Efficient Privacy-Preserving Computation Based on Additive Secret Sharing

Lizhi Xiong, Wenhao Zhou, Zhihua Xia, *Member,IEEE*, Qi Gu, and Jian Weng, *Member,IEEE*

Abstract—The emergence of cloud computing provides a new computing paradigm for users—massive and complex computing tasks can be outsourced to cloud servers. However, the privacy issues also follow. Fully homomorphic encryption shows great potential in privacy-preserving computation, yet it is not ready for practice. At present, secure multiparty computation (MPC) remains mainly approach to deal with sensitive data. In this paper, following the secret sharing based MPC paradigm, we propose a secure 2-party computation scheme, in which cloud servers can securely evaluate functions with high efficiency. We first propose the multiplicative secret sharing (MSS) based on typical additive secret sharing (ASS). Then, we design protocols to switch shared secret between MSS and ASS, based on which a series of protocols for comparison and nearly all of the elementary functions are proposed. We prove that all the proposed protocols are *Universally Composable* secure in the honest-but-curious model. Finally, we will show the remarkable progress of our protocols on both communication efficiency and functionality completeness.

Index Terms—Privacy-Preserving Computation, Secure Multiparty Computation, Additive Secret Sharing, Cloud Computing.

1 Introduction

CLOUD computing as a new computing and data processing paradigm has significantly influenced the way we live. By aggregating the virtualized and interconnected computers and other IT infrastructures, cloud computing provides high-quality Internet service with characteristics of on-demand, broad network access, resource pooling, rapid elasticity, and measured service for users [1], [2]. More concretely, for individuals, cloud computing reduces the threshold of accessing computing resources, eliminates obstacles brought by infrastructures, and makes possible complex tasks such as deep learning [3]. For groups and companies, cloud computing gives more flexibility in allocating resources, make them concentrate more on their business, and cuts expenses. In the past decade, the needs of economy constantly promoted the development of cloud computing, which in turn played an important role in integrating social resources and improving social efficiency.

However, behind such a perfect cooperation model, there are some privacy concerns that cannot be ignored [4]. In the cloud computing scenario, users inevitably have to hand over data which contain sensitive information to the cloud server. This means that cloud servers can access our private data whenever and wherever they like. And as it turns out, the rise of more and more data breach issues, such as the massive data leakage of Facebook impacting 267 million users in 2017, attracts more attention from the public and drives the implementation of relevant laws [5]. For instance, General Data Privacy Regulation (GDPR), the most comprehensive and widely applied privacy regulation, implemented in the European Union in 2018, is geared towards protecting user privacy and against infringement of data breach. Nevertheless, merely by virtue of socio-legal, it is insufficient to provide truly convincing cloud computing security. Therefore, recent researches on privacy-preserving computation offered a cryptography way to achieve this and it was shown on feasible in theoretical.

The most direct idea is to use Homomorphic Encryption (HE) which happens to coincide with the requirement of cloud computing security: allows performing a series of certain functions on ciphertext with reserving the ability to be decrypted. For example, Paillier [6] is one of the partially homomorphic encryption (PHE) with additive homomorphism. Considering two plaintexts m_1 and m_2, cloud servers can compute $E(m_1 + m_2)$ with $E(m_1)$ and $E(m_2)$ without knowing the private key under Paillier encryption scheme. Yet PHE like Paillier allows only one type of operation on ciphertext and thus is not competent for some complex cloud computing tasks. Somewhat homomorphic encryption (SHE) improves the number of types of operations comparing to PHE, such as SYY [7], BGN [8], IP [9]. These schemes introduce noise to achieve security, resulting in an increase of noise on ciphertext with every evaluation. When the accumulation of noise on ciphertext exceeds a threshold, the ciphertext will lose the ability to be decrypted, which against the definition of HE. Therefore, SHE limits the number of operations on ciphertext and the possibility of fully computation offload. In 2009, a pioneering scheme [10] proposed by Gentry introduced bootstrapping technique to eliminate the noise of ciphertext and achieved the first plausible fully homomorphic encryption (FHE) which can evaluate any operation on ciphertext for any times. In a nutshell, bootstrapping means homomorphically decrypt the ciphertext to get a clean ciphertext that contains no noise and is decryptable. However, bootstrapping is costly and takes tens to hundreds of minutes for every "clean" [11], [12]. Hence, HE remains a long way to go from theory to practice.

- *L. Xiong (email: xionglz@nuist.edu.cn), W. Zhou (email: apurance@gmail.com), Z. Xia (corresponding author, email: xia_zhihua@163.com) and Q. Gu are with the School of Computer and Software, Nanjing University of Information science and Technology, Nanjing, China.*
- *J. Weng is with Jinan University, Guangzhou 510632, China.*

Another way is to use secure multiparty computation (MPC) which allows a group of parties who don't trust each other jointly compute a certain function $\mathcal{F}(x_1, x_2, \cdots, x_n)$ without revealing their private input x_i. MPC first introduced by Yao in his seminal Two Millionaires Problem [13] has been developed for four decades and a lot of valuable work for it has emerged. In 1986, Yao proposed a well-known secure 2-party computation (2PC) technic called Garbled Circuits (GC) [14] where Alice first generates a garbled Boolean circuit $\mathcal{C}$ which represents the function that Alice and Bob want to perform with their private input, then Bob receives the circuit $\mathcal{C}$ and evaluate it by interacting with Alice. Later, Ben-Or *et al.* proposed BGW protocol [15] which allow parties to evaluate arithmetic circuit consisting of addition and multiplication. In the privacy-preserving computation field, the GMW (aka. secret sharing based MPC) paradigm [16] is usually applied, where users can use secret sharing technic [17] to share their private data to servers and then servers evaluate a circuit which represents the function users want to perform. Although Boolean circuit or arithmetic circuit that contains only addition and multiplication is Turing-complete in theoretical, constructing a specific circuit for nonlinear functions to improve efficiency is still a meaningful work. Many efforts to design secure protocols for nonlinear functions such as comparison, division, exponent, etc. have been made [18], [19], [20]. However, these schemes either use Taylor series to fit or directly construct Boolean circuit, which not only lead to accuracy loss, but also to low efficiency. The purpose of this paper is to design secure protocols that generate arithmetic circuit for nonlinear functions without Taylor series or bit-level operation as with typical addition and multiplication protocols.

Contribution. In this paper, we propose a secure 2-party computation scheme, which can be applied in privacy-preserving computation under the context of cloud computing. The main contributions can be described as follows:

- Inspired by additive secret sharing (ASS) in the context of MPC, we first introduce the multiplicative secret sharing (MSS) into secret sharing based MPC. The shares shared through ASS are with additive homomorphism, while shares shared through MSS are with multiplicative homomorphism. It is almost impossible to give share fully homomorphism, yet it is possible to give share proper homomorphism at the right time.
- We design two basic 2PC protocols for switching shared secret between MSS and ASS, including `SecMulRes` that converts shared secret over MSS to over ASS, and `SecAddRes` that is the inverse of `SecMulRes`.
- Based on the above two basic protocols, we design a series of 2PC protocols for comparison, exponentiation, logarithm, power, division, product and trigonometric functions, where both the input and output are shared secrets, which follows the GMW paradigm. Finally, we nearly achieve all the elementary functions.
- All of these protocols contain only arithmetic operation and no bit-level operation or any fitting. This means that our scheme is not limited to specific floating point precision and avoids the loss of accuracy in theoretical. Moreover, all the protocols are with constant round complexities and low communications.
- We prove that all the proposed protocols are Universally Composable secure against the honest-but-curious model.

Oranization. The organization of this paper is as follows: We briefly introduce the related work of HE and MPC in Section 2. The system model and preliminaries are in Section 3. We formally describe secure protocols for linear and multiplication in Section 4 and our proposed protocols can be found in Section 5. The complexity of protocols is summarized in Section 6. Finally, we give conclusions about our work in Section 7.

2 Related Work

In cloud computing, privacy-preserving computation means that cloud servers can evaluate some functions on "encrypted" user data and return "encrypted" results to users, where the term "encrypted" indicates that the privacy of data is preserved. There are mainly two methodologies which could achieve it: homomorphic encryption and secure multiparty computation.

Homomorphic Encryption. Since the term "homomorphic encryption" first proposed by Rivest *et al.* [21] in 1978, the purist of FHE has never stopped. As the first attempts of HE, PHE can be divided into two categories: additive HE and multiplicative HE. RSA is the first public key cryptography proposed by Rivest *et al.* [22] in 1978, later Rivest *et al.* [21] showed the multiplicative homomorphism of RSA and presented the term HE in the same year. Elgamal [23] proposed another widely used multiplicative HE scheme which has higher security than RSA. The additive HE schemes represented by Paillier [6] mainly include GM [24], Benaloh [25], OU [26], NS [27], Galbraith [28] and KTX [29]. Since PHE only supports either addition or multiplication, in some privacy-preserving schemes using PHE [30], users usually have to undertake part of or even most of the computing tasks, and cloud servers fulfill more of an auxiliary role. Before Gentry's seminal work in 2009, the development of SHE is very limited, where BGN proposed by Boneh *et al.* [8] is one of the famous SHE schemes, which allow one multiplication and unlimited addition to being performed on ciphertext. In 2009, Gentry [10] proposed the first FHE scheme in his Ph.D. thesis. Gentry first showed a SHE scheme based on ideal lattices, and then introduced bootstrapping technique (also called decrypt homomorphically) to reduce noise of ciphertext. If a SHE scheme can evaluate its decryption algorithm, then this scheme is bootstrappable and can be transformed to FHE scheme through bootstrapping technique. After Gentry's work, many SHE schemes have been proposed [31], [32], [33], [34], and bootstrapping also be used to realize FHE. Due to the inefficiency of bootstrapping, Brakerski *et al.* [35] proposed a leveled-FHE scheme that can evaluate functions with predefined circuit depth without using bootstrapping. However, the scheme is hardly considered as real FHE, it has better performance than other FHE schemes in some scenario though. At present, FHE is gradually applied in practice and has great efficiency improvement potential.

Secure Multiparty Computation. After Yao presented the Two Millionaires Problem [13], he proposed GC [14] which is one of the first MPC protocols. Later, Beaver *et al.* [36] proposed BMR protocols which extend GC from 2PC to MPC by separating Alice's task. Since GC is high costly, many optimization techniques have been proposed to improve efficiency and could be divided into three categories: optimization in GC generation [37], [38], optimization in GC evaluation [36], [39], [40], optimization in protocol level [41]. In GC-type schemes, the parties are the private data owner, in other words, they know what input data meaning for. Therefore, Goldreich, Micali, and Wigderson (GMW) [16] presented another MPC paradigm in 1987, where every party first shares his private input (into n shares); then n parties run MPC protocols on shares; finally, n parties recover results from output shares. Specifically, GMW using ASS over $\mathbb{Z}_2$ to share each bit of input, where oblivious transfer [42] is used to evaluate AND gate and XOR gate can be evaluated locally. The GMW paradigm expands the source of MPC input, that is, allowing private input outside the parties such as users. Follow this paradigm, Ben-Or *et al.* [15] proposed BGW protocol which allows parties evaluate arithmetic circuit based on Shamir secret sharing [17], where both addition and multiplication can be evaluated locally. Since multiplication will double the order of polynomial and thus the secret cannot be recovered from shares, BGW introduced degree reduction via resharing shares. However, this method is not only inefficient, but also limits the threshold of Shamir secret sharing requiring that $2t < n$. To solve this problem, Beaver creatively introduced Beaver triple [43] generated in offline phase to significantly improve efficiency of evaluating multiplication. In 2006, Damgård *et al.* [44] presented bit-decomposition protocol to convert shared secret over arithmetic sharing to over bitwise sharing. This technique allows us to combine the advantages of Boolean circuit and arithmetic circuit. Since then many MPC schemes based on ASS and bit-decomposition technique have been proposed [19], [41], [45]. In addition, there are many works that are dedicated to implementing MPC protocols and even provide secure language to hide details of MPC protocols [41], [45], [46], [47]. Recently, with the rapid development of machine learning, many schemes that use MPC technic to support privacy-preserving for the task have been proposed [20], [48], [49]. Most of them optimized MPC they used according to the characteristics of machine learning.

3 System Model and Preliminaries

In this section, we describe the GWM paradigm which is the basis of this paper, the threat model we use, definition of security, and the secret sharing technic.

3.1 GMW Paradigm

GMW [16] introduced secret sharing in MPC to encrypt private input and become a new MPC paradigm called GMW paradigm. In this paper, we follow the GMW paradigm, and which can be divided into three stages:

1) *All private inputs are shared by their owners*. On the one hand, secret sharing encrypts private data, on the other hand, it decouples the secret owner and the sharing holder, i.e., the private data owner and the party. Therefore, secret sharing expands input sources, which means that non-parties, such as users, can provide data to parties, while in MPC scenarios, the private data owner is equal to the party. In this paper, the ASS is set to the default secret sharing technic. Since ASS is an (n, n) threshold scheme, sharing private input entails that every party holds a share of every input.
2) *The parties evaluate the circuit on shares.* In MPC, parties will generate corresponding circuit according to specific tasks before the execution of protocol, while in the context of cloud computing, since the party (i.e., the server) is the outsourcer, the users need to send the circuit to the parties. Indeed, the users can also send the function to be executed and the circuit will be generated by the parties. Regardless of the data source, the function and its details are clear and public to parties. This means that users can only keep data private to the server but not function private. In this paper, we specify that there are two parties P_1 and P_2 (i.e., two servers) due to the fact that the fewer servers involved in computing, the more secure it is for users.
3) *The final result is recovered by output shares.* After the evaluating stage, parties will output shares of the result. In cloud computing, these shares will then be sent back to users. The principal feature of GMW paradigm is that input and output, including intermediate results, are in the form of share, which is similar to the ciphertext form in homomorphic encryption.

In GMW paradigm, the main effort (include this paper) is to design efficient protocols to generate circuit for functions. Moreover, since the users are not involved in computation, the number of users and communication between users and parties are not considered in protocol design.

3.2 Threat and Security Model

Similar to most secret sharing based MPC schemes [16], [19], [45], the *honest-but-curious* model (aka. semi-honest model) is used in this paper, where an honest-but-curious adversary could corrupt parties before the execution of protocol, in other words, each party could be seen as honest-but-curious adversary. An honest-but-curious adversary will abide by the protocol honestly but will analyze all the data he has as much as possible. In addition, we assume that two parties will not collude with each other, which means the one adversary cannot corrupt two parties at the same time.

The real-ideal paradigm firstly presented in [50] is widely used to prove security of stand-alone MPC protocol. In the ideal world, there is a trusted third party $\mathcal{T}$ who will receive private input x_i sent by every party to compute $\mathcal{F}(x_1, x_2, \cdots, x_n)$ and send results to parties. The trusted third party $\mathcal{T}$ is also called ideal functionality $\mathcal{F}$, in other words, with the help of $\mathcal{T}$, parties securely compute the function $\mathcal{F}$. In the real world, there is no such $\mathcal{T}$, parties have to interact with each other to compute $\mathcal{F}$ according to protocol. Roughly, a protocol π is secure or securely realizes ideal functionality $\mathcal{F}$ if the distributions of input and output in the ideal world and in the real world are indistinguishable.

However, the real-ideal paradigm cannot guarantee the composability of secure protocol, which means that a complex protocol composed of secure sub-protocols is still a stand-alone protocol and should be proved from scratch. The Universally Composable (UC) Security framework proposed by Canetti [51] introduced the environment on the basis of the real-ideal paradigm to give secure protocol with the composability. A sub-protocol can be considered as the ideal functionality in proving security, if it is UC-secure. In this paper, we will prove the security of our protocols under UC framework and the following security definition will be used.

Definition 1. A protocol π is UC-secure or UC-securely realizes the ideal functionality $\mathcal{F}$ if for any real world adversary $\mathcal{A}$ there exists a probabilistic polynomial-time simulator $\mathcal{S}$ that can simulate an ideal world view such that is indistinguishable from real world view.

To prove that a protocol is UC-secure, it suffices to show that the incoming view consisting of the message sent by other parties and output is simulatable for any parties (adversary). In addition, the following lemmas will be used.

Lemma 1. The element $x + r$ is uniformly distributed and independent from x for any element $x \in \mathbb{F}$ if the element $r \in \mathbb{F}$ is also uniformly distributed and independent from x [19].

Proof: Readers can refer to the proof in [19]. □

Lemma 2. The nonzero element $x \cdot r$ is uniformly distributed and independent from x for any element $x \in \mathbb{F}$ if the element $r \in \mathbb{F}$ is also uniformly distributed and independent from x.

Proof: If the nonzero element $r \in \mathbb{F}$ is uniformly distributed and independent from all x then so is $x \cdot r$, since $f_r(x) := x \cdot r$ is a bijective mapping for $\mathbb{F}$. □

3.3 Secret Sharing

Secret sharing introduced by Shamir [17] is aimed at addressing a problem in which the secret data divided into n pieces can be restored with over a certain number of pieces, whereas cannot with less of those. Specifically, in [17], Shamir presented a (k, n) threshold scheme such that: 1) the secret data s which is named *secret* will be securely divided into n pieces s_i which are named *shares*; 2) any k shares are able to reveal complete information about the secret; 3) any $k-1$ or less shares reveals no useful information about the secret. One wants to share a secret s, he first securely divides secret into n shares s_i via secret sharing, then sends them to n parties respectively. Finally, any k parties can restore the secret s together, whereas any $k-1$ or less parties cannot.

Additive secret sharing (ASS) is defined over a field $\mathbb{F}$. In ASS, a secret $x \in \mathbb{F}$ will be randomly divided into n shares $[x_1, x_2, \cdots, x_n]$ such that $x_1 + x_2 + \cdots + x_n = x$. Specially, ASS defines an (n, n) threshold scheme, since the restoration of the secret requires all of the shares. The secret sharing used in GMW protocol [16] can be regarded as ASS defined over a finite field $\mathbb{Z}_2$ and is suitable for Boolean circuit. In [19], [20], [45], [48], [49], ASS defined over a finite ring $\mathbb{Z}_{2^l}$ or a finite filed $\mathbb{F}$ is used and most of the secure operation is on arithmetic circuit. The defined field for ASS determines the element form of shares and type of circuit on which parties want to perform. In this paper, the ASS defined over real field $\mathbb{R}$ is considered, which is adequate for comprehensive operations on arithmetic circuit.

Inspired by ASS, we introduce the multiplicative secret sharing (MSS) defined over real field $\mathbb{R}$ for the needs of protocol design. Specifically, in MSS, a secret $u \in \mathbb{R}$ will be randomly divided into n shares $[u_1, u_2, \cdots, u_n]$ such that $u_1 \times u_2 \times \cdots \times u_n = u$. Similar to ASS, MSS defines an (n, n) threshold scheme. Note that it is not appropriate to use MSS defined over some fields such as $\mathbb{Z}_2$ due to the fact that once a share is 0, the party could know the secret is 0.

The original intention of secret sharing is to provide access control for sensitive data. In this case, the threshold t should make a tradeoff between security and reliability, where a higher t means more security and less reliability, while a lower t is the opposite. As an (n, n) threshold scheme, ASS and MSS obviously sacrifice reliability and thus is not suitable in this scenario. However, they provide MPC with capabilities of data encryption and data distribution.

For the sake of brevity, we use $[\![x]\!]$ to denote all of shares $[x_1, x_2, \cdots, x_n]$ generated by the secret x through ASS, and use $\langle u \rangle$ to denote all of shares $[u_1 \times u_2 \times \cdots \times u_n]$ generated by the secret u through MSS. More precisely, the notation $[\![x]\!]$ or $\langle u \rangle$ mean share x_i or u_i for party $\mathcal{P}_i$, in other words, it represents an ensemble of individuals.

4 Linear and Multiplication Protocols

Secure protocols for addition can be implemented directly based on additive homomorphism of ASS, meanwhile, secure multiplication protocol can be constructed efficiently with Beaver triple [43]. The implementation of these protocols is not the contribution of this paper, but we still devote a section here to formally describe them for completeness.

As illustrated in Algorithm 1, secure linear protocol which further considers the operation of multiplication by public number takes n shared secrets $[\![x^1]\!], [\![x^2]\!], \cdots, [\![x^n]\!]$ with their public coefficient $a^1, a^2, \cdots, a^n$ and a public bias b as input, and return the shared secret $[\![\sum_{j=1}^{n} a^j \cdot x^j + b]\!]$. For clarity, superscript here is used to distinguish different secrets or public numbers, whereas subscript is used to represent the share of corresponding party. During the execution of secure linear protocol, $\mathcal{P}_1$ and $\mathcal{P}_2$ are only required to perform same linear operation on their own shares asynchronously.

Theorem 1. Algorithm 1 is correct and UC-secure in the honest-but-curios model.

Proof: For correctness, there are

$$\begin{aligned} f &= f_1 + f_2 = \sum_{j=1}^{n} a^j \cdot x_1^j + \sum_{j=1}^{n} a^j \cdot x_2^j + b \\ &= \sum_{j=1}^{n} a^j \cdot (x_1^j + x_2^j) + b = \sum_{j=1}^{n} a^j \cdot x^j + b \end{aligned}$$

For security, note that Algorithm 1 contains only local operations and no interaction with each party. Therefore, the protocol is perfectly simulatable and is UC-secure in the honest-but-curios model. □

```
Algorithm 1 Secure Linear Protocol $[\![f]\!] \leftarrow SecLinear([\![x^1]\!], [\![x^2]\!], \cdots, [\![x^n]\!], a^1, a^2, \cdots, a^n, b)$
Input: Shared secrets $[\![x^j]\!]$, public coefficient $a^j$ for all $j \in \{1, 2, \cdots, n\}$ and bais $b$.
Output: Shared secret $[\![f]\!]$ such that $f = \sum_{j=1}^{n} a^j \cdot x^j + b$.
1: $\mathcal{P}_1$ computes $f_1 \leftarrow \sum_{j=1}^{n} a^j \cdot x_1^j$.
2: $\mathcal{P}_2$ computes $f_2 \leftarrow \sum_{j=1}^{n} a^j \cdot x_2^j + b$.
3: Return $[\![f]\!]$.
```

Unlike linear operation, secure protocol for multiplication cannot be constructed by such intuitive way, since $[\![x \cdot y]\!] = [\![x]\!] \cdot [\![y]\!]$ fails. In 1991, Beaver [43] presented a crucial work to solve this problem, where a multiplicative triple (aka. Beaver triple) is used to assist parties to securely compute product. The idea of beaver triple has its roots in a simple but inconspicuous mind: obfuscate inputs and then use a particular linear combination to correct them. A Beaver triple consists of three numbers a, b, c such that $c = a \cdot b$ where a and b are random and will be generated and shared by ASS in offline phase. It means that each party $\mathcal{P}_i$ will own shares of input x_i and y_i and shares of Beaver triple a_i, b_i, and c_i before the execution of the protocol. In general, there are two ways to generate and share Beaver triple in offline phase: 1) done by parties through other MPC technic [49]; 2) done by trusted third party $\mathcal{T}$ [20]. Specially, in privacy-preserving computation, users can act as a trusted third party.

Secure multiplication protocol takes two shared secrets $[\![x]\!]$ and $[\![y]\!]$ as input, and return the shared secret $[\![x \cdot y]\!]$. Firstly, two random number a and b used to mask two multipliers x and y respectively via computing differences $[\![d]\!] \leftarrow [\![x]\!] - [\![a]\!]$, $[\![e]\!] \leftarrow [\![y]\!] - [\![b]\!]$. Secondly, parties reveal d and e to make them into public numbers. Since both a and b are random and privately shared, revealing d and e leaks no information about x and y. Now multiply two differences d and e to obtain $d \cdot e = x \cdot y - a \cdot y - b \cdot x + a \cdot b$, where $x \cdot y$ is the result that we want, meanwhile $a \cdot y$, $b \cdot x$, and $a \cdot b$ can be eliminated by $e \cdot [\![a]\!]$, $d \cdot [\![b]\!]$, and $[\![c]\!]$. Finally, it conclude that $[\![x \cdot y]\!] = d \cdot e + d \cdot [\![b]\!] + e \cdot [\![a]\!] + [\![c]\!]$. Algorithm 2 shows the details, where the term $e \cdot d$ can be computed by either party, and we specify $\mathcal{P}_2$ here without loss of generality. Moreover, secure protocol for product which has more than 2 multipliers can be implemented by recursively splitting input sequences in half and then computing each pair in parallel.

Theorem 2. Algorithm 2 is correct and UC-secure in the honest-but-curios model.

Proof: For correctness, there are

$$\begin{aligned} f =& f_1 + f_2 \\ =& c_1 + d \cdot b_1 + e \cdot a_1 + c_2 + d \cdot b_2 + e \cdot a_2 + e \cdot d \\ =& c + d \cdot b + e \cdot a + e \cdot d = x \cdot y \end{aligned}$$

To prove security, we first consider the incoming view and output of $\mathcal{P}_1$ and prove that both are simulatable. The

```
Algorithm 2 Secure Multiplication Protocol $[\![f]\!] \leftarrow SecMul([\![x]\!], [\![y]\!])$
Input:
    Shared secrets $[\![x]\!]$ and $[\![y]\!]$.
    Beaver triple: $[\![a]\!], [\![b]\!], [\![c]\!]$, where $c = a \cdot b$.
Output: Shared secret $[\![f]\!]$ such that $f = x \cdot y$.
1: $\mathcal{P}_1$ computes $d_1 \leftarrow x_1 - a_1, e_1 \leftarrow y_1 - b_1$.
2: $\mathcal{P}_2$ computes $d_2 \leftarrow x_2 - a_2, e_2 \leftarrow y_2 - b_2$.
3: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively reveal $d$ and $e$.
4: $\mathcal{P}_1$ computes $f_1 \leftarrow c_1 + d \cdot b_1 + e \cdot a_1$.
5: $\mathcal{P}_2$ computes $f_2 \leftarrow c_2 + d \cdot b_2 + e \cdot a_2 + e \cdot d$.
6: Return $[\![f]\!]$
```

incoming view of $\mathcal{P}_1$ is (d_2, e_2) and the output is (f_1), where $d_2 = x_2 - a_2$, $e_2 = y_2 - b_2$, and $f_1 = c_1 + d \cdot b_1 + e \cdot a_1$. The values a_2, b_2, and c_1 are uniformly distributed and independent from any private input, due to Lemma 1, the incoming view and output are simulatable. Since $\mathcal{P}_1$ and $\mathcal{P}_2$ are symmetric in Algorithm 2, it is trivial to build a simulator $\mathcal{S}$ for $\mathcal{P}_2$. □

5 Proposed Protocols

In this section, we firstly propose two resharing protocols. Then, based on these, we design a series of secure protocols for comparison, exponentiation, logarithm, power, and trigonometric functions. Finally, we discuss the remainder of basic elementary functions—inverse trigonometric functions.

5.1 Resharing

MSS described in Preliminaries has multiplicative homomorphism different from ASS, and in fact, is the basis of all the remaining protocols. However, all data (secret) should eventually return to the form shared by ASS in our setting, hence the resharing technic which switches shared secret between MSS and ASS need to be first implemented. Besides, based on MSS, we can directly construct secure protocol for comparison.

5.1.1 *Multiplicative Resharing*

The multiplicative resharing is to convert shared secret over MSS to over ASS. In fact, it is homologous with of multiplication. Recall that multiplication is essentially to compute $(f_1 + f_2) \leftarrow (x_1 + x_2) \cdot (y_1 + y_2)$, where the inputs of $\mathcal{P}_i$ are x_i and y_i; whereas multiplicative resharing is essentially to compute $(x_1 + x_2) \leftarrow (u_1 \cdot u_2)$, where the input of $\mathcal{P}_i$ is u_i. The outputs of both are isomorphic, while in terms of input, u_1 can be considered as $(x_1 + x_2)$ and u_2 can be considered as $(y_1 + y_2)$. That is to say, for $\mathcal{P}_1$, he has x_1 and x_2 instead of x_1 and y_1, thereby Beaver triple allocation needs to change accordingly. Specifically, a Beaver triple will be generated and the value c will be shared by ASS in offline phase as usual. The difference is that the values a and b will be directly sent to $\mathcal{P}_1$ and $\mathcal{P}_2$ instead of sharing them, since for $\mathcal{P}_1$, he owns x_1 and x_2 and hopes to get a_1 and a_2. In other words, before the execution of this protocol, $\mathcal{P}_1$ owns u_1, a, and c_1, meanwhile $\mathcal{P}_2$ owns u_2, b, and c_2. As illustrated in Algorithm 3, at first, $\mathcal{P}_1$ computes $d \leftarrow u_1 - a$ which can

Algorithm 3 Secure Multiplicative Resharing Protocol $[\![x]\!] \leftarrow SecMulRes(\langle u\rangle)$

```
Input:
    Shared secret $\langle u\rangle$ over MSS.
    Beaver triple: $a, b, [\![c]\!]$, where $c = a \cdot b$. The value $c$ is shared by ASS, meanwhile $a$ is held by $\mathcal{P}_1$ and $b$ is held by $\mathcal{P}_2$.
Output: Shared secret $[\![x]\!]$ over ASS such that $x = u$.
1: $\mathcal{P}_1$ computes $d \leftarrow u_1 - a$.
2: $\mathcal{P}_2$ computes $e \leftarrow u_2 - b$.
3: $\mathcal{P}_1$ sends $d$ to $\mathcal{P}_2$.
4: $\mathcal{P}_2$ sends $e$ to $\mathcal{P}_1$.
5: $\mathcal{P}_1$ computes $x_1 \leftarrow c_1 + e \cdot a$.
6: $\mathcal{P}_2$ computes $x_2 \leftarrow c_2 + d \cdot b + e \cdot d$.
7: Return $[\![x]\!]$
```

be seen as $[\![e]\!] \leftarrow [\![y]\!] - [\![b]\!]$. Then the values d and e are revealed by two parties. At last, $\mathcal{P}_1$ computes $x_1 \leftarrow c_1 + e \cdot a$ and $\mathcal{P}_2$ computes $x_2 \leftarrow c_2 + d \cdot b + e \cdot d$ respectively, which can be seen as $[\![c]\!] + d \cdot [\![b]\!] + e \cdot [\![a]\!] + e \cdot d$. Same as secure multiplication protocol, the term $e \cdot d$ is specified here to be computed by $\mathcal{P}_2$.

Theorem 3. Algorithm 3 is correct and UC-secure in the honest-but-curios model.

Proof: For correctness, there are

$$\begin{aligned} x =& x_1 + x_2 = c + d \cdot b + e \cdot a + e \cdot d \\ =& c + u_1 \cdot b - a \cdot b + u_2 \cdot a - a \cdot b \\ &+ u_1 \cdot u_2 - u_2 \cdot a - u_1 \cdot b + a \cdot b \\ =& u_1 \cdot u_2 = u \end{aligned}$$

To prove security, since $\mathcal{P}_1$ and $\mathcal{P}_2$ are symmetric in Algorithm 3, it suffices to show that the view of $\mathcal{P}_1$ is simulatable. The incoming view of $\mathcal{P}_1$ is (e) and the output is (x_1), where $e = u_2 - b$ and $x_1 = c_1 + e \cdot a$. Since the values b and c_1 are uniformly distributed and independent of any private input, due to Lemma 1, it is easy to perfectly build a simulator for $\mathcal{P}_1$. □

5.1.2 Additive Resharing

Additive resharing which converts shared secret over ASS to over MSS is defined as the inverse of multiplicative resharing and secure protocol for it can be constructed by following Algorithm 3 backward step by step. In offline phase, the allocation of Beaver triple is the same as Algorithm 3. In online phase, $\mathcal{P}_1$ and $\mathcal{P}_2$ first compute $[\![t]\!] = [\![x]\!] - [\![c]\!]$. Note that since the term $e \cdot d$ is assumed to be done by $[\![P]\!]_2$, the difference e can then be restored with $e \leftarrow t_1/a$. When $\mathcal{P}_2$ receives e from $\mathcal{P}_1$, the share of result u_2 can be directly obtained by computing $u_2 \leftarrow e + b$ and another difference d can be restored with $d \leftarrow t_2/u_2$. Lastly, $\mathcal{P}_1$ compute $u_1 \leftarrow d + a$ after he received d from $\mathcal{P}_2$. Algorithm 4 shows full details of this protocol and combines some computations.

It is worth mentioning that Algorithm 4 includes two division operations and thus may cause the problem of division by zero. However, in fact, the cases that a happens to be 0 or b happens to be $(c_1 - x_1)/a$ are negligible. Another

$\mathcal{F}_{AR}$ interacts with parties $\mathcal{P}_1$, $\mathcal{P}_2$, and adversary $\mathcal{S}$.

1. $\mathcal{F}_{AR}$ receives x_i from $\mathcal{P}_i$ for all $i = 1, 2$.
2. $\mathcal{F}_{AR}$ computes $x \leftarrow x_1 + x_2$.
3. $\mathcal{F}_{AR}$ chooses random nonzero value $u_2 \in \mathbb{F}$.
4. $\mathcal{F}_{AR}$ computes $u_1 \leftarrow x/u_2$.
5. $\mathcal{F}_{AR}$ sends u_1 to $\mathcal{P}_1$ and u_2 to $\mathcal{P}_2$.

Fig. 1. Additive resharing ideal functionality $\mathcal{F}_{AR}$.

Algorithm 4 Secure Additive Resharing Protocol $\langle u\rangle \leftarrow SecAddRes([\![x]\!])$

```
Input:
    Shared secret $[\![x]\!]$ over ASS.
    Beaver triple: $a, b, [\![c]\!]$, where $c = a \cdot b$. The value $c$ is shared by ASS, meanwhile $a$ is held by $\mathcal{P}_1$ and $b$ is held by $\mathcal{P}_2$.
Output: Shared secret $\langle u\rangle$ over MSS such that $u = x$.
1: $\mathcal{P}_1$ computes $e = (x_1 - c_1)/a$.
2: $\mathcal{P}_1$ sends $e$ to $\mathcal{P}_2$.
3: $\mathcal{P}_2$ computes $u_2 \leftarrow e + b, d \leftarrow (x_2 - c_2)/u_2$.
4: $\mathcal{P}_2$ sends $d$ to $\mathcal{P}_1$.
5: $\mathcal{P}_1$ computes $u_1 \leftarrow d + a$.
6: Return $\langle u\rangle$
```

case needs to point out is that when the input value $x = 0$ (i.e., $x_1 = -x_2$), the share of result

$$\begin{aligned} u_1 =& d + a = \frac{x_2 - c_2}{u_2} + a \\ =& \frac{-x_1 - c_2}{\frac{x_1 - c_1}{a} + b} + a = \frac{-ac + a^2 b}{x_1 - c_1 + ab} = 0, \end{aligned}$$

is identically equal to zero. This means that once $\mathcal{P}_1$ get $u_1 = 0$, he can know that the secret value $x = 0$, but it is the necessary price of performing this protocol. Even in the ideal world, one of the parties $\mathcal{P}_i$ will get knowledge of the secret value when it is zero. As illustrated in Figure 1, the ideal functionality $\mathcal{F}_{AR}$ which the additive resharing protocol wants to securely realize describes the behavior in the ideal world.

Theorem 4. Algorithm 4 is correct and UC-secure in the honest-but-curios model.

Proof: For correctness, there are

$$\begin{aligned} u =& u_1 \cdot u_2 = (d + a) \cdot (e + b) \\ =& x_2 - c_2 + ae + ab = x_2 - c_2 + x_1 - c_1 + ab \\ =& x_1 + x_2 = x \end{aligned}$$

To prove security, we need to show that both views of $\mathcal{P}_1$ and $\mathcal{P}_2$ are simulatable. For $\mathcal{P}_2$, his incoming view is (e), where $e = (x_1 - c_1)/a$. And for $\mathcal{P}_1$, his incoming view is (d), where $d = (x_2 - c_2)/(e + b)$. Since the values a, b, c_1, c_2 are uniformly distributed and independent of any private input, due to Lemma 1 and Lemma 2, the incoming views of $\mathcal{P}_1$ and $\mathcal{P}_2$ are simulatable. Besides, it is trivial to see that the outputs of $\mathcal{P}_1$ and $\mathcal{P}_2$ can be also simulated. □

Algorithm 5 Secure Comparison Protocol $SecCom([\![x]\!], [\![y]\!])$

Input: Shared secrets $[\![x]\!]$ and $[\![y]\!]$.
Output: 0 if $x == y$, 1 if $x > y$, -1 if $x < y$.

```
1:  $\mathcal{P}_1$ computes $d_1 \leftarrow x_1 - y_1$.
2:  $\mathcal{P}_2$ computes $d_2 \leftarrow x_2 - y_2$.
3:  $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $\langle u \rangle \leftarrow SecAddRes([\![d]\!])$.
4:  $\mathcal{P}_1$ and $\mathcal{P}_2$ reveal the signs of $u_1$ and $u_2$.
5:  if $u_1 == 0$ or $u_2 == 0$ then
6:      Return 0.
7:  else if ($u_1 > 0$ and $u_2 > 0$) or ($u_1 < 0$ and $u_2 < 0$) then
8:      Return 1.
9:  else
10:     Return $-1$.
11: end if
```

5.1.3 Comparison

It is easy to see that MSS allows share to reserve the sign of secret, thus secure comparison can be accomplished via additive resharing. As presented in Algorithm 5, $\mathcal{P}_1$ and $\mathcal{P}_2$ compute the difference $[\![d]\!] = [\![x]\!] - [\![y]\!]$ at first, then call secure additive resharing protocol to get the difference $\langle u \rangle$ over MSS, and after that, the sign of $\langle u \rangle$ will be revealed between each party. Since there exists round-off error in practice, two floating-point numbers are almost impossible to be completely equal. A precision threshold Δ could be set to determine whether two floats are equal, i.e., whether their difference is reduced to zero. When the signs of all multiplicative shares are public, parties will arrive at a consensus over the comparison of two numbers.

Theorem 5. Algorithm 5 is correct and UC-secure in honest-but-curios model.

Proof: It is trivial to prove the correctness and security of Algorithm 5, since those of `SecAddRes` protocol has been proven. □

5.2 Nonlinear Functions

Switching shared secret between two sharing forms as well as some identities makes possible secure protocols for exponentiation, logarithm, and power without any approximate or bit-level operation. Besides, the division can be seen as a special case of power.

5.2.1 Exponentiation

Secure exponentiation is to compute $[\![f]\!] \leftarrow a^{[\![x]\!]}$, where a is a public base number and is usually specified as e. According to the identity

$$a^{x_1+x_2} = a^{x_1} \cdot a^{x_2}, \tag{1}$$

when parties perform exponential operation on their shares, the problem can be transformed into the problem of converting multiplicative shares which has been solved by `SecMulRes`. Without loss of generality, Algorithm 6 takes a secret exponent $[\![x]\!]$ and a public base number a as input, and returns the shared result $[\![a^x]\!]$.

In addition, when base number a is negative, an inappropriate exponent such as $1/2$ will lead to a mistake in computing. However, such case should be avoided in designing function, not be guaranteed of this algorithm.

Algorithm 6 Secure Exponentiation Protocol $[\![f]\!] \leftarrow SecExp([\![x]\!], a)$

Input: Shared secret $[\![x]\!]$ and public base number a.
Output: Shared secret $[\![f]\!]$ such that $f = a^x$.

```
1: $\mathcal{P}_1$ computes $u_1 \leftarrow a^{x_1}$.
2: $\mathcal{P}_2$ computes $u_2 \leftarrow a^{x_2}$.
3: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f]\!] \leftarrow SecMulRes(\langle u \rangle)$.
4: Return $[\![f]\!]$.
```

Theorem 6. Algorithm 6 is correct and UC-secure in honest-but-curios model.

Proof: According to correctness of `SecMulRes`, we have

$$f_1 + f_2 = u_1 \cdot u_2,$$

therefore, the correctness of this algorithm can be proven as follow

$$f = f_1 + f_2 = u_1 \cdot u_2 = a^{x_1} \cdot a^{x_2} = a^{x_1+x_2} = a^x.$$

Since `SecMulRes` is secure and this algorithm contains no additional communication, security is trivial. □

5.2.2 Logarithm

As the inverse of exponentiation, secure logarithm is to compute $[\![f]\!] \leftarrow \log_a [\![x]\!]$, where a is a base number. Similar to exponentiation, logarithm has its own ”product-to-sum” identity

$$\log_a(u_1 \cdot u_2) = \log_a u_1 + \log_a u_2. \tag{2}$$

From this point, parties first call the protocol `SecAddRes` to convert additive shares into multiplicative shares and then perform logarithmic operation on each share. Since logarithmic function requires positive input, we have to make the share absolute in advance. It means that what we actually designed is the secure function $[\![f]\!] \leftarrow \log_a |[\![x]\!]|$, and which is more reasonable. Note that the steps of `SecLog` are exactly the opposite of `SecExp`, in which the steps of `SecAddRes` and `SecMulRes` they called are also opposite.

Theorem 7. Algorithm 7 is correct and UC-secure in honest-but-curios model.

Proof: For correctness, there are

$$f = f_1 + f_2 = \log_a |u_1| + \log_a |u_2| = \log_a |u_1 \cdot u_2|,$$

due to the correctness of `SecAddRes`, we have

$$f = \log_a |u_1 \cdot u_2| = \log_a |x_1 + x_2| = \log_a |x|.$$

Since `SecAddRes` is secure and this algorithm contains no additional communication, security is trivial. □

Algorithm 7 Secure Logarithm Protocol $[\![f]\!] \leftarrow SecLog([\![x]\!], a)$

```
Input: Shared secret $[\![x]\!]$ and public base number $a$.
Output: Shared secret $[\![f]\!]$ such that $f = \log_a |x|$.
1: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $\langle u \rangle \leftarrow SecAddRes([\![x]\!])$.
2: $\mathcal{P}_1$ computes $f_1 \leftarrow \log_a |u_1|$.
3: $\mathcal{P}_2$ computes $f_2 \leftarrow \log_a |u_2|$.
4: Return $[\![f]\!]$.
```

Algorithm 8 Secure Power Protocol $[\![f]\!] \leftarrow SecPow([\![x^1]\!], [\![x^2]\!], \cdots, [\![x^n]\!], \alpha^1, \alpha^2, \cdots, \alpha^n)$

```
Input: Shared secrets $[\![x^j]\!]$ and public integers $\alpha^j$ for all $j \in \{1, 2, \cdots, n\}$.
Output: Shared secret $[\![f]\!]$ such that $f = \prod_{j=1}^n {x^j}^{\alpha^j}$.
1: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $\langle u^j \rangle \leftarrow SecAddRes([\![x^j]\!])$ for all $j \in \{1, 2, \cdots, n\}$ in parallel.
2: $\mathcal{P}_1$ computes $v_1 \leftarrow \prod_{j=1}^n {u_1^j}^{\alpha^j}$.
3: $\mathcal{P}_2$ computes $v_2 \leftarrow \prod_{j=1}^n {u_2^j}^{\alpha^j}$.
4: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f]\!] \leftarrow SecMulRes(\langle v \rangle)$.
5: Return $[\![f]\!]$.
```

5.2.3 *Power*

Secure power is to compute $[\![f]\!] \leftarrow [\![x]\!]^\alpha$, where α is a public integer. Inspired by the identity

$$(u_1 \cdot u_2)^\alpha = u_1^\alpha \cdot u_2^\alpha, \tag{3}$$

we can first turn shares into multiplicative shares, then perform power operation on each share, and finally, turn shares back to additive. In fact, the above identity also holds for multiple multipliers, in other words, multiplication and power are to MSS what linear is to ASS. For n multipliers, we have

$$(u_1^1 u_2^1 u_1^2 u_2^2 \cdots u_1^n u_2^n)^\alpha = (u_1^1 u_1^2 \cdots u_1^n)^\alpha \cdot (u_2^1 u_2^2 \cdots u_2^n)^\alpha. \tag{4}$$

Therefore, as shown in Algorithm 8, we design secure power protocol which implements secure function $[\![f]\!] \leftarrow \prod_{j=1}^n [\![x]\!]^{j^{\alpha^j}}$. Note that division is the special case of power, i.e., SecPow($[\![x]\!]$,$[\![y]\!]$,1,−1), and product is also the special case, i.e., SecPow($[\![x^1]\!]$,$[\![x^2]\!]$,$\cdots$,$[\![x^n]\!]$,1,1,$\cdots$,1).

In addition, when the exponent α is a real number or even a shared secret, this algorithm no longer applies. We discussed both cases in the Appendix.

Theorem 8. Algorithm 8 is correct and UC-secure in honest-but-curios model.

Proof: For correctness, there are

$$f = f_1 + f_2 = v_1 \cdot v_2 = \prod_{j=1}^n {u_1^j}^{\alpha^j} \cdot {u_2^j}^{\alpha^j} = \prod_{j=1}^n {u^j}^{\alpha^j} = \prod_{j=1}^n {x^j}^{\alpha^j}.$$

Since `SecAddRes` and `SecMulRes` are secure and this algorithm contains no additional communication, security is trivial. □

Algorithm 9 Secure Sine Protocol $[\![f]\!] \leftarrow SecSin([\![x]\!])$

```
Input: Shared secret $[\![x]\!]$.
Output: Shared secret $[\![f]\!]$ such that $f = \sin x$.
1: $\mathcal{P}_1$ computes $m_1 \leftarrow \sin x_1, n_1 \leftarrow \cos x_1$.
2: $\mathcal{P}_2$ computes $n_2 \leftarrow \sin x_2, m_2 \leftarrow \cos x_2$.
3: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f^m]\!] \leftarrow SecMulRes(\langle m \rangle), [\![f^n]\!] \leftarrow SecMulRes(\langle n \rangle)$ in parallel.
4: $\mathcal{P}_1$ computes $f_1 \leftarrow f_1^m + f_1^n$.
5: $\mathcal{P}_2$ computes $f_2 \leftarrow f_2^m + f_2^n$.
6: Return $[\![f]\!]$.
```

Algorithm 10 Secure Cosine Protocol $[\![f]\!] \leftarrow SecCos([\![x]\!])$

```
Input: Shared secret $[\![x]\!]$.
Output: Shared secret $[\![f]\!]$ such that $f = \cos x$.
1: $\mathcal{P}_1$ computes $m_1 \leftarrow \sin x_1, n_1 \leftarrow \cos x_1$.
2: $\mathcal{P}_2$ computes $m_2 \leftarrow \sin x_2, n_2 \leftarrow \cos x_2$.
3: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f^m]\!] \leftarrow SecMulRes(\langle m \rangle), [\![f^n]\!] \leftarrow SecMulRes(\langle n \rangle)$ in parallel.
4: $\mathcal{P}_1$ computes $f_1 \leftarrow f_1^n - f_1^m$.
5: $\mathcal{P}_2$ computes $f_2 \leftarrow f_2^n - f_2^m$.
6: Return $[\![f]\!]$.
```

5.2.4 *Trigonometric Functions*

Trigonometric functions which contain sine, cosine, tangent, cotangent, cosecant, and secant describe the relationship of an angle and two edges of a right triangle. We first design secure protocols for two basic functions: sine and cosine, then discuss the remains.

Secure sine is to compute $[\![f]\!] \leftarrow \sin[\![x]\!]$. Considering the angle sum and difference identity

$$\sin(x_1 + x_2) = \sin x_1 \cdot \cos x_2 + \cos x_1 \cdot \sin x_2, \tag{5}$$

the problem is transformed into how to convert shares. Algorithm 9 first computes the sine and cosine of each share, then divides above four trigonometric results into two groups of multiplicative shares, and the rest is to call `SecMulRes` for each group in parallel.

Secure cosine is to compute $[\![f]\!] \leftarrow \cos[\![x]\!]$. Similar to Algorithm 9, it is easy to design secure cosine protocol that securely implements cosine according to the identity

$$\cos(x_1 + x_2) = \cos x_1 \cdot \cos x_2 - \sin x_1 \cdot \sin x_2. \tag{6}$$

Algorithm 10 shows the details.

Theorem 9. Algorithm 9 and Algorithm 10 are correct and secure in honest-but-curios model.

Proof: For the correctness of Algorithm 9, there are

$$\begin{aligned} f &= f_1 + f_2 = f_1^m + f_1^n + f_2^m + f_2^n = f^m + f^n \\ &= m + n = m_1 m_2 + n_1 n_2 = \sin x_1 \cdot \cos x_2 + \cos x_1 \cdot \sin x_2 \\ &= \sin(x_1 + x_2) = \sin x. \end{aligned}$$

Referring to the above proof, it is trivial to prove the correctness of Algorithm 10. Since `SecMulRes` is secure and Algorithm 9 and Algorithm 10 contain no additional communication, security of both is trivial. □

Tangent, cotangent, cosecant, and secant could be defined by sine and cosine as $^{sine}/_{cosine}$, $^{cosine}/_{sine}$, $^{1}/_{sine}$, and $^{1}/_{cosine}$ respectively. Therefore, constructing secure protocols for these functions is merely a combination of `SecSin`, `SecCos`, and `SecPow`, and it is trivial to prove the correctness and security of them.

5.3 Inverse Trigonometric Functions

The inverse trigonometric functions is the final problem to integrity basic elementary functions. However, there is no angle sum and difference identity for inverse trigonometric functions. Therefore, we have to settle for second best by using Taylor series. For instance, $arcsin(x)$ can be calculated using Maclaurin series:

$$arcsin(x) = x + \left(\frac{1}{2}\right)\frac{x^3}{3} + \left(\frac{1\cdot 3}{2\cdot 4}\right)\frac{x^5}{5} + \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)\frac{x^7}{7} + \cdots \tag{7}$$

Since secure power protocol is with constant round complexities, high order Taylor series (Maclaurin series) will not increase additional interaction between parties. As a result, we could increase the order of Taylor series as much as possible to improve accuracy.

6 Complexity

Table 1 summarized round complexities and communications of all the protocols in this paper. We can see that all proposed protocols have low constant round complexities and low communications, which is suitable for practice.

The round complexities of `SecCom` are 3 which contains 2 rounds of `SecAddRes` and 1 round of revealing signs. As the sign of secret over MSS is precisely in ASS form, parties do not need to know the exact result of comparison in some scenarios. Consequently, `SecAddRes` can be regarded as secure comparison protocol with semaphore, and its round complexities are 2.

Secure product is available in two methods: 1) recursively splitting input sequences in half and then computing each pair in parallel; 2) as a special case of power. The Round complexities of the two methods are $\lceil\log_2 n\rceil$ and the constant 3, and communications are $(4n-4)l$ and $(2n+2)l$, respectively. Hence the round complexities of product are $min(3, \lceil\log_2 n\rceil)$ by selecting the appropriate method according to the number of inputs.

The round complexities of secure protocols for tangent, cotangent, cosecant and secant defined by sine and cosine are 4, while what using the angle sum and difference identities are 4 as well.

Table 2 compared the complexities of secure protocols for comparison and division with previous work, and showed that our scheme is more efficient.

TABLE 1
Complexities of protocols
(Here, l is the bit-width of the element in practice, and n denotes the number of inputs)

Protocol	Round Complexities	Communications
`SecMul`	1	$4l$
`SecMulRes`	1	$2l$
`SecAddRes`	2	$2l$
`SecCom`	3	$2l+2$
`SecExp`	1	$2l$
`SecLog`	2	$2l$
`SecPow`	3	$(2n+2)l$
`SecSin`, `SecCos`	1	$4l$
Division	3	$6l$
Product	$min(3, \lceil\log_2 n\rceil)$	$(2n+2)l$, $(4n-4)l$
Tagent, Cotangent	4	$14l$
Secant, Cosecant	4	$12l$

7 Conclusions

In this paper, we firstly proposed two basic protocols, multiplicative resharing and additive resharing, which can switch shared secrets. And then, based on this ability, protocols for comparison, exponentiation, logarithm, power, product, division, and trigonometric functions were proposed, where product and division derived from power. Additionally, we discussed the cases about power with real exponent or with secret exponent in Appendix, and constructed protocols for them. All the protocols proposed in this paper are composed of arithmetic operations and contain no bit-level operation with low constant round complexities, and have been proven secure in honest-but-curios model.

Specially, secure power protocol with constant round complexities provides the possibility of higher order Taylor series which can approximate most functions that include those not implemented in this paper and will benefit from higher order. Meanwhile, secure sine and cosine protocols provide the possibility of efficient Fourier series which can approximate an arbitrary function in a period.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under grant numbers 61702276, 61672294, 61502242, in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD) fund.

References

[1] M. Hamdaqa and L. Tahvildari, "Cloud computing uncovered: a research landscape," in *Advances in Computers*. Elsevier, 2012, vol. 86, pp. 41–85.
[2] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.
[3] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.
[4] M. D. Ryan, "Cloud computing privacy concerns on our doorstep," *Communications of the ACM*, vol. 54, no. 1, pp. 36–38, 2011.
[5] J. Mancuso, "Privacy-preserving machine learning 2018: A year in review," Tech. Rep., 2019.
[6] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.
[7] T. Sander, A. Young, and M. Yung, "Non-interactive cryptocomputing for nc/sup 1," in *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 1999, pp. 554–566.
[8] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory of Cryptography Conference*. Springer, 2005, pp. 325–341.

TABLE 2
Comparison of Round Complexities and Communications
(Here, l is the bit-width, $l' = \log_2 l$, and the values of m, n' determine the accuracy of division)

Scheme	Comparison		Division	
	Rounds	Comm.	Rounds	Comm.
[18]	15	$279l + 5$	-	-
[19]	$l' + 3$	$5l'^2 + 12(l' + 1)l$	$4l' + 9$	$2ml + 6ml' + 39l'l + 35l'n' + 126l + 32n' + 24$
[20]	$l + 3$	$10l - 2$	-	-
Ours	3	$2l + 2$	3	$6l$

[9] Y. Ishai and A. Paskin, "Evaluating branching programs on encrypted data," in *Theory of Cryptography Conference*. Springer, 2007, pp. 575–594.

[10] C. Gentry and D. Boneh, *A fully homomorphic encryption scheme*. Stanford university Stanford, 2009, vol. 20, no. 9.

[11] J.-S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 446–464.

[12] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Annual Cryptology Conference*. Springer, 2011, pp. 487–504.

[13] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[14] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 1986, pp. 162–167.

[15] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 351–371.

[16] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

[17] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[18] T. Nishide and K. Ohta, "Multiparty computation for interval, equality, and comparison without bit-decomposition protocol," in *International Workshop on Public Key Cryptography*. Springer, 2007, pp. 343–360.

[19] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *International Journal of Information Security*, vol. 11, no. 6, pp. 403–418, 2012.

[20] K. Huang, X. Liu, S. Fu, D. Guo, and M. Xu, "A lightweight privacy-preserving cnn feature extraction framework for mobile sensing," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[21] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[22] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[23] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[24] S. Goldwasser and S. Micali, "Probabilistic encryption &amp; how to play mental poker keeping secret all partial information," in *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 1982, pp. 365–377.

[25] J. Benaloh, "Dense probabilistic encryption," in *Proceedings of the workshop on selected areas of cryptography*, 1994, pp. 120–128.

[26] T. Okamoto and S. Uchiyama, "A new public-key cryptosystem as secure as factoring," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1998, pp. 308–318.

[27] D. Naccache and J. Stern, "A new public key cryptosystem based on higher residues," in *Proceedings of the 5th ACM conference on Computer and communications security*, 1998, pp. 59–66.

[28] S. D. Galbraith, "Elliptic curve paillier schemes," *Journal of Cryptology*, vol. 15, no. 2, pp. 129–138, 2002.

[29] A. Kawachi, K. Tanaka, and K. Xagawa, "Multi-bit cryptosystems based on lattice problems," in *International Workshop on Public Key Cryptography*. Springer, 2007, pp. 315–329.

[30] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[31] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 24–43.

[32] P. S. Pisa, M. Abdalla, and O. C. M. B. Duarte, "Somewhat homomorphic encryption scheme for arithmetic operations on large integers," in *2012 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, 2012, pp. 1–8.

[33] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Annual cryptology conference*. Springer, 2011, pp. 505–524.

[34] ——, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[35] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[36] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, 1990, pp. 503–513.

[37] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 411–428.

[38] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[39] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 129–139.

[40] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2008, pp. 486–498.

[41] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[42] J. Kilian, "Founding crytpography on oblivious transfer," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, 1988, pp. 20–31.

[43] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual International Cryptology Conference*. Springer, 1991, pp. 420–432.

[44] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography Conference*. Springer, 2006, pp. 285–304.

[45] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 192–206.

[46] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004, p. 9.

[47] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Tasty: tool for automating secure two-party computations," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 451–462.

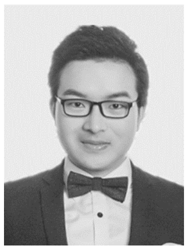
[48] S. Wagh, D. Gupta, and N. Chandran, “Securenn: Efficient and private neural network training.” *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 442, 2018.

[49] P. Mohassel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[50] S. Goldwasser and S. Micali, “Probabilistic encryption,” *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[51] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

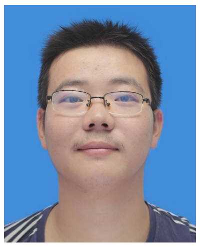
[52] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.



Lizhi Xiong received Ph.D. degree in Communication and Information System from Wuhan University, China in 2016. From 2014 to 2015, he was a Joint-Ph.D. student with Electrical and Computer Engineering, New Jersey University of Technology, New Jersey, USA. He is currently an Associate Professor with School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, China. His main research interests include privacy-preserving computation, information hiding, and multimedia security.

Wenhao Zhou is currently pursuing his M.S. degree in the School of Computer and Software, Nanjing University of Information Science and Technology, China. His research interests include secure multiparty computation and privacy-preserving computation.

Zhihua Xia received his B.S. degree in Hunan City University, China, in 2006, and the Ph.D. degree in computer science and technology from Hunan University, China, in 2011.He is currently an associate professor with the School of Computer and Software, Nanjing University of Information Science and Technology, China. He was a visiting professor with the Sungkyunkwan University, Korea, 2016. His research interests include cloud computing security and digital forensic. He is a member of the IEEE.



Qi Gu is currently pursuing his M.S. degree in the School of Computer and Software, Nanjing University of Information Science and Technology, China. His research interests include functional encryption, image retrieval and nearest neighbor search.

Jian Weng received the B.S. and M.S. degrees in computer science and engineering from South China University of Technology, Guangzhou, China, in 2000 and 2004, respectively, and the Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China, in 2008. From 2008 to 2010, he held a Postdoctoral position with the School of Information Systems, Singapore Management University. He is currently a Professor and the Dean with the College of Information Science and Technology, Jinan University, Guangzhou,China. He has authored or coauthored more than 100 papers in cryptography and security conferences and journals, such as CRYPTO, EUROCRYPT, ASIACRYPT, TCC, PKC, TPAMI, TIFS, and TDSC. His research interests include public key cryptography, cloud security, and blockchain. He was the PC Co-Chairs or PC Member for more than 30 international conferences. He also serves as an Associate Editor for the IEEE TRANSACTIONS ON VEHICULART ECHNOLOGY.

Appendix
Power with Real or Secret Exponent

When the exponent is real as well as the base is negative, Algorithm 8 will produce wrong result, since the identity 3 doesn't hold under this case. However, we note that the identity

$$u^{\alpha} = (-1)^{\alpha}|u|^{\alpha}, \tag{8}$$

where u is a negative number. This provides us with an idea: firstly, compute $|[\![x]\!]|^{\alpha}$ as with Algorithm 8, secondly, multiply each share by $(-1)^{\alpha}$ if x is negative. We thus constructed secure power with real exponent protocol as described in Algorithm 11, where to simplify the problem, the case with one multiplier is considered. The protocol is with complexity of 4 rounds and $4l + 2$ communications.

Besides, it should be emphasized that the coefficient $(-1)^{\alpha}$ usually not only directs the real field to the complex field, but also leads to multiple results. Although all protocols in this paper also available in the complex field with necessary adjustments, many practical applications do not need to enter the complex field. In this case, it only needs to compute $|[\![x]\!]|^{\alpha}$ without multiplying $(-1)^{\alpha}$. For example, batch normalization [52] involving root operation assumes that the input is positive.

Theorem 10. Algorithm 11 is correct and UC-secure in honest-but-curios model.

Proof: For correctness, if x is negative, there are

$$f = (-1)^{\alpha}(f_1 + f_2) = (-1)^{\alpha}(v_1 v_2) = (-1)^{\alpha}|u|^{\alpha} = x^{\alpha}.$$

If not, it is same as Algorithm 8. Since `SecAddRes`, `SecCom`, `SecMulRes` are secure and this algorithm contains no additional communication, security is trivial. □

Algorithm 11 Secure Power with Real Exponent Protocol $[\![f]\!] \leftarrow SecPRE([\![x]\!], \alpha)$

```
Input: Shared secret $[\![x]\!]$ and public real $\alpha$.
Output: Shared secret $[\![f]\!]$ such that $f = x^{\alpha}$.
1: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $\langle u \rangle \leftarrow SecAddRes([\![x]\!])$.
2: $\mathcal{P}_1$ computes $v_1 \leftarrow |u_1|^{\alpha}$.
3: $\mathcal{P}_2$ computes $v_2 \leftarrow |u_2|^{\alpha}$.
4: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f]\!] \leftarrow SecMulRes(\langle v \rangle)$.
5: if $SecCom([\![x]\!], [\![0]\!]) == -1$ then
6:     $\mathcal{P}_1$ computes $f_1 \leftarrow f_1 \cdot (-1)^{\alpha}$.
7:     $\mathcal{P}_2$ computes $f_2 \leftarrow f_2 \cdot (-1)^{\alpha}$.
8: end if
9: Return $[\![f]\!]$.
```

We now discuss the case that even the exponent is a shared secret. Since share cannot be exponentiated directly, we need to use the identity

$$\alpha \ln u = \ln u^{\alpha}, \tag{9}$$

to convert exponentiation into multiplication. In addition, as a shared secret, the exponent is almost impossible to be an integer, thereby we should also classify the positive and negative of the base. Algorithm 12 shows the details of secure power with secret exponent protocol. The protocol requires complexity of 5 rounds and $8l+2$ communications.

Theorem 11. Algorithm 12 is correct and UC-secure in honest-but-curios model.

Proof: For correctness, if x is negative, there are

$$\begin{aligned} f =& f_1 + f_2 = v_1 \cdot v_2 = e^{s_1} \cdot (-1)^{y_1} \cdot e^{s_2} \cdot (-1)^{y_2} = (-1)^y e^s \\ =& (-1)^y e^{ty} = (-1)^y e^{y \ln |x|} = (-1)^y |x|^y = x^y. \end{aligned}$$

Otherwise, it is trivial. Since this algorithm consists of only secure protocols and contains no additional communication, security is trivial. □

Algorithm 12 Secure Power with Secret Exponent Protocol $[\![f]\!] \leftarrow SecPSE([\![x]\!], [\![y]\!])$

```
Input: Shared secrets $[\![x]\!]$ and $[\![y]\!]$.
Output: Shared secret $[\![f]\!]$ such that $f = x^y$.
1: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![t]\!] \leftarrow SecLog([\![x]\!])$.
2: $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![s]\!] \leftarrow SecMul([\![t]\!], [\![y]\!])$.
3: if $SecCom([\![x]\!], [\![0]\!]) == 1$ then
4:     $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f]\!] \leftarrow SecExp([\![s]\!])$.
5:     Return $[\![f]\!]$.
6: else if $SecCom([\![x]\!], [\![0]\!]) == -1$ then
7:     $\mathcal{P}_1$ computes $v_1 \leftarrow e^{s_1} \cdot (-1)^{y_1}$.
8:     $\mathcal{P}_2$ computes $v_2 \leftarrow e^{s_2} \cdot (-1)^{y_2}$.
9:     $\mathcal{P}_1$ and $\mathcal{P}_2$ collaboratively compute $[\![f]\!] \leftarrow SecMulRes(\langle v \rangle)$.
10:    Return $[\![f]\!]$.
11: else
12:    Return $[\![0]\!]$.
13: end if
```