Fair Outsourcing Polynomial Computation Based on the Blockchain

Yunguo Guan, Hui Zheng, Jun Shao, Rongxing Lu, *Fellow, IEEE*, and Guiyi Wei

Abstract—Due to the big data blowout from the Internet of Things and the rapid development of cloud computing, outsourcing computation has received considerable attention in recent years. Particularly, many outsourcing computation schemes have been proposed to dedicate the outsourcing polynomial computation due to its use in numerous fields, such as data analysis and machine learning. However, none of those schemes are practical enough, as they either require some time-consuming cryptographic operations to achieve fairness between the user and the worker, or cannot allow the user to outsource arbitrary polynomial to the worker, or need two non-collusive workers. To tackle these challenges, in this article, we propose a new outsourcing polynomial computation scheme by employing a variant of Horner's method and the blockchain technology. Specifically, the former makes the computational cost on the worker side as low as possible, and the latter guarantees the fairness between the user and the worker if the result from the worker can be publicly verified. To achieve the public verifiability property, we apply the sampling technique, which is effective in our proposal according to a game-theoretic analysis. Furthermore, we also implement a prototype of our proposal and run it on an Ethereum test net. The extensive experimental results demonstrate that our proposal is efficient in terms of computational cost.

Index Terms—Outsourcing computation, polynomial computation, fairness, public verifiablility, blockchain, sampling technique, game theory

1 Introduction

As it is fundamental to the success of intelligent applications in our daily lives, the Internet of Things (IoT) has received considerable attention in recent years. For instance, it has been reported that more than 50 billion IoT devices would be connected to the Internet in 2020 [1]. The rapid growth of IoT devices contributes to the blowout of big data, and the latter further contributes to the increasing demand to outsource relatively complex computing tasks to a much more powerful computation service, e.g., cloud computing [2]. Meanwhile, with the advancement of computer hardware, there would be many idle cycles existing in many computing devices, and their owners are willing to sell these surplus computing resources to make some profits [3]. Therefore, it is fair to say both the demand and supply of outsourcing computation are prosperous.

However, the advance of outsourcing computation, in reality, is not so successful as expected. The main obstacle is the distrust between the two participants of the computing task, namely, the user and the worker. In particular, the worker may return a wrong but plausible result without performing the actual computation for some incentive reason. At the same time, the user may try to obtain the result without any payment. As a result, this situation intensely asks for the fairness of the outsourcing computation scheme. Informally speaking, the fairness guarantees that the user (resp. worker) can get a valid computing result (resp. the promised reward) if he/she follows the protocol exactly. Nevertheless, most of the previously reported outsourcing computation schemes [4], [5], [6], [7], [8], [9], [10] usually focus on guaranteeing that the user can obtain a valid result before his/her payment while ignoring the worker's benefit. When a dispute between the user and worker happens, a complex process involving a trusted third party (TTP) would be required [11]. As an ex-post measure, the TTP ensures that the dispute will be finally addressed, but it cannot promptly respond to the dispute. Moreover, TTP-based schemes are weak in scalability and suffer from a single point of failure. Hence, there is a desire to design a fair outsourcing computation scheme without TTP.

Based on the blockchain technology [12], some fair outsourcing computation schemes without TTP [13], [14] have recently been proposed.[1] As the blockchain can be used to build a trust relationship among untrusted entities, it is natural to introduce the blockchain into the outsourcing computation scenario. However, the existing blockchain-based solutions require either complex cryptographic primitives or the involvement of non-collusive workers. The former requirement may make the resulting scheme inefficient, and

- *Yunguo Guan is with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, Zhejiang 310018, China, and also with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B5A3, Canada. E-mail: guan_yg@163.com.*
- *Hui Zheng and Jun Shao are with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, Zhejiang 310018, China. E-mail: {zh312934, chn.junshao}@gmail.com.*
- *Rongxing Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B5A3, Canada. E-mail: rlu1@unb.ca.*
- *Guiyi Wei is with the School of Electronic and Information Engineering, Hangzhou, Zhejiang 310018, China. E-mail: weigy@zjgsu.edu.cn.*

Manuscript received 1 June 2020; revised 18 Jan. 2021; accepted 22 Jan. 2021. Date of publication 26 Jan. 2021; date of current version 7 Oct. 2022. (Corresponding author: Jun Shao.)
Digital Object Identifier no. 10.1109/TSC.2021.3054772

1. Note that the blockchain-based solutions do not overturn the impossibility result on the fair exchange without a trusted third party [11]. The blockchain in such solutions is essentially the TTP, though it is a decentralized system.

1939-1374 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
The Comparison Between Our Proposal and Previous Schemes

Types	Schemes	Fairness	W/O Crypto-Operations	W/O Non-collusive Servers	Arbitrary	Dynamic Coefficients
Crypto-based	GGP10 [4]	×	×	√	√	×
	FG12 [5]	×	×	√	×	×
	PST13 [6]	×	×	√	×	√
	EOA16 [7]	×	×	√	×	×
	SWW17 [8]	×	×	√	×	×
	SZQ18 [9]	×	×	√	×	×
	ZXZS20 [26]	×	×	√	√	×
Replica-based	CRR13 [10]	×	×	×	√	×
Blockchain-based	KB14 [13]	√	×	√	√	√
	DWA17 [14]	√	√	×	√	√
Our proposal		√	√	√	√	√

the latter may not always be true in reality, especially in the blockchain system.[2]

Nowadays, the previously reported outsourcing computation schemes can be roughly classified into two categories: general and specific. The former can outsource any computing task, while the latter can only support a specific one. Generally speaking, schemes in the latter are more efficient than those in the former, since they can make some targeted optimizations related to computing tasks [15], [16]. Also, because the efficiency is always a critical factor for outsourcing computation, schemes in the latter are more attractive than those in the former from a practical point of view. Among all particular computing tasks, polynomial computation, containing only two arithmetic operations, namely, addition and multiplication, has received considerable attention. This is mainly because it can be applied in various scenarios, e.g., data analysis [17], [18], scientific computing [19], [20], and machine learning [21], [22], and a large amount of computational tasks can be represented as arithmetic circuits and further evaluated as polynomials [23]. In this paper, as a step towards efficient general-purpose fair outsourced computation, we focus on designing a blockchain-based fair outsourcing polynomial computation scheme. Specifically, the contribution of this paper can be summarized as follows.

- First, based on the blockchain technology [12] and Horner's method [24], we propose a new outsourcing polynomial computation scheme. The fairness of our proposal is demonstrated through game-theoretic analyses.
- Second, compared to the previous schemes, our proposal supports arbitrary polynomials and dynamic coefficients without any complex cryptographic operations or the involvement of two non-collusive servers. A detailed comparison with some representative schemes can be found in Table 1 and will be further explained in Section 6.
- Third, we also implement a prototype of our proposal and run it on Rinkeby Ethereum Testnet [25]. The experimental results show that our proposal is efficient in terms of computational cost. The details of our experiments can be found in Section 5.2.

2. According to the statistics from https://blockchain.com/pools, more than 15 percent of mining power may come from the same mining pool.

The remainder of this paper is organized as follows. In Section 2, we formalize the system model and security model, and identify our design goals. Then, we give some preliminaries including blockchain and Horner's method in Section 3. In Section 4, we present the details of our proposed scheme, followed by the performance evaluation in Section 5. Section 6 reviews the related works. In the end, Section 7 gives the conclusions of our paper.

2 Models and Design Goals

In this section, we formalize our system model and security model, and identify our design goals.

2.1 System Model

As shown in Fig. 1, in our system model, we mainly consider a typical blockchain-based outsourcing computation scenario, which mainly consists of four entities, namely, a user, a worker, a blockchain system, and Interplanetary File System (IPFS).

- *User:* In our system model, we consider the user needs to evaluate a polynomial $f(x)$ with a given x, but it is incapable in running this computing task due to its restricted computing and energy resources. Therefore, he/she tends to outsource the polynomial computing task to get the value of $f(x)$ with as little overhead as possible.
- *Worker:* In our system model, the worker could be a cloud server or even a computing device belonging to an individual, and it would like to employ its computing power to run polynomial computing tasks for

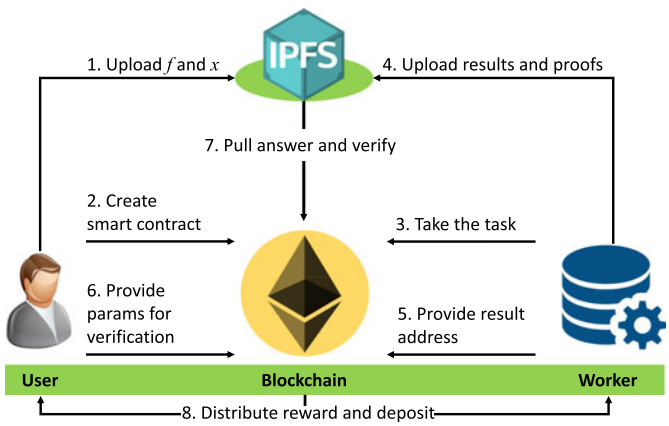


Fig. 1. The system model under consideration.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

others. Assume that the worker can find tasks published by the user from the blockchain, which could be realized by some explorer, like the bitcoin explorer (https://www.blockchain.com/explorer) showing everything on Bitcoin.

- *Blockchain:* Our proposal utilizes a blockchain system supporting smart contracts, e.g., Ethereum [27], EOS [28], or NEO [29], to guarantee fairness. As shown in Fig. 1, the blockchain sits between the user and the worker to supervise their interactions and takes the role of verifying the computing result. Moreover, it is responsible for managing the revenue and deposit, respectively, from the user and the worker, such that whoever deviates from the protocol will pay the price.
- *IPFS:* As the blockchain is usually designed to handle small-sized data, our proposal employs IPFS to store the proof of correctness generated by the worker. Specifically, after conducting the task, the worker uploads the computing result and proof to IPFS and submits their addresses to the blockchain. While handling the verification request from the user, the blockchain pulls the required parts of data from IPFS and conducts the verification. In this work, similar to the approach adopted by many real-world applications that rely on inputs from the physical world, we employ a blockchain oracle service [30], [31], [32] for enabling smart contracts in blockchain to read data in IPFS.

2.2 Security Model

We assume that both the worker and the user are profit-driven due to the nature of the outsourcing computation scenario. In particular, the worker is interested in obtaining the reward with less or none computational cost. At the same time, the user would like to receive the correct result from the worker without any payment by claiming to get no result, or the obtained result is incorrect. On the other hand, based on the nature of the blockchain and IPFS, they are considered honest. That is, smart contracts deployed on the blockchain will be faithfully executed, and IPFS will correctly and promptly store and serve the data stored in it to smart contracts. Although some attacks, e.g., selfish mining, block withholding, and pool hopping, are targeting the underlying blockchain system, those attacks are beyond the scope of this paper and will be discussed in our future work.

Note that we mainly focus on how to efficiently obtain the fairness property for outsourcing polynomial computation in this paper. We do not consider the confidentiality of the input x, the polynomial $f(\cdot)$, and the computing result $f(x)$, which can be achieved by using (fully) homomorphic encryption schemes like those in references [4], [5], [6], [7], [8], [9]. Moreover, in the proposed scheme, we only consider errors intentionally introduced by workers to increase their revenue, and unintended errors will be considered in our future work.

2.3 Design Goal

Our design goal in this paper is to develop a new blockchain-based fair outsourcing polynomial computation scheme satisfying the following properties under the above system model and security model.

- *Fairness:* The main goal of this work is to design a fair outsourcing polynomial computation scheme. In other words, our proposal should guarantee that the user can obtain a valid computing result if he/she has paid, and the worker can get the reward if he/she follows the protocol correctly. The faithful and automatic execution of smart contracts in blockchains allows us to realize fairness if the computing result can be publicly verifiable. In this work, we use the sampling technique to gain public verifiability, and we give a game-theoretic analysis to show that the sampling method works well in our proposal.
- *Efficiency:* The primary motivation of outsourcing computation is the efficiency issue; hence, all the cost on each entity in our protocol should be as low as possible. 1) User side: Only a small computational cost on the user side is sufficient for outsourcing the computation of $f(x)$ properly and obtaining the value of $f(x)$ from the blockchain. The efficiency is also the main reason that the user is willing to outsource the computing task. 2) Worker side: The cost on the worker side should be as close as that of the original computing task. 3) Blockchain side: It is not free for the miners to do the result verification. No matter it is paid by the user or worker, there should be an upper bound for the cost. Ideally, only a (small) constant computational cost is enough for the result verification.
- *Functionality:* 1) Arbitrary polynomial support: It would be convenient and useful for engineering practice if the proposal can support all kinds of polynomials. No matter how many variables of the input, or how large the coefficients or orders of the polynomial are.[3] 2) Reusability: To further reduce resource usage of the user, he/she does not need to re-upload the polynomial for different inputs. Furthermore, it is only a small computational cost for the user to support polynomials with changeable coefficients.

3 Preliminaries

In this section, we briefly review the basic knowledge related to our proposal, including the blockchain, IPFS, and Horner's method.

3.1 Blockchain and Smart Contract

The blockchain technology, the famous essential tool for realizing cryptocurrencies, was invented in 2008 by Nakamoto [12]. It enables a group of distrusted entities to trust a public ledger maintained by these distrusted entities in a decentralized way. This building-trust functionality is mainly achieved by combining the underlying cryptographic primitives (such as hash functions and digital signatures) and consensus protocols (such as PoW [12] and DPoS [28]). Besides realizing cryptocurrencies, blockchain

3. Of course, it should be restricted by the limitation of the current information technology.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

technology is also the only viable platform to implement smart contracts. The smart contract concept was proposed by Szabo [33], who considered the smart contract as a program executed faithfully and automatically.

The smart contract in the blockchain usually proceeds as follows. First, the creator deploys a smart contract in the blockchain. Second, anyone who satisfies the predefined conditions in the smart contract can invoke it. Finally, the smart contract is executed by a set of participants (usually called miners) in the blockchain, and the internal state of the smart contract is also updated accordingly. Moreover, the underlying consensus protocol enforces that the miners will faithfully run the smart contract. As a result, the smart contract is executed automatically from the view of others (not the miners). In many blockchain systems [27], [29], the miners will not execute smart contracts for free, and they will charge the one who invokes the smart contract. The fee is usually termed Gas.

It is worth mentioning that not every blockchain system, such as Bitcoin, supports the smart contract. However, in this paper, we assume that the blockchain in our system supports it. For evaluating the performance of our proposal, we will use Ethereum [27] as the underlying blockchain system.

3.2 IPFS

Inter-Planetary File System (IPFS) is proposed by Benet [34] and developed by Protocol Labs [35]. It aims to provide users a resilient peer-to-peer file system for big file storage and sharing similar to BitTorrent. Meanwhile, it additionally supports content-addressing and version-controlling properties by using distributed hash tables and git, respectively. With the former property, users can obtain and verify the data easily and quickly. The latter property enables users to review the old version of the data.

Combination of Blockchain and IPFS. As blockchain is originally designed to be a public ledger to record transactions, most blockchain systems adopted many approaches to encourage the transaction size to be small to ensure the network performance. Consequently, it is either impossible or expensive to store big files directly in blockchain systems. Hence, as a distributed file system, IPFS has become a popular solution for data and resource storage in blockchain-based distributed applications (DApps) [36], [37], [38]. Specifically, the DApp can store data into IPFS and keep the corresponding addresses in the blockchain. Then, users can retrieve the corresponding data or files from IPFS with the addresses obtained from the DApp. On the other hand, when the DApp needs to read a block of data from IPFS, it can request through Oracle services. Currently, there are many centralized oracle services, e.g., Provable [30]; many works [39], [40] focus on decentralized oracle services; and several application-specific decentralized oracle services are operating, e.g., Chainlink [31], Origin Sport [41]. However, as we mainly focus on fair outsourcing polynomial computation, these works are beyond this paper's scope. Hence, in this paper, we consider a decentralized oracle service providing IPFS data for smart contracts.

3.3 Horner's Method

Horner's method [24] (or Horner's rule) is a polynomial evaluation method that aims to simplify the calculation of polynomials. It transfers a polynomial of degree n to n linear functions, and it could be expressed by:

$$\begin{aligned} f(x) &= a_0 + a_1 \times x + a_2 \times x^2 + \cdots + a_n \times x^n \\ &= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)))). \end{aligned} \tag{1}$$

Hence, it only requires n multiplications and n additions for computing $f(x)$, and the resulting time complexity is only $O(n)$. By contrast, the normal evaluation of $f(x)$ needs $n(n+1)/2$ multiplications and n additions, resulting in the time complexity $O(n^2)$. It is believed that Horner's method is the best way to compute polynomials of a high order, which is the main reason that we apply Horner's method in our proposal.

4 Our Proposed Outsourcing Polynomial Computation Schemes

In this section, to make a clear introduction to our protocol, we start from our basic scheme for regular univariate polynomials. Then, we show how to get better performance on sparse polynomials and how to support multivariate polynomials. The fairness of each scheme will be also analyzed in this section.

4.1 Basic Scheme for Univariate Polynomials

In this subsection, we will show how to outsource the computation of $f(x) = \sum_{i=0}^{n} a_i x^i$ in a fair way. We have fiv e phases in this basic scheme, namely, outsourcing, claiming, computing, verifying, and closing phases. In the first phase, the user uploads the specification of $f(x)$ to the blockchain. After that, the worker would claim the computing task and performs it in the second phase and third phase, respectively. In the fourth phase, the miners in the blockchain would verify the validity of the result output by the worker and decide the money transfer flow according to the verification result. If the outsourcing process does not go as well as expected, the user or the worker will execute the closing phase. The details of these phases can be found in the following.

4.1.1 The Description

Outsourcing phase: In this phase, as shown in Fig. 2, the user uploads the coefficients $\vec{a}$ (i.e., $\{a_0, a_1, \ldots, a_n\}$) of the polynomial $f(\cdot)$ and the input x to IPFS and deploys a smart contract in the blockchain based on a given template.

The smart contract contains a constructor and five functions, including functions PayReward, Claim, UploadResult, Verify, and Terminate as shown in Fig. 3. The function PayReward and the function Verify can only be invoked by the user, the function UploadResult can only be invoked by

1: **function** Outsource(x, $f(\cdot)$, w, d, t_w, t_u, ipfs, contract_tmpl)
2: $\quad\vec{a} = [\,]$
3: $\quad$**for** coefficient a_i in $f(\cdot)$ **do**
4: $\quad\quad\vec{a}$.push(a_i)
5: $\quad x\_addr$ = ipfs.upload(x)
6: $\quad\vec{a}\_addr$ = ipfs.upload($\vec{a}$)
7: $\quad$contract = contract_tmpl.deploy($\vec{a}\_addr, x\_addr$, w, d, t_w, t_u)
8: $\quad$contract.PayReward({value: w})
9: $\quad$**return**

Fig. 2. Outsourcing phase of the basic scheme.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

```
1: function PayReward() payable
2:     require(msg.sender == user_addr
3:             and msg.value == w)
4:
5: function Claim() payable
6:     require(worker_addr == ⊥
7:             and msg.value == d)
8:     worker_addr = msg.sender
9:
10: function UploadResult(addr)
11:     require(msg.sender == worker_addr
12:             and now < t_w)
13:     r_addr = addr
14:
15: function Verify(indexes, ipfs)
16:     require(msg.sender == user_addr)
17:     a = ipfs.download(a_addr)
18:     r = ipfs.download(r_addr)
19:     x = ipfs.download(x_addr)
20:     for i in indexes do
21:         if i == 0 then
22:             prev_r = a[n]
23:         else
24:             prev_r = r[i − 1]
25:         if r[i]! = a[n − i − 1] + x × prev_r then
26:             transfer(user_addr, d + w)
27:             return False
28:     transfer(worker_addr, d + w)
29:     return True
30:
31: function Terminate
32:     no_worker = (msg.sender == user_addr
33:             and worker_addr == ⊥)
34:     submit_timeout = (msg.sender == user_addr
35:             and now > t_w
36:             and r_addr == ⊥)
37:     verify_timeout = (msg.sender == worker_addr
38:             and now > t_u)
39:     require(no_worker or submit_timeout
40:             or verify_timeout)
41:     transfer(msg.sender, this.balance)
```

Fig. 3. Functions in the smart contract of the basic scheme.

the worker who has successfully invoked the function Claim, and the function Terminate can be invoked by the user or the worker. These restrictions on the invoker of the smart contract can be realized by the built-in variable `msg.sender` which is derived from the invoker's signature. Furthermore, the modifier payable after the function name shows that the function can be invoked with a certain amount of coins as shown at line 8 in Fig. 2, the value attached to an invocation can be obtained by the built-in variable, `msg.value`.

Moreover, the smart contract has the following fields: $\vec{a}$_addr, x_addr, t_w, t_u, w, d, user_addr, worker_addr, and r_addr. The meanings of these fields are listed in Table 2, and they are initialized as follows.

- The user directly sets the first six values. Note that the values of w and d will affect the fairness, which will be detailed in the fairness analysis later.
- The field user_addr is set as the built-in variable msg.sender mentioned above. Hence, this field stands for the address of the invoker of the constructor (the user).
- The rest of the fields including worker_addr and r_addr will be set to empty (denoted by $\perp$) and will be initialized in the next phases.

TABLE 2
Notations Used in the Smart Contract

Notations	Description
$\vec{a}$_addr	The address of $\vec{a}$ in IPFS.
x_addr	The address of x in IPFS when $f(x)$ is univariate.
user_addr	One of the addresses the user uses to receive coins.
worker_addr	One of the addresses the worker uses to receive coins.
w, d	The reward and deposit of the computing task, respectively.
t_w	The latest time the worker should return the computing result based on how long ago the computing task is recorded in the blockchain.
t_u	The latest time the user should launch the verifying process based on how long ago the computing result is recorded in the blockchain.

After deploying the smart contract, the user needs to transfer the reward w to it by invoking the function PayReward. This reward will be transferred to someone according to the conditions specified in the smart contract automatically. The other functions in the smart contract will be invoked in the next phases.

Claiming Phase. When a worker finds a computing task he/she is interested in from the blockchain, he/she would claim the task by invoking the function Claim in the corresponding smart contract with the amount d coins attached. The field worker_addr in the smart contract will be accordingly set to msg.sender. By doing so, the worker_addr will be the address of the worker who has successfully invoked this function, and the address cannot be forged since it is derived based on its signature as mentioned before. If the worker wants to unlink his/her two different computing tasks, he/she will use different addresses. Note that the function Claim in each smart contract could have only one successful invocation.

Computing Phase. Once the worker claims the computing task successfully, he/she would first download $\vec{a}$ and x from IPFS according to $\vec{a}$_addr and x_addr in the smart contract. After that, the worker computes $f(x)$ by Horner's method, particularly, calculating the recursion defined in:

$$\begin{aligned} r_0 &= a_n, \\ r_i &= a_{n-i} + x \times r_{i-1}. \end{aligned} \tag{2}$$

Before closing the phase, the worker should upload $\vec{r} = [r_0, r_1, \ldots, r_n]$ to IPFS and submit the corresponding address r_addr to the smart contract by invoking the function uploadResult through the address he/she used to claim this task. The high-level description of this phase can be found in Fig. 4.

Verifying Phase. When the result output by the worker is recorded in the blockchain, the user will receive an event. To respond it, an honest user will invoke the function Verify in the smart contract with several randomly selected indexes from $\{0, \ldots, n\}$. According to the function Verify, the miners download x, $\vec{a}$ and $\vec{r}$ from IPFS and check the validity of r_i's corresponding to selected indexes by using Eq. (2) (See line 20–line 27 in Fig. 3). As shown in lines 21-24, the verification algorithm directly gets r_{i-1} from worker's output or the original polynomial. Thus, the mismatch of r_i will not propagate, and the algorithm rejects if any of the

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

```
1: function COMPUTE(task, ipfs)
2:     contract = task.contract
3:     x = ipfs.download(contract.x_addr)
4:     a⃗ = ipfs.download(contract.a⃗_addr)
5:     r⃗ = [ ]
6:     prev_r = a⃗[n]
7:     for i in range(1 . . . n) do
8:         prev_r = a⃗[n − i] + x × prev_r
9:         r⃗.push(prev_r)
10:    r⃗_addr = ipfs.upload(r⃗)
11:    return contract.UPLOADRESULT(r⃗_addr)
```

Fig. 4. Computing phase in the basic scheme.

selected indexes satisfies $r_i \neq a_{n-i-1} + x \times r_{i-1}$. If the output cannot pass the verification, $\mathtt{w}$ and $\mathtt{d}$ in the smart contract would be transferred to $\mathtt{user\_addr}$; otherwise, the funds are transferred to the worker, and the user gets r_n as the result of polynomial $f(x)$.

Although the computation phase and the verification phase are the same for one iteration of Horner's method, the verification phase only verifies some steps. Therefore, the computation cost for the verification phase is less than that of the computation phase. Moreover, only verification-related data will be uploaded to the blockchain. Thus, the gas consumption can be significantly reduced compared to uploading all data to the blockchain. Also, as the coefficients and variables are submitted to IPFS as separated files, the user can use different coefficient addresses to enjoy the feature of dynamic coefficients.

Closing Phase. If no worker responded to the computing task, or the worker has not returned the computing result until t_w, the user would invoke the function TERMINATE to get funds $\mathtt{w}$ and $\mathtt{d}$ (if there exists). If the worker returned the computing result before t_w, while the user has not invoked the verifying phase until t_u, then the worker would invoke the function TERMINATE to get funds $\mathtt{w}$ and $\mathtt{d}$.

Note: At first glimpse, our scheme cannot guarantee that the result the user obtained is valid with 100 percent due to the use of the sampling technique. However, according to the game-theoretic analysis below, the worker will not deviate the steps in our scheme if he/she is profit-driven.

4.1.2 Fairness of the Basic Scheme

Our proposal should provide the following two properties: 1) If the worker executes correctly, then the user would pay, and vice versa; and 2) the party who deviated the underlying scheme would get punished. As rational parties, the action decisions of the worker and the user are based on their utilities. Based on the analysis of the utilities below, we have that the worker will execute the task faithfully, and the user will get the correct answer and pay the reward in time. In other words, fairness holds.

Worker's Utility. According to the description of the basic scheme, the utility of the worker can be expressed as follows.

$$u_w = \mathtt{w} \cdot (1 - \Pr[\mathtt{failure}]) - ((n - \iota) \cdot c_w + \mathtt{d} \cdot \Pr[\mathtt{failure}]),$$

where $\Pr[\mathtt{failure}]$ is the probability of verification failure, ι is the number of r_i's mismatching Eq. (2), and c_w is the cost for the worker to execute one multiplication operation.

According to Lemma 1, the worker's utility is $u_w \leq (c_w - \frac{\mathtt{w}+\mathtt{d}}{n}) \cdot \iota + \mathtt{w} - n \cdot c_w$. The inequality $n \cdot c_w < \mathtt{w} + \mathtt{d}$ usually holds since the reward $\mathtt{w}$ is larger than the cost $n \cdot c_w$ in most cases. Hence, the worker would get the largest utility when $\iota = 0$, i.e., the worker gets the largest utility when he/she follows our proposal.

On the other hand, Horner's method is considered as the most efficient method for univariate dense polynomials, and the vector $\vec{r}$ is the set of intermediate values of calculating $f(x)$ using Horner's method. In other words, when the worker generates $\vec{r}$ as specified, c_w reaches the smallest value.

Lemma 1. *The probability of verification failure increases as the number of r_i's to be verified increases. In particular, $\Pr[\mathtt{failure}] \geq \frac{\iota}{n}$. The equality holds when only one r_i in $\vec{r}$ is selected to be verified.*

Proof. Let k be the number of r_i's selected to be verified. Here, we only consider the case that $k + \iota \leq n$, since one of the invalid r_i's will be selected to be verified if $k + \iota > n$. The probability of verification failure can be expressed as

$$\begin{aligned}\Pr[\mathtt{failure}] &= 1 - \frac{(n-\iota)! \cdot (n-k)!}{(n-\iota-k)! \cdot n!} \\ &= 1 - \frac{(n-\iota)(n-\iota-1)\cdots(n-\iota-k+1)}{n(n-1)\cdots(n-k+1)}\end{aligned}$$

Let $F(k) = \frac{(n-\iota)(n-\iota-1)\cdots(n-\iota-k+1)}{n(n-1)\cdots(n-k+1)}$, we have that

$$\frac{F(k+1)}{F(k)} = \frac{n-\iota-k}{n-k} \leq 1.$$

From the above inequality, we have that $\Pr[\mathtt{failure}]$ increases as k increases. Furthermore, when $k = 1$, $F(k)$ reaches the maximum value $F(1) = 1 - \frac{\iota}{n} \leq 1$, and $\Pr[\mathtt{failure}]$ gets the minimum value $\frac{\iota}{n}$. Note that the cheating action cannot be detected when $k < 1$; hence, k must be 1 at least. □

As a result, rational workers will calculate polynomials exactly following the scheme, since cheating will not increase the utility.

User's Utility. Since the miners execute the verification, the user cannot refuse to pay by pretending that the answer is incorrect or it is not received. As a result, the user will follow the proposed scheme. In this case, his/her utility can be expressed as follows.

$$\begin{aligned}u_u &= \mathtt{d} \cdot \Pr[\mathtt{failure}] - (k \cdot c_u + \mathtt{w} \cdot (1 - \Pr[\mathtt{failure}])) \\ &= \mathtt{d} - (\mathtt{d} + \mathtt{w}) \cdot F(k) - k \cdot c_u,\end{aligned}$$

where c_u is the cost for the user to pay the miners to calculate one multiplication, like the gas in Ethereum. That is, the overall cost for the verification phase is linear to the number of multiplication operations in this phase.

According to the analysis in the proof of Lemma 1, $F(k) \leq 1$ and $F(k)$ will decrease when k increases. Furthermore, $(\mathtt{d} + \mathtt{w})$ is usually larger than c_u. Hence, u_u will first increase and then decrease when k increases from 1 to n, and it will reach the largest value when k satisfies the following inequalities.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

```
1: function OUTSOURCE(x, f(·), w, d, t_w, t_u, ipfs, contract_tmpl)
2:     a⃗ = [ ]
3:     Rewrite f(·) as f(x) = Σ_{i=0}^{m} a_{c_i} · x^{c_i}
4:     for coefficient a_{c_i} in f(·) do
5:         Set ā_i = a_{c_i}
6:         Compute d_i = c_{i+1} − c_i
7:         a⃗.push((ā_i, d_i))
8:     x_addr = ipfs.upload(x)
9:     a⃗_addr = ipfs.upload(a⃗)
10:    contract = contract_tmpl.deploy(a⃗_addr, x_addr, w, d, t_w, t_u)
11:    contract.PAYREWARD({value: w})
12:    return
```

Fig. 5. Outsourcing phase for the sparse polynomial.

$$\begin{cases} F(k-1) - F(k) < \frac{c_u}{\mathtt{d}+\mathtt{w}}; \\ F(k) - F(k+1) \leq \frac{c_u}{\mathtt{d}+\mathtt{w}}. \end{cases}$$

Since the user does not know the value of ι, he/she cannot choose the optimal k for largest u_u. Fortunately, as analyzed for the worker's utility, $k = 1$ is enough for the verification if the worker is rational.

4.2 Enhancing Performance for Sparse Polynomials

In the previous subsection, we proposed an outsourced univariate polynomial computation scheme based on Horner's method and illustrated that the rational worker and user would not deviate from the scheme. However, the solution is not quite efficient or effective as expected for sparse polynomials. For instance, we have a sparse polynomial $f(x) = a_n x^n$, and there exists only one r_i. To verify the result, the miners should compute $f(x) = a_n x^n$ again, which is not acceptable for the blockchain or the user. To solve this problem, we make use of the binary exponentiation technique, which can provide enough r_i's to be sampled. In the rest of this subsection, we will show how we extend our basic scheme to get better performance on evaluating sparse polynomials and analyze the fairness of the resulting scheme. Note that, we use the gray background color to highlight the differences with that in the basic scheme in Figs. 5, 6, and 8. The claiming phase and closing phase are the same as those in the basic scheme, and we omit them in the description.

4.2.1 The Description

Outsourcing phase: To get better performance, we need to modify the data structure of the vector $\vec{a}$. In particular, the element in $\vec{a}$ changes $(\bar{a}_i, d_i)$ as follows.

```
1: function VERIFY(indexes, ipfs)
2:     require(msg.sender == user_addr)
3:     a⃗ = ipfs.download(a⃗_addr)
4:     r⃗ = ipfs.download(r⃗_addr)
5:     x = ipfs.download(x_addr)
6:     for i in indexes do
7:         if i == 0 then
8:             prev_r = a⃗[m].left
9:         else
10:            prev_r = r⃗[i − 1]
11:        prev_r = a⃗[i].left + GETPOWER(L, x, a⃗[i].right) × prev_r
12:        if r⃗[i]! = prev_r then
13:            transfer(user_addr, d + w)
14:            return False
15:    transfer(worker_addr, d + w)
16:    return True
```

Fig. 6. The function VERIFY for the sparse polynomial.

```
1: function GETPOWER(L, x, p)
2:     if p == 1 then
3:         return x
4:     if L.contains(p) then
5:         return L.get(p)
6:     Find (p′, xp′) ∈ L that max({p′|p − p′ > 0})
7:     if (p′, xp′) exists then
8:         xp = xp′ × GETPOWER(L, x, p − p′)
9:         L.put(p, xp)
10:        return xp
11:    else
12:        xp = GETPOWER(L, x, ⌊p/2⌋)
13:        xp = xp × xp
14:        if p&1 == 1 then
15:            xp = xp × x
16:            L.put(p − 1, xp)
17:        L.put(p, xp)
18:        return xp
```

Fig. 7. Function for evaluating the power of x.

Assume the outsourced sparse univariate polynomial is $f(x) = \sum_{i=0}^{n} a_i \cdot x^i$. We rewrite it as $f(x) = \sum_{i=0}^{m} a_{c_i} \cdot x^{c_i}$ with $c_0 = 0$, and $a_{c_i} \neq 0$ and $c_i \in [1, n]$ for $i \in [1, m]$. Without loss of generalization, we assume that $c_i < c_{i+1}$ always holds. Note that $a_{c_0} (= a_0)$ could be zero and $c_m = n$. The pairs $(\bar{a}_i, d_i)$ can be computed as follows.

$$\bar{a}_i = a_{c_i},\ d_i = c_{i+1} - c_i \text{ for } i \in [0, m],$$

where $c_{m+1} = n$.

For clarification, let us take the following toy example. Assume we have polynomial $2x^3 + 5x^{23}$, then $\bar{a}_0 = a_0 = 0$, $d_0 = c_1 - c_0 = 3 - 0 = 3$, $\bar{a}_1 = a_{c_1} = 2$, $d_1 = c_2 - c_1 = 23 - 3 = 20$, $\bar{a}_2 = a_{c_2} = 5$, and $d_2 = c_3 - c_2 = 23 - 23 = 0$.

The rest part of this phase is the same as that in the basic scheme. We give a high-level description in Fig. 5. Note that the smart contract built in this phase is almost the same as that in the basic scheme, the only difference is the function VERIFY that is given in Fig. 6. The description of function GETPOWER used in Fig. 7 will be given in the computing phase.

Computing Phase: With the new vector $\vec{a} = [(\bar{a}_0, d_0), (\bar{a}_1, d_1), \ldots, (\bar{a}_m, d_m)]$, the recursion in Horner's method is accordingly evolved as follows.

$$\begin{aligned} r_0 &= \bar{a}_m \\ r_i &= \bar{a}_{m-i} + x^{d_{m-i}} \cdot r_{i-1} \end{aligned} \tag{3}$$

Note that the worker can compute $x^{d_{m-i}}$ by any method he/she wants. Here, we give a method called GETPOWER that is more

```
1: function COMPUTE(task, ipfs)
2:     contract = task.contract
3:     x = ipfs.download(contract.x_addr)
4:     a⃗ = ipfs.download(contract.a⃗_addr)
5:     r⃗ = [ ]
6:     L = { }
7:     prev_r = a⃗[m].left
8:     for i in range(0 . . . m) do
9:         prev_r = a⃗[m − i].left + GETPOWER(L, x, a⃗[m − i].right)
10:            +prev_r
11:        r⃗.push(prev_r)
12:    r⃗_addr = ipfs.upload(r⃗)
13:    return contract.UPPLOADRESULT(r⃗_addr)
```

Fig. 8. Computing phase for the sparse polynomial.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

efficient than the one simply applying the binary exponentiation technique repeatedly for computing many $x^{d_{m-i}}$'s. The main idea in the function GETPOWER is to make use of the intermediate values generated from the process of computing previous $x^{d_{m-i}}$'s. Assume we have the intermediate values $(x^{p_0}, x^{p_1}, \ldots, x^{p_j})$, and we aim to compute x^p. We first find p' that is the biggest value not larger than p in $\{p_0, p_1, \ldots, p_j\}$. Then, we have that $x^p = x^{p'} \cdot x^{p-p'}$. Since we already have $x^{p'}$, we only need to compute $x^{p-p'}$ to obtain the value of x^p. It is easy to see that $x^{p-p'}$ can be further computed as above. The details of the function GETPOWER can be found in Fig. 7, where L is the list used to record the previous intermediate values.

For clarification, let us take the following toy example. Assume we need to compute x^{13} and x^8. If we just repeatedly use the binary exponentiation technique, we need to compute the intermediate values $(x^2, x^3, x^6, x^{12}, x^{13})$ and (x^2, x^4, x^8), respectively. However, by using our GETPOWER function, x^8 can be computed by using $x^2 \cdot x^6$ if x^{13} was computed before, or x^{13} can be computed by using $x^8 \cdot x^4 \cdot x$ if x^8 was computed before.

The rest part of this phase is the same as that in the basic scheme. A high-level description of this phase is given in Fig. 8.

Verifying Phase. Similar with the one in the basic scheme, the user invokes the function VERIFY (as shown in Fig. 6) in the smart contract with several randomly selected indexes from $\{0, \ldots, m\}$. According to the function VERIFY, the miners would download x, $\vec{\boldsymbol{a}}$, $\vec{\boldsymbol{r}}$ from IPFS, and check the validity of r_i's corresponding to selected indexes by using Eq. (3). The rest of this phase is the same with the one in the basic scheme.

4.2.2 Fairness in the Scheme for Sparse Polynomials

With the similar analysis as that in the basic scheme, we have the following result.

- The worker's utility is $u_w \leq \left(c'_w - \frac{\mathtt{w}+\mathtt{d}}{m}\right) \cdot \iota + \mathtt{w} - n \cdot c'_w$, where c'_w is the worker's cost for computing one r_i. The inequality $m \cdot c'_w < \mathtt{w} + \mathtt{d}$ usually holds since the reward $\mathtt{w}$ is larger than the cost $m \cdot c'_w$ in most cases. Hence, the worker would get the largest utility when $\iota = 0$.
- The user's utility $u_u = \mathtt{d} - (\mathtt{d} + \mathtt{w}) \cdot F'(k) - k \cdot c'_u$ will reach the largest value when k satisfies the following inequalities,

$$\begin{cases} F'(k-1) - F'(k) < \frac{c'_u}{\mathtt{d}+\mathtt{w}}, \\ F'(k) - F'(k+1) \leq \frac{c'_u}{\mathtt{d}+\mathtt{w}}, \end{cases}$$

  where $F'(k) = \frac{(m-\iota)! \cdot (m-k)!}{m! \cdot (m-\iota-k)!}$, and c'_u is the user's average cost for employing miners to compute one r_i. Note that, for different polynomials, c'_u varies according to their degrees and numbers of terms.

Hence, the worker will execute the task faithfully, and the user will get the correct answer and pay the reward in time. In other words, the property of fairness holds.

4.3 Supporting Multivariate Polynomials

Now, we can present our final scheme that additionally supports multivariate polynomials. In the final scheme, the worker first transforms a general multivariate polynomial into Horner's method format, and then it follows the previous methods in Section 4.2 to finish the outsourced computation. Similarly, we also use the gray background color to highlight the differences in Figs. 9, 10, and 13. Moreover, similar to the scheme for the sparse polynomial, the claiming phase and closing phase are the same as those in the basic scheme, and we omit them in the description.

```
1: function OUTSOURCE(x⃗, f(·), w, d, t_w, t_u, ipfs, contract_tmpl)
2:     a⃗ = [ ]
3:     for A_i in f(·)p do
4:         a⃗.push((a_i, {(j, p_ij)}_{j∈[1,ñ], p_ij≠0}))
5:     x⃗_addr = ipfs.upload(x⃗)
6:     a⃗_addr = ipfs.upload(a⃗)
7:     contract = contract_tmpl.deploy(a⃗_addr, x⃗_addr, w, d, t_w, t_u)
8:     contract.PayReward({value: w})
9:     return
```

Fig. 9. Outsourcing phase for the multivariate polynomial.

4.3.1 Description

Outsourcing Phase. Similar to the previous schemes, the user needs to upload the data related to the underlying multivariate polynomial, i.e., the vector $\vec{\boldsymbol{a}}$ and the input $\vec{\boldsymbol{x}} = [x_1, \ldots, x_{\tilde{n}}]$. The vector $\vec{\boldsymbol{a}}$ is different from the one in the previous schemes and obtained as follows.

Let $f(\vec{\boldsymbol{x}}) = \sum_{i=1}^{N} A_i$ be a multivariate polynomial, where $\vec{\boldsymbol{x}} = [x_1, \ldots, x_{\tilde{n}}]$, $A_i = a_i \cdot \prod_{j=1}^{\tilde{n}} x_j^{p_{ij}}$, $a_i \neq 0$, and $0 \leq p_{ij} \leq n$ for $1 \leq j \leq \tilde{n}$ and $1 \leq i \leq N$. For each A_i, we have one element in $\vec{\boldsymbol{a}}$, i.e., $(a_i, \{(j, p_{ij})\}_{\substack{j \in [1,\tilde{n}] \\ p_{ij} \neq 0}})$. The rest of this phase is almost the same as the one for the basic scheme (see Fig. 9), except that the VERIFY function is changed as shown in Fig. 10. The explanations on the input nodes in Fig. 10 will be given in the verifying phase.

Computing Phase. In this phase, the worker also needs to compute a vector $\vec{\boldsymbol{r}}$ for the underlying multivariate polynomial $f(x_1, \ldots, x_{\tilde{n}}) = \sum_{i=1}^{N} (a_i \cdot \prod_{j=1}^{\tilde{n}} x_j^{p_{ij}})$. First, the worker builds a binary tree BTree corresponding to the polynomial as follows.

- Find the most-occurring variable x_ℓ, and denote $\mathcal{A}'$ as the set of A_i's containing x_ℓ. If there exist several x_ℓ's, then compute the biggest p_ℓ for each x_ℓ that $\forall A_i \in \mathcal{A}'$ contains $x_\ell^{p_\ell}$. The final x_ℓ is the one whose

```
1: function VERIFY(nodes, ipfs)
2:     require(msg.sender == user_addr)
3:     a⃗ = ipfs.download(a⃗_addr)
4:     r⃗ = ipfs.download(r⃗_addr)
5:     x⃗ = ipfs.download(x⃗_addr)
6:     for node in nodes do
7:         if node.left == 0 then
8:             tmp = node.right
9:         else
10:            tmp = node.rightchild.r +
11:                x_{node.left}^{node.right} × node.leftchild.r
12:        if tmp_p! = node.r then
13:            transfer(user_addr, d + w)
14:            return False
15:    transfer(worker_addr, d + w)
16:    return True
```

Fig. 10. The function VERIFY for the multivariate polynomial.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

1: **function** BUILDBTREE($f(x_1, \cdots, x_{\tilde{n}})$)
2: **if** All A_i's are constants **then**
3: Record $(0, \sum_{i=1}^{N} A_i)$ as the root node
4: **else**
5: Find the most-occurring variable x_ℓ
6: Compute the biggest p_ℓ that $\forall A_i \in \mathcal{A}$ contains
7: $x_\ell^{p_\ell}$
8: Record (ℓ, p_ℓ) as the root node
9: Record BUILDBTREE($\prod_{A_i \in \mathcal{A}'} \frac{A_i}{x_\ell^{p_\ell}}$) as the
10: left-child substree
11: Record BUILDBTREE($\prod_{A_i \in \mathcal{A}'/\mathcal{A}} A_i$) as the
12: right-child substree
13: **return** the built binary tree

Fig. 11. Building a binary tree for a multivariate polynomial.

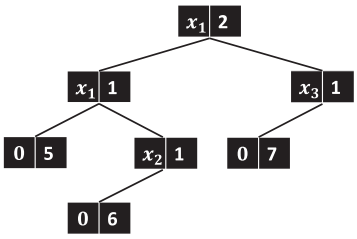


Fig. 12. A binary tree for polynomial $f(x_1, x_2, x_3) = 5x_1^3 + 6x_1^2x_2 + 7x_3$.

p_ℓ is the largest. If there still exist several x_ℓ's, then choose any of them as the final x_ℓ.

- Compute the biggest p_ℓ that $\forall A_i \in \mathcal{A}'$ contains $x_\ell^{p_\ell}$. This step will be skipped if p_ℓ was computed before.
- Record (ℓ, p_ℓ) as the root node of the current tree.
- Repeat the above steps to build the left-child subtree and right-child subtree of node (ℓ, p_ℓ) with polynomials $\prod_{A_i \in \mathcal{A}'} A_i / x_\ell^{p_\ell}$ and $\prod_{A_i \in \mathcal{A}/\mathcal{A}'} A_i$, respectively.

A high-level description of the above steps is given in Figs. 11 and 12 gives a toy example of a polynomial $f(x_1, x_2, x_3) = 5x_1^3 + 6x_1^2x_2 + 7x_3 = x_1^2(5x_1 + 6x_2) + 7x_3$.

After obtaining the binary tree, the worker computes r for every node in the binary tree starting from leaf nodes by using Eq. (4) (the process can be also found in lines 6–12 in Fig. 13).

$$\texttt{node}.r = \texttt{node.rightchild}.r + \texttt{node}.x_\ell^{\texttt{node}.p_\ell} \times \texttt{node.leftchild}.r. \tag{4}$$

Then the worker records r's into $\vec{r}$. The rest part of this phase is the same as that in the basic scheme. A high-level description can be found in Fig. 13.

1: **function** COMPUTE(task, ipfs)
2: contract = task.contract
3: $\vec{x}$ = ipfs.download(contract.$\vec{x}$_addr)
4: $\vec{a}$ = ipfs.download(contract.$\vec{a}$_addr)
5: Call BUILDBTREE to obtain BTree
6: $\vec{r} = []$
7: **for** node in BTree **do**
8: **if** node.left == 0 **then**
9: node.r = node.right
10: **else**
11: node.r = node.rightchild.r+
12: $x_{\text{node.left}}^{\text{node.right}} \times$ node.leftchild.r
13: $\vec{r}$_addr = ipfs.upload($\vec{r}$)
14: **return** contract.uploadResult($\vec{r}$_addr)

Fig. 13. Computing phase for the multivariate polynomial.

Verifying Phase: Similar to the one in the basic scheme, the user invokes the function VERIFY (as shown in Fig. 10) in the smart contract with several randomly selected nodes from the underlying BTree. According to the function VERIFY, the miners would download $\vec{x}$, $\vec{a}$, $\vec{r}$ from IPFS, and check the validity of r_i's corresponding to selected nodes by using Eq. (4). The rest of this phase is the same as that of the basic scheme.

4.3.2 Fairness in the Scheme

Since the scheme has a similar procedure for computing and verifying, the probability of verification failure will not change. Hence, the utilities and actions of both parties are similar to the analysis in the previous subsection. As a result, the fairness will not lose in the multivariate setting.

5 ANALYSIS OF OUR PROPOSAL

As shown in the previous section, our proposal holds the fairness. In this section, we will examine the functionality of our proposal one by one as listed in Section 2.3, and show the efficiency by experiments.

5.1 Functionality

In this subsection, we analyze the functionality of our proposed scheme in the following two aspects, namely, arbitrary polynomial support and reusability.

Arbitrary Polynomial Support. Based on the description of our last scheme in Section 4.3, to outsource a polynomial, the user only needs to upload $\vec{a}$ and $\vec{x}$, i.e., the vectors of terms and values of variables. Then, the worker needs to compute the polynomial according to our modified Horner's method. Furthermore, the miners can verify the result by using the sampling technique. Hence, our proposal can support arbitrary polynomials.

Reusability. In our final scheme, as it points to the polynomial and the input separately, the reusability consists of two parts, namely, reusing inputs and reusing polynomials. To evaluate the same polynomial on different inputs, the user can directly reuse the address to the original $\vec{a}$ and only upload the new inputs $\vec{x}$ to IPFS. On the other hand, to reuse the input, the only thing the user needs to do is to upload the new $\vec{a}$ to replace the old one. After that, the worker can evaluate the new polynomial as before.

5.2 Efficiency

We implemented our scheme in Java and Solidity.[4] First, to demonstrate its actual executing time of the computing phase and the verifying phase, we test the Java implementation with a Debian 10 platform equipped with an Intel (R) Xeon (R) CPU and 26GB RAM. Then, to evaluate the gas consumption of the verification phase, we test the Solidity implementation on Rinkeby Ethereum Testnet [25], and we simulate an oracle service to feed data for the smart contract. Note that, in real-world systems, the values of polynomials are usually bounded. Therefore, in the following experiments, we randomly generate a prime modulus for each polynomial and show the performance of our

4. The source code of our implementation can be found at https://github.com/guanyg/polyOutsourcing

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

proposed scheme with different bit-lengths of moduli $b = 1024, 2048, 4096, 8192$.

5.2.1 Univariate Polynomials

In the computing phase, upon receiving the array of the $n+1$ coefficients in the polynomial and the value of the variable, the worker computes n multiplications according to Horner's method. In the verifying phase, the miners verify the result according to the position chosen by the user. As analyzed in Section 4, one position is enough for deterring the rational worker from deviating. Therefore, the computational complexities for computing and verifying the univariate polynomials are $O(n)$ and $O(1)$, respectively. As illustrated in Fig. 14a, the computation time increases as the degree of the polynomials increases, and the average time used for computing and verification increases with the bit-length of moduli b. Also, the verification phase is very efficient. Specifically, the average time for the verifying phase remains 4.3 μs if only one position is chosen. As shown in Fig. 14b, the gas consumption for verifying univariate polynomials increases with the degree of polynomials and the bit-length of moduli, and for the case where bit-length of moduli $b = 8192$, the gas consumption is lower than 400,000. As the current gas price in the Ethereum mainnet is 11 gwei, the average cost for running the verification is less than 0.0044 ETH (about 0.88 USD).

5.2.2 Sparse Polynomials

For sparse polynomials, the time consumption in different phases depends on not only their degrees but also the number of non-zero terms in them. Thus, we ran experiments with varying degrees (i.e., 100, 1000, and 10000) and different non-zero terms. As shown in Figs. 14c and 14d, the average time for the computing phase increases with the degree n and the number of non-zero terms, but the time consumption for outsourcing polynomials decreases as the number of terms increases. This is mainly because that as the number of terms increase, the cost for computing the power of variables becomes lower. Also, Fig. 14e shows that the average gas consumption for verifying sparse univariate polynomials decreases as the number of terms increases, and it increases with the bit-length of moduli b, and it is lower than 300,000 (about 0.66 USD in the current Ethereum mainnet).

5.2.3 Multivariate Polynomials

To evaluate the performance of our multivariate polynomials scheme, we generated a series of polynomials with different degrees, different numbers of non-zero terms, different numbers of variables, and different bit-lengths of moduli. Since the user uploads the multivariate polynomial in the outsourcing phase, here we focused on the performance of the computing phase and the verifying phase. As shown in Figs. 14f, 14g, 14h, 14j and 14k, the computation time increases with the number of variables, while the verification time decreases as the number of variables increases. This is mainly because that, as the number of variables increases, the cost for constructing the binary tree increases while the cost for computing the power of variables decreases. Also, Figs. 14f and 14k show that the computation time for multivariate polynomials increases with the degree of polynomials and the number of terms. As illustrated in Figs. 14h and 14k, the average gas consumption for verifying multivariate polynomials is lower than 550,000 (about 1.21 USD in the current Ethereum mainnet).

5.2.4 Performance Comparison

Currently, there are many works focusing on verifiable outsourcing polynomial computation as detailed in Section 6, and these schemes usually spend heavy computational costs to achieve verifiability of computation results. Different from the existing schemes, we build our fair outsourcing polynomial computation scheme based on the rationality assumption, i.e., all participants are rational actors. Thereby, our proposed scheme can deter rational workers from submitting incorrect results and enforce the fairness between users and workers. To further demonstrate the efficiency of our proposed scheme, we compare our scheme with two schemes, namely, Song *et al.*'s scheme [8] (improved by Wang *et al.* [42]) and Zhang *et al.*'s scheme [26]. In specific, we implement our scheme and Song *et al.*'s scheme, and compare them with the open-source implementation[5] of Zhang *et al.*'s scheme [26]. Since the implementation of Zhang *et al.*'s scheme only supports computation over 64 bits integers, we mainly compare the computation and proof generation time ("Proof" in Fig. 14l) and verification time ("Verification" in Fig. 14l) of the three schemes handling polynomials with varied number of variables. In these polynomials, the bit-lengths of the coefficients and variables are respectively chosen to be 16 and 4. As shown in Fig. 14l, our proposed scheme is much efficient than the compared schemes.

6 Related Work

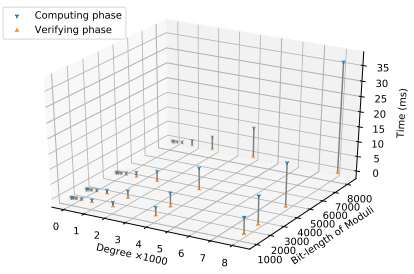
Since the polynomial computation is very common in many applications, many outsourcing polynomial computation schemes have been proposed. The existing schemes can be roughly divided into three types, namely crypto-based [5], [6], [7], [8], [43], [44], [45], [46], [47], [48], replica-based [10], [49], [50], and blockchain-based schemes [13], [14], [51], [52], [53]. However, none of them can outsource arbitrary polynomials fairly and efficiently.

6.1 Crypto-Based Outsourcing Polynomial Computation

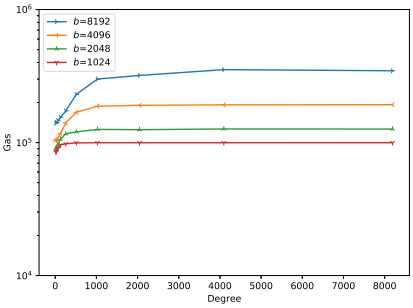
In the cryptography community, outsourcing polynomial computation is studied under the name of verifiable computation. The first formal security model of verifiable computation was proposed by Gennaro *et al.* [4] at CRYPTO 2010, and a concrete scheme was also present in the same work. Based on Gennaro *et al.*'s model, Benabbas *et al.* [43] proposed another outsourcing polynomial scheme by using the CFE-PRF (Closed-form Efficiency Pseudorandom Functions). However, their scheme only supports private verification as [4]. To solve this problem, Fiore *et al.* [5] gave a new outsourcing polynomial computation scheme based on

5. https://github.com/sunblaze-ucb/Virgo

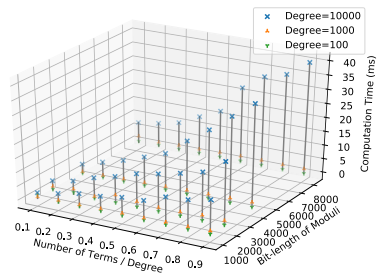
Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.



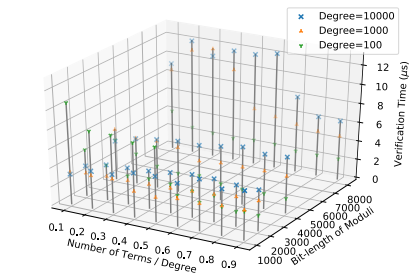
(a) The computation and verification time for univariate polynomials.



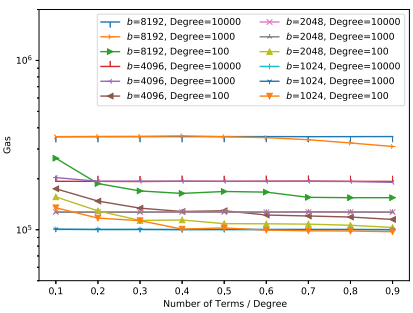
(b) The gas consumption for verifying univariate polynomials.



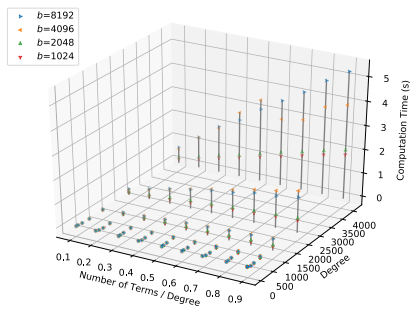
(c) The computation time for sparse univariate polynomials.



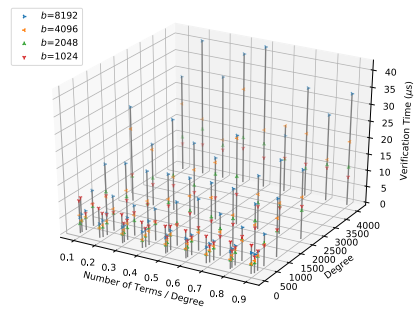
(d) The verification time for sparse univariate polynomials.



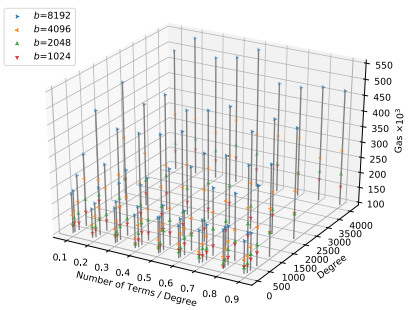
(e) The gas consumption for verifying sparse univariate polynomials.



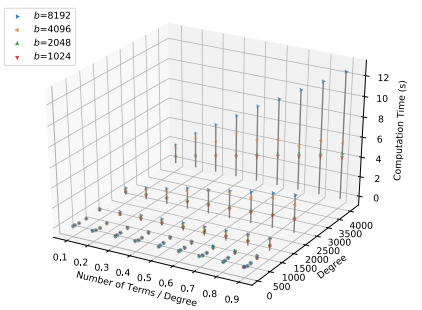
(f) The computation time for multivariate polynomials with 2 variables.



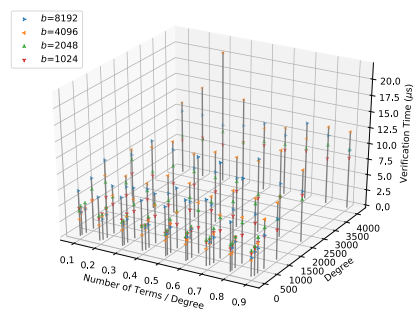
(g) The verification time for multivariate polynomials with 2 variables.



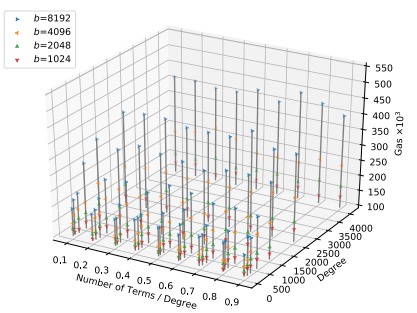
(h) The gas consumption for verifying multivariate polynomials with 2 variables.



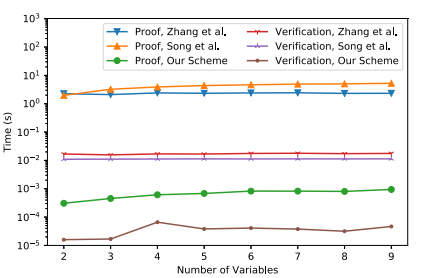
(i) The computation time for multivariate polynomials with 4 variables.



(j) The verification time for multivariate polynomials with 4 variables.



(k) The gas consumption for verifying multivariate polynomials with 4 variables.



(l) The proof and verification time comparison between our proposed scheme, Zhang et al.'s scheme [26] and Song et al.'s scheme [8].

Fig. 14. The experimental results of our schemes.

the bilinear map, while the verification of the resulting scheme is not efficient as expected. Later on, Catalano *et al.* [44], Backes *et al.* [45], Fiore *et al.* [46], and Zhang *et al.* [47] present new schemes with more efficient verification by using different techniques, such as homomorphic hash function and homomorphic message authentic codes.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

Nevertheless, all the above schemes can only outsource some special polynomials. At EUROCRYPT 2011, Boneh *et al.* [54] constructed a homomorphic signature scheme which supported multivariate polynomials of constant degree. Following this, Catalano *et al.* [55] proposed a more efficient scheme with the help of graded encoding schemes, and it obtained a higher security level. By using the arithmetic circuit, Parno *et al.* [56] built a system named Pinocchio, but this system cannot offer public verifiability. Papamanthou *et al.* [6] proposed the signatures of correct computation model (SCC), the worker would produce a succinct signature to vouch for the correctness of the result, and it could be verified efficiently. This protocol supports the dynamic update of coefficients, but it cannot support arbitrary polynomial. Zhang *et al.* [48] proposed batch verifiable computation of polynomials, but it cannot support public verifiability. Sun *et al.* [57] solved this problem. Elkhiyaoui *et al.* [7] proposed a publicly verifiable computation of polynomial based on the concept of the euclidean division. However, it is only suitable for univariate polynomials. Song *et al.* [8], amended by Want *et al.* [42], used homomorphic verifiable computation tags to design an outsourcing polynomial computation program. The efficiency of this program is independent of the size of input data or the highest degree of the polynomial. However, this scheme also has restrictions on the type of polynomials. Zhang *et al.* [58] employed a linearly homomorphic private-key encryption scheme to achieve public verifiability. However, since polynomials are represented as matrices, it is inefficient for the scheme to handle sparse and multivariate polynomials. Based on lightweight cryptographic primitives, Zhang *et al.* [26] proposed a succinct zero knowledge argument scheme for layered arithmetic circuits, which can support polynomial delegation and other arithmetic circuits.

Although crypto-based schemes are accurate and of high-security levels, they suffer from one of the following disadvantages, at least. 1) Most of the crypto-based schemes focus on how the user verifies the result from the worker, and there is a process for the worker to ensure that the worker can get the promised reward. In other words, fairness cannot be guaranteed. 2) The proof generation costs in most of the crypto-based schemes are 3-6 orders of magnitude higher than the original task [8]. 3) Most of the crypto-based schemes cannot support general polynomials, and they are always with the constraint of coefficients, the number of variants, or the degree of the polynomials.

Unlike the crypto-based outsourcing polynomial computation schemes, our proposed scheme focuses on enforcing result correctness for rational workers. Based on our game-theoretic analysis, when the workers are rational, which is reasonable in most applications, they will not deviate from the protocol by submitting incorrect computational results. Therefore, our proposed scheme can achieve a similar accuracy and a trade-off between security and efficiency without employing cryptographic operations compared with the crypto-based schemes. Moreover, comparing with most of these schemes, the proposed scheme has the feature of dynamic coefficients as the variables and coefficients are separately uploaded to IPFS. Furthermore, different from our proposed scheme, the existing crypto-based schemes cannot support fairness because they are not integrated with payment channel and cannot programmatically transfer funds based on the results' correctness.

6.2 Replica-Based Outsourcing Computation

The basic idea of the replica-based schemes is to outsource a computing task to two or more workers, and the workers compute independently. The user will crosscheck the results after all servers have returned the results so that the user could verify the results quickly. Based on refereed games [59], Canetti *et al.* [49] constructed an interactive verifiable computation scheme on multiple servers. Based on this scheme, they continually improved efficiency and security in [50] and [10]. All the schemes work correctly only if the workers do not collude with each other, while this assumption would not always hold in reality. Furthermore, like the crypto-based schemes, replica-based schemes do not consider the workers' profits.

6.3 Blockchain-Based Outsourcing Computation

Blockchain is a technology allowing entities to establish trust relationships even if they distrusted each other before. Hence, it is natural to apply blockchain in outsourcing computation, and many attempts have happened in industry and academia.

In industry, Golem [60], SONM [61] and iExec [62] are three notable projects. Golem can only provide result verifiability but it cannot guarantee that the worker will get the promised reward. SONM obtains fairness according to the reputation system. iExec uses proof of contribution, majority voting, and reputation score to ensure fairness. In other words, SONM and IExec cannot support result verifiability.

In academia, Kumaresan *et al.* [13] proposed a model to motivate correct computing in the Bitcoin network. In their system, the worker should pay a deposit that would be lost if the result cannot pass the verification. Their scheme mainly focused on the timely delivery of the results and the fairness of the payment. Campanelli *et al.* [51] realized the zero-knowledge payment of services based on blockchain, but their scheme lacks design details, and its efficiency is not so good as expected. Based on game theory and smart contract, Dong *et al.* [14] proposed an efficient verifiable outsourcing computing solution. However, their scheme only works when the workers do not collude with each other. This assumption is a little bit strong in the blockchain system. For example, more than 15 percent of mining power may come from the same mining pool in Bitcoin. In other words, there is more than 15 percent chance that the two workers are from the same organization. Huang *et al.* [52] proposed a blockchain-based outsourcing computation scheme based on commitment-based sampling technology, but it requires a trusted third-party. Krol *et al.* [53] proposed an efficient and secure blockchain-based outsourcing computation solution by using trusted hardware that would increase the user's cost. Lin *et al.* [63] studied how to use blockchain to secure outsourcing bilinear pairings. Zhang *et al.* [64] employed a challenge-and-proof manner to build a fair payment framework for outsourcing service, but it did not discuss the approach of constructing correctness proof for specific computational tasks. Cui *et al.* [65] proposed an outsourced decryption scheme for a functional encryption scheme, in

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

which a decrypted result is first verified by the user and further verified by miners if the user rejects the result. It is fair to say that none of the existing blockchain-based schemes can outsource all kinds of polynomials fairly and efficiently.

7 Conclusion

By using Horner's method and the blockchain, we have proposed an outsourcing polynomial computation scheme supporting arbitrary polynomials and dynamic coefficients. Horner's method allows the worker to perform polynomial computation as efficient as possible. The blockchain can guarantee fairness between the user and worker when the miners can verify the result from the worker. We obtained the public verifiability of the result by using the sampling technique. With the assumption that both user and worker are rational, our game-theoretic analysis shows that our verification process works well. By running the extensive experiments, our proposal is indicated as efficient in terms of computational cost.

Acknowledgments

This work was supported by the Natural Science Foundation of Zhejiang Province under Grant LZ18F020003, the National Natural Science Foundation of China under Grant U1709217, NSERC Discovery under Grants 04009, and LMCRF-S-2020-03.

References

[1] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, 2019.
[2] Y. Guan, J. Shao, G. Wei, and M. Xie, "Data security and privacy in fog computing," *IEEE Netw.*, vol. 32, no. 5, pp. 106–111, 2018.
[3] J. Shao and G. Wei, "Secure outsourced computation in connected vehicular cloud computing," *IEEE Netw.*, vol. 32, no. 3, pp. 36–41, May/Jun. 2018.
[4] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. Annu. Cryptology Conf.*, 2010, pp. 465–482.
[5] 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.
[6] C. Papamanthou, E. Shi, and R. Tamassia, "Signatures of correct computation," in *Proc. Theory Cryptogr. Conf.*, 2013, pp. 222–242.
[7] K. Elkhiyaoui, M. Önen, 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.
[8] W. Song, B. Wang, Q. Wang, C. Shi, W. Lou, and Z. Peng, "Publicly verifiable computation of polynomials over outsourced data with multiple sources," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 10, pp. 2334–2347, Oct. 2017.
[9] J. Sun, B. Zhu, J. Qin, J. Hu, and J. Ma, "Confidentiality-preserving publicly verifiable computation schemes for polynomial evaluation and matrix-vector multiplication," *Secur. Commun. Netw.*, vol. 2018, pp. 5 275 132:1–5 275 132:15, 2018.
[10] R. Canetti, B. Riva, and G. N. Rothblum, "Refereed delegation of computation," *Inf. Comput.*, vol. 226, pp. 16–36, 2013.
[11] H. Pagnia and F. C. Gärtner, "On the impossibility of fair exchange without a trusted third party," Darmstadt Univ. Technol., Darmstadt, Germany, Tech. Rep. TUD-BS-1999–02, 1999.
[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/en/bitcoin-paper
[13] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 30–41.
[14] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 211–227.
[15] C. Wang, K. Ren, J. Wang, and Q. Wang, "Harnessing the cloud for securely outsourcing large-scale systems of linear equations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1172–1181, Jun. 2013.
[16] 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 Security*, vol. 10, no. 1, pp. 69–78, Jan. 2015.
[17] H. S. Ahn and Y. H. Lim, "Distributed coordination for optimal energy flow in smart grid networks: High-order polynomial approach," in *Proc. 16th Int. Conf. Control Automat. Syst.*, 2016, pp. 660–665.
[18] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*, Berlin, Germany: Springer, 2016, pp. 649–666.
[19] R. Archibald, A. Gelb, and R. B. Platte, "Image reconstruction from undersampled fourier data using the polynomial annihilation transform," *J. Sci. Comput.*, vol. 67, no. 2, pp. 432–452, 2016.
[20] J. A. de la O Serna, "Synchrophasor measurement with polynomial phase-locked-loop taylor–fourier filters," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 2, pp. 328–337, Feb. 2015.
[21] M. Blondel, M. Ishihata, A. Fujino, and N. Ueda, "Polynomial networks and factorization machines: New insights and efficient training algorithms," in *Proc. 33nd Int. Conf. Mach. Learn.*, 2016, pp. 850–858.
[22] M. Sznaier and O. I. Camps, "SOS-RSC: A sum-of-squares polynomial approach to robustifying subspace clustering algorithms," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8033–8041.
[23] A. Shpilka and A. Yehudayoff, "Arithmetic circuits: A survey of recent results and open questions," *Found. Trends Theor. Comput. Sci.*, vol. 5, no. 3–4, pp. 207–388, 2010.
[24] "Horner's method," 2020. [Online]. Available: https://en.wikipedia.org/wiki/Horner%27s_method
[25] "Rinkeby: Ethereum testnet." Accessed: Jan. 2021. [Online]. Available: https://www.rinkeby.io
[26] J. Zhang, T. Xie, Y. Zhang, and D. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 859–876.
[27] "Ethereum." Accessed: Jan. 2021. [Online]. Available: https://ethereum.org/
[28] "Eosio - fast, flexible, and forward-driven." Accessed: Jan. 2021. [Online]. Available: https://eos.io/
[29] "Neo smart economy." Accessed: Jan. 2021. [Online]. Available: https://neo.org/
[30] "Provable - blockchain oracle service, enabling data-rich smart contracts." Accessed: Jan. 2021. [Online]. Available: https://provable.xyz/
[31] "Chainlink." Accessed: Jan. 2021. [Online]. Available: https://chain.link/
[32] "DOS network." Accessed: Jan. 2021. [Online]. Available: https://dos.network/
[33] S. Nick, "Formalizing and securing relationships on public networks," *First Monday 2(9)*, 1997. [Online]. Available: https://doi.org/10.5210/fm.v2i9.548
[34] J. Benet, "IPFS-content addressed, versioned, P2P file system," 2014. [Online]. Available: https://arxiv.org/abs/1407.3561
[35] "Protocol labs." Accessed: Jan. 2021. [Online]. Available: https://protocol.ai/
[36] M. A. P. Chamikara, P. Bertók, I. Khalil, D. Liu, S. Camtepe, and M. Atiquzzaman, "A trustworthy privacy preserving framework for machine learning in industrial IoT systems," *IEEE Trans. Ind. Inform.*, vol. 16, no. 9, pp. 6092–6102, Sep. 2020.
[37] M. Li *et al.*, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1251–1266, Jun. 2019.
[38] S. S. Arslan and T. Goker, "Compress-store on blockchain: A decentralized data processing and immutable storage for multimedia streaming," *CoRR*, vol. abs/1905.10458, 2019. [Online]. Available: http://arxiv.org/abs/1905.10458
[39] D. Middleton and S. Amundson, "Distributed blockchain oracle," US Patent 16/234,157, May 2, 2019.
[40] S. Woo, J. Song, and S. Park, "A distributed oracle using intel SGX for blockchain-based IoT applications," *Sensors*, vol. 20, no. 9, 2020, Art. no. 2725.
[41] "Origin sport whitepaper," Mar. 2018. [Online]. Available: https://www.originsport.io/Origin_Whitepaper_EN.pdf
[42] X. A. Wang, K.-K. R. Choo, J. Weng, and J. Ma, "Comments on "publicly verifiable computation of polynomials over outsourced data with multiple sources"," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1586–1588, Aug. 22, 2020.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.

[43] S. Benabbas, R. Gennaro, and Y. Vahlis, "Verifiable delegation of computation over large datasets," in *Proc. Annu. Cryptol. Conf.*, 2011, pp. 111–131.
[44] D. Catalano, D. Fiore, R. Gennaro, and K. Vamvourellis, "Algebraic (trapdoor) one-way functions and their applications," in *Proc. Theory Cryptogr. Conf.*, 2013, pp. 680–699.
[45] M. Backes, D. Fiore, and R. M. Reischuk, "Verifiable delegation of computation on outsourced data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 863–874.
[46] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 844–855.
[47] L. F. Zhang and R. Safavi-Naini, "Generalized homomorphic MACs with efficient verification," in *Proc. 2nd ACM Workshop ASIA Public-Key Cryptography*, 2014, pp. 3–12.
[48] L. F. Zhang and R. Safavi-Naini, "Batch verifiable computation of polynomials on outsourced data," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2015, pp. 167–185.
[49] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 445–454.
[50] R. Canetti, B. Riva, and G. N. Rothblum, "Two protocols for delegation of computation," in *Proc. Int. Conf. Inf. Theoretic Secur.*, 2012, pp. 37–61.
[51] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 229–243.
[52] H. Huang, X. Chen, Q. Wu, X. Huang, and J. Shen, "Bitcoin-based fair payments for outsourcing computations of fog devices," *Future Gener. Comput. Syst.*, vol. 78, pp. 850–858, 2018.
[53] M. Król and I. Psaras, "SPOC: Secure payments for outsourced computations," *CoRR*, vol. abs/1807.06462, 2018. [Online]. Available: http://arxiv.org/abs/1807.06462
[54] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *Proc. Annu. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2011, pp. 149–168.
[55] D. Catalano, D. Fiore, and B. Warinschi, "Homomorphic signatures with efficient verification for polynomial functions," in *Proc. Annu. Cryptol. Conf.*, 2014, pp. 371–389.
[56] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, 2013, pp. 238–252.
[57] Y. Sun, Y. Yu, X. Li, K. Zhang, H. Qian, and Y. Zhou, "Batch verifiable computation with public verifiability for outsourcing polynomials and matrix computations," in *Proc. Australas. Conf. Inf. Secur. Privacy*, 2016, pp. 293–309.
[58] L. F. Zhang and R. Safavi-Naini, "Protecting data privacy in publicly verifiable delegation of matrix and polynomial functions," *Des. Codes Cryptogr.*, vol. 88, no. 4, pp. 677–709, 2020.
[59] U. Feige and J. Kilian, "Making games short," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 506–516.
[60] "Golem." Accessed: Jan. 2021. [Online]. Available: https://golem.network/
[61] "Sonm — decentralized fog computing platform."Accessed: Jan. 2021. [Online]. Available: https://sonm.com/
[62] "iexec blockchain-based decentralized cloud computing." Accessed: Jan. 2021. [Online]. Available: https://iex.ec/
[63] C. Lin, D. He, X. Huang, X. Xie, and K.-K. R. Choo, "Blockchain-based system for secure outsourcing of bilinear pairings," *Inf. Sci.*, vol. 527, pp. 590–601, 2018.
[64] Y. Zhang, R. Deng, X. Liu, and D. Zheng, "Outsourcing service fair payment based on blockchain and its applications in cloud computing," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2018.2864191.
[65] H. Cui, Z. Wan, X. Wei, S. Nepal, and X. Yi, "Pay as you decrypt: Decryption outsourcing for functional encryption using blockchain," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3227–3238, Feb. 2020.

Yunguo Guan is currently working toward the PhD degree with the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include applied cryptography and game theory.

Hui Zheng received the master's degree from the School of Computer and Information Engineering, Zhejiang Gongshang University, China. Her research interests include applied cryptography and blockchain.

Jun Shao received the PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2008. He was a postdoc with the School of Information Sciences and Technology, Pennsylvania State University, from May 2008 to April 2010. He was also a visiting professor with the Faculty of Computer Science, University of New Brunswick, Canada from October 2017 to March 2018. He is currently a professor with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China. His research interests include network security and applied cryptography.

Rongxing Lu (Fellow, IEEE) received the first PhD degree from Shanghai Jiao Tong University, China, in 2006, and the PhD degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He is currently an associate professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor with the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. He worked as a postdoctoral fellow with the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governor General's Gold Medal" and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise (with H-index 72 from Google Scholar as of November 2020), and was the recipient of nine best (student) paper awards from some reputable journals and conferences. Currently, he serves as the vice-chair (Conferences) of the IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee). He is the Winner of 2016-17 Excellence in Teaching Award, FCS, UNB.

Guiyi Wei received the PhD degree from Zhejiang University, in December 2006. He is currently a professor with the School of Electronic and Information Engineering, Zhejiang Gongshang University. He was advised by Cheung Kong chair professor Yao Zheng. His research interests include wireless networks, mobile computing, cloud computing, social networks and network security.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: University of New Brunswick. Downloaded on January 11,2023 at 13:20:17 UTC from IEEE Xplore. Restrictions apply.