: Secure and verifiable outsourcing of exponentiation operations for mobile cloud computing," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 11, pp. 2518–2531, Nov. 2017.
[34] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Comput. Syst.*, vol. 28, no. 3, pp. 583–592, 2012.

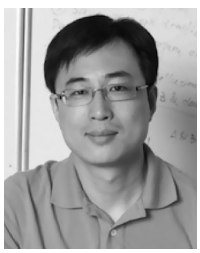
Chengliang Tian received the BS and MS degrees in mathematics from Northwest University, Xi'an, China, in 2006 and 2009, respectively, and the PhD degree in information security from Shandong University, Ji'nan, China, in 2013. He held a post-doctoral position with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing. He is currently with the College of Computer Science and Technology, Qingdao University, as an assistant professor. His research interests include lattice-based cryptography and cloud computing security.

Jia Yu received the BS and MS degrees from the School of Computer Science and Technology, Shandong University, in 2000 and 2003, respectively, and the PhD degree from the Institute of Network Security, Shandong University, in 2006. He was a visiting professor with the Department of Computer Science and Engineering, State University of New York at Buffalo, from 2013 to 2014. He is currently a professor with the College of Computer Science and Technology, Qingdao University. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.

Hanlin Zhang received the BS degree in software engineering from Qingdao University, in 2010, and the MS degree in applied information technology and PhD degree in information technology from Towson University, Maryland, in 2011 and 2016, respectively. He is currently working with Qingdao University as an assistant professor in the College of Computer Science and Technology. His research interests include cloud computing security, blockchain technology, and IoT security.

Haiyang Xue received the PhD degree in information security from the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, in 2015. He is currently a researcher assistant with the Institute of Information Engineering. His research interests include post-quantum cryptography and authenticated key exchange.

Cong Wang received the BE and ME degrees in electrical and computer engineering from Wuhan University, in 2004 and 2007, respectively, and the PhD degree in electrical and computer engineering from the Illinois Institute of Technology, in 2012. He has worked with Palo Alto Research Center in the summer of 2011. He has been an assistant professor with the Department of Computer Science, City University of Hong Kong, since 2012. His research interests include the areas of cloud computing and security, with current focus on secure data services in cloud computing, and secure computation outsourcing. He is a senior member of the IEEE and a member of the ACM.



Kui Ren received the PhD degree from Worcester Polytechnic Institute. He is currently a professor with the Institute of Cyberspace Research, Zhejiang University, China. His current research interests span cloud and outsourcing security, wireless and wearable systems security, and mobile sensing and crowdsourcing. His research has been supported by NSF, DoE, AFRL, MSR, and Amazon. He has published extensively in peer reviewed journals and conferences and received several Best Paper Awards, including at ICDCS 2017, IWQoS 2017, and ICNP 2011. He currently serves as an associate editor of the *IEEE Transactions on Dependable and Secure Computing*, the *IEEE Transactions on Service Computing*, the *IEEE Transactions on Mobile Computing*, the *IEEE Wireless Communications*, the *IEEE Internet of Things Journal*, and as an editor for SpingerBriefs on Cyber Security Systems and Networks. He is a fellow of the IEEE, a distinguished lecturer of IEEE, a member of the ACM, and a past board member of the Internet Privacy Task Force, State of Illinois.