2017 IEEE 23rd International Conference on Parallel and Distributed Systems

# Scalable Blockchain Based Smart Contract Execution

Zhimin Gao, Lei Xu, Lin Chen, Nolan Shah, Yang Lu, Weidong Shi
Computer Science Department
University of Houston
Houston, Texas, United States
mtion@msn.com, xuleimath@gmail.com, chenlin198662@gmail.com, nolanshah212@gmail.com, ylu17@uh.edu, wshi3@uh.edu

***Abstract*—Blockchain, or distributed ledger, provides a way to build various decentralized systems without relying on any single trusted party. This is especially attractive for smart contracts, that different parties do not need to trust each other to have a contract, and the distributed ledger can guarantee correct execution of the contract. Most existing distributed ledger based smart contract systems process smart contracts in a serial manner, i.e., all users have to run a contract before its result can be accepted by the system. Although this approach is easy to implement and manage, it is not scalable and greatly limits the system's capability of handling a large number of smart contracts. In order to address this problem, we propose a scalable smart contract execution scheme that can run multiple smart contract in parallel to improve throughput of the system. Our scheme relies on two key techniques: a fair contract partition algorithm leveraging integer linear programming to partition a set of smart contracts into multiple subsets, and a random assignment protocol assigning subsets randomly to a subgroup of users. We prove that, our scheme is secure as long as more than $50\%$ of the computational power is possessed by honest nodes. We then conduct experiments with data from existing smart contract system to evaluate the efficiency of our scheme. The results demonstrate that our approach is scalable and much more efficient than the existing smart contract platform.**

***Index Terms*—smart contract, blockchain, scalability**

## I. Introduction

The blockchain is a decentralized ledger maintained by a group of independent users. Users can add new blocks to the blockchain through proof-of-work, i.e., solving a computationally intensive problem and attaching the proof to the newly generated block. Everyone who receives the new block can easily verify the proof to determine whether the new block is valid or not. When there are more than one branches, users choose the longest one as the legitimate blockchain. As long as the majority of the users are honest, an attacker cannot fake a longer branch as he/she needs to control more than $50\%$ of the total computational resources to achieve this [39]. The decentralized feature of the blockchain makes it a perfect platform to implement business logic or process such as smart contracts without requiring trust among parties who are involved in the contracts [34], [7]. Briefly speaking, a smart contract is a computer program that executes the terms of a contract, which is submitted to the blockchain system by its creators and embedded into a block. After the block is accepted by the system, all users try to execute the smart contract and create a new block to hold the execution result. Everyone in the system checks the result before accepting the new block and correctness is guaranteed if majority of the users are honest. To encourage users to participate in mining and execution of smart contracts, rewards/incentives are provided to those who successfully produce a valid block or a smart contract execution result.

The original design of proof-of-work based blockchain system is not efficient, and many recent studies have been done to improve the performance of crypto-currency based systems [12], [14], [22]. However, these approaches are not directly applicable to smart contracts mainly due to the extra requirements of contract execution and additional complexity for supporting smart contracts over blockchain. Currently, most existing blockchain systems that support smart contracts/chain codes execute contracts one by one in a strictly sequential manner. The situation becomes even worse when considering that smart contracts themselves could be more computationally sophisticated than the mining process. For example, a computationally extensive smart contract could occupy all users' computational resources and prevent other smart contracts from being executed at all. There is a lack of scalable technologies that allow efficient and high throughput execution model of smart contracts, which greatly limits the potential of smart contract based systems built on top of blockchain.

To address these challenges, we develop a scalable blockchain based smart contract system that enables multiple smart contracts to be executed simultaneously. The new system applies divide-and-conquer principle. It provides a novel method to partition a set of smart contracts into subsets satisfying fairness constraints, and randomly assign each subset to a subgroup of users without reducing the strength of proof-of-work based systems. In this system, a smart contract is only executed and verified by a subset of the users. However, we prove that it captures almost the same level of security as those systems that process smart contracts sequentially, i.e., correctness of the smart contract execution is guaranteed in our scheme as long as majority of the users are honest.

In summary, our main contributions include:

- We provide a scalable smart contract execution scheme, which provides a novel approach to partition smart contracts satisfying fairness constraint; and then assign them to users via a random assignment protocol;
- We present theoretical analyses of the proposed scheme that show its correctness and security strength;
- We also conduct simulations to demonstrate practicality of the scalable smart contract execution scheme.

The remainder of the paper is organized as follows. In Section II we describe backgrounds and a formal definition of our problem. In Section III we provide an overview of the scalable smart contract execution scheme. Detailed design of

978-1-5386-2129-5/17/31.00 ©2017 IEEE
DOI 10.1109/ICPADS.2017.00054

key components is given in Section IV. Security and fairness analysis of the proposed scheme are given in Section V and Section VI. In Section VII we evaluate performance of our approach with data collected from Ethereum. We review related works in Section VIII and conclude the paper in Section IX.

## II. Preliminaries

In this section, we provide a brief introduction of blockchain and smart contract, and mathematical tools used in this paper.

### A. Blockchain and Smart Contracts

A blockchain, or a distributed ledger, is a system involving multiple users who work together to maintain a set of blocks that are linked together[38]. Blockchain based systems are developed under different trust models with different consensus protocols. In this paper, we are only concerned with proof-of-work based public blockchain [25], which is the most widely used blockchain scheme. Specifically, a proof-of-work based public blockchain is a blockchain open to everyone, in which all users compete with each other to solve computationally intensive problems to build new blocks. When there are more than one branches in the blockchain, users follow the principle of longest-chain, i.e., the branch with the most number of blocks is the legitimate one.

We remark that there are also other kinds of blockchain systems, e.g., permissioned blockchain [37]. Although the proposed approach is also applicable to smart contracts based on permissioned blockchain, it is not the focus of this paper because most permissioned blockchains do not depend on proof-of-work.

Smart contracts have recently become one of the most promising applications of blockchain technology. The concept of a smart contract was first proposed by Szabo in 1997 [34]: *A smart contract is a set of promises, specified in a digital form, including protocols within which the parties perform on these promises.* Roughly speaking, a smart contract is a piece of program that consists of a set of rules and corresponding operations of related accounts [40], [11]. Current blockchain systems that support smart contracts work in a serial manner, i.e., all smart contracts submitted to the system are executed one by one sequentially, and each smart contract has to be run by all users. As a result, existing smart contract systems do not scale with the number of users and performance of contract execution is limited due to the sequential execution model.

### B. Terms and Notations

We summarize terms and notations that are used in this paper as follows.

**Users.** Users refer to the real identities who participate in the system.

**Pseudonyms.** Users are identified with pseudonyms, which are their public keys and are chosen by themselves with the same public parameters. If the length of a public key is $\beta$, we assume that all the public keys are distributed uniformly in $\{0,1\}^{\beta}$. If the original public keys are not uniformly distributed, we can apply some transformations to guarantee this feature [4], [35], [10].

**Workload.** The workload of a smart contract is the amount of computation required to obtain result of this smart contract. In general, a smart contract with heavier workload is associated with a higher profit (e.g., concept of gas in Ethereum).

**Rate.** The rate is the ratio between the profit of a smart contract and its workload.

### C. Useful Probability Bounds

Our analysis used following probability bounds.

**Lemma 1** (Chernoff Bounds [24])**.** *Suppose $X_1, X_2, \cdots, X_\ell$ are independent random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^{\ell} X_i$ and $\mu = \mathbb{E}[X]$. Then for any $\delta \in (0,1)$, we have*

$$\Pr(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{3}}.$$

**Lemma 2** (Balls into Bins [30])**.** *Suppose that there are $n$ identical balls and $m$ identical boxes (or "bins"). Each time, one of the bins is selected uniformly at random and a single ball is placed into it. After all balls are in the bins, consider the bin that contains the largest number of balls and let this $z$ be such a number, then for any $\epsilon > 0$ there exists some constant $c$ such that*

$$\Pr[z \geq (1+\epsilon)\frac{n}{m}] \leq \epsilon, \quad \text{if } n > m\log m.$$
$$\Pr[z \geq c\log m] \leq \epsilon, \quad \text{if } n \leq m\log m$$

Briefly speaking, with $1-o(1)$ probability every bin contains no more than $(1+o(1))\frac{n}{m}$ balls if $n > m\log m$, and no more than $O(\log m)$ balls if $n \leq m\log m$.

## III. Overview of the Scalable Smart Contract Execution Scheme

In this section, we first analyze the essential requirements of scalable smart contract execution and then provide an overview of our proposed approach.

### A. Essential Requirements

The basic concept for improving scalability of a blockchain based smart contract system is natural: we divide smart contracts into subsets and users within the system into groups (which we call sub-committees), and let users in each sub-committee only work on a subset of smart contracts. By doing so, different smart contracts are executed simultaneously; and scalability is thereafter achieved. However, a straightforward implementation based on this natural idea may cause serious problems regarding security and fairness, as we elaborate below.

**Security:** Our scheme should ensure that any attacker or malicious user who control a small percentage of computational resources cannot compromise or hijack the whole system. There are two important issues. One issue is related to the (absolute) percentage of computational power owned by users. Note that a user may generate as many pseudonyms as he/she wants. Hence we cannot simply partition pseudonyms into sub-committees. Indeed, sub-committees should consist of pseudonyms that have proved their computational power so that the attacker cannot participate in all sub-committees and strategically works on the smart contract that he/she prefers by generating a lot of pseudonyms. To achieve this, when we partition pseudonyms into sub-committees, we only consider the set of pseudonyms that have generated blocks recently. More precisely, we consider the most recent $\ell$ blocks for some constant $\ell$ and only the pseudonyms that have generated one of these blocks will be selected into one of the sub-committees. Consequently, if an attacker generates a lot of pseudonyms, these pseudonyms will be ignored if no blocks are generated by them. The second issue is related to the relative percentage of computational power owned by users. Note that an attacker only controls a small percentage of computational power with respect to the whole set of users. However, once in a small sub-committee, he/she may turn out to control more than $50\%$ of the total computational power within that sub-committee. Therefore, the scheme needs to ensure that even if a user controls majority of the computational power within a sub-committee, he/she cannot manipulate the execution result of the smart contract. To handle this, all the pseudonyms in a sub-committee are treated as the same regardless of their computational power. The execution result of a smart contract is verified by pseudonyms within a sub-committee and will be accepted if it is approved by a majority of them. Hence, to manipulate the execution result, the attacker needs to control more than half different pseudonyms in a sub-committee. We prove that, if each pseudonym that has generated one of the most recent $\ell$ blocks is randomly assigned to one sub-committee, then for any user who controls $\lambda$ fraction of the computational power, the probability that he/she can control more than $(1+\epsilon)\lambda$ fraction of the pseudonyms within any sub-committee is sufficiently low for arbitrarily small constant $\epsilon$. Therefore, an attacker cannot compromise the system without being able to control more than $(50-\epsilon)\%$ of the total computational power.

**Fairness:** To ensure that users in each sub-committee are working on distinct smart contracts, it is required that users in one sub-committee cannot work on smart contracts that belong to another sub-committee. Such a requirement may cause some problem regarding fairness. Indeed, if some sub-committee is assigned with very profitable smart contracts while another sub-committee is assigned with much less profitable ones, it may harm the incentive of users to participate in the execution of smart contracts. To handle this, we propose a two-level load balancing scheme. First, we require that all the smart contracts to be assigned to sub-committees are partitioned in a balanced way in terms of their total workload and profits. The partition program will be formulated as a special smart contract and will be executed in advance. We show that, in general, computing a balanced partition is NP-hard, and we propose an efficient algorithm based on ILP (Integer Linear Programming) techniques. Therefore, every user will run the ILP-solver to produce a balanced partition, which is the first level of our load balancing scheme ensuring that workload and profits of smart contracts do not differ too much among different sub-committees. The second level of our load balancing scheme is to ensure that every pseudonym is randomly assigned to one of the sub-committees. Therefore, even if a user is assigned to a sub-committee working on less profitable smart contracts at some rounds, he/she should be assigned to a sub-committee working on more profitable smart contracts at other rounds. In expectation, we prove that the total workload finished by a user and the total profit he/she can get is proportional to his/her computational power.

### B. Overview of the Scalable Blockchain based Smart Contract Execution Scheme

From a high level, our scheme works as follows: it first collects $n$ smart contracts, then creates a special smart contract asking for a balanced partition of the $n$ contracts into $m$ subsets. All users execute this special smart contract and compute the partition. The scheme then checks the most recent $\ell$ blocks prior to this special smart contract and collects the $\ell$ pseudonyms that generate these blocks (if multiple blocks are generated by one pseudonym, we create the same number of copies of this pseudonym and treat each copy as distinct). The scheme then creates $m$ sub-committees, each containing one subset of the smart contracts. It randomly assigns each pseudonym to one sub-committee. Pseudonyms in each sub-committee only work on the smart contracts of this sub-committee (consequently if a user owns several pseudonyms that are in different sub-committees, he/she is able to work on smart contracts in these sub-committees). Within each sub-committee, pseudonyms use the Byzantine protocol to reach consensus on the execution results[1]. The execution result of each smart contract is then broadcasted by the sub-committee to the whole system. Users whoever mines the next block will put the execution result in the block. The profit of a smart contract will be evenly distributed among all the pseudonyms in the sub-committee that executes it. Algorithm 1 gives an overview of the scalable blockchain based smart contract execution scheme. Details of sub-routings are discussed in the following sections.

**Algorithm 1** Scalable blockchain based smart contract execution scheme.

```
Input: $L$, the blockchain of the smart contract platform; $n$, the
    number of smart contracts to be collected before execution;
    $m$, the number of sub-committees; $U$, the set of pseudonyms
    that have generated the most recent $\ell$ blocks.
Output: Updated blockchain $L$ with smart contracts and
    execution results
  % the system runs continually
  while true do
    % collection of smart contracts
    for $i = 1$ to $n$ do
      User submits smart contract $c_i$ to $L$
    end for
    % the whole runs the special contract to partition a set
    of smart contracts to subsets $s_i$, which are then submitted
    to the blockchain
    $\{s_1, \ldots, s_m\} \leftarrow \texttt{Partition}(c_1, \ldots, c_n)$
    $\texttt{AddPartition}(L, \{s_1, \ldots, s_m\})$
    % every user achieves consensus on how to randomly
    assign contract subsets, sub-committee $A_i$ is responsible
    for $s_i$
    $\{A_1, \ldots, A_m\} \leftarrow \texttt{RandomAssign}(L, U, \{s_1, \ldots, s_m\})$
    in parallel
      $r_i \leftarrow \texttt{Execute}(U, A_i, s_i)$
      $\texttt{AddResult}(L, r_i)$
    end parallel
  end while
```

Algorithm 1 only describes one-round smart contracts execution. After the system receives enough number of smart contracts, it can schedule another round of partition and random assignment without waiting for termination of the previous round. Therefore, the system can keep $m$ smart contracts running in parallel.

[1] Briefly speaking, the Byzantine protocol is a protocol that allows users in a network to reach consensus on the correct result, given that more than $50\%$ of them are honest [21].

Although there are some works on verifiable computation [17], [29], they usually involve expensive cryptography operations and key management. For general smart contracts, repeating the computation is still the most feasible way to verify correctness of results. This prevents a smart contract system from being scalable. In our scheme, this is not a problem for the `Partition` operation as it uses a special algorithm that is much easier to verify a result than to compute one. We also limit verification of each smart contract result within the sub-committee that it is assigned to. Once the result is broadcast to the whole system, pseudonyms in other sub-committees and miners who attempt to generate a block to hold this result are not required to verify the result again. Considering all these features, the proposed smart contract execution scheme can achieve a speedup rate that is linear to the number of sub-committees.

## IV. Efficient Smart Contract Execution

In the following two subsections, we discuss the two key components of our scheme, the partitioning of smart contracts and the assignment of pseudonyms to sub-committees.

### A. `Partition`: Partitioning of Smart Contracts

The partitioning of smart contracts could be formulated as the following *fair load balancing* problem.

**Fair Load Balancing:** Given a set $J$ of $n$ tasks. Each task $j$ is specified by

- a workload $W_j$ which is random variable with its mean $w_j = \mathbb{E}(W_j)$,
- a rate $r_j$ for the processing of each unit of its workload, i.e., the profit of processing task $j$ is $r_j w_j$.

The goal is to partition $J$ into $m$ disjoint subsets $J_1, J_2, \cdots, J_m$ in a fair way, that is, for some parameter $\gamma_1, \gamma_2 \geq 1$, the followings are true:

$$\frac{\max_{k=1}^{m} \sum_{j \in J_k} w_j}{\min_{k=1}^{m} \sum_{j \in J_k} w_j} \leq \gamma_1, \tag{1a}$$

$$\frac{\max_{k=1}^{m} \sum_{j \in J_k} r_j w_j}{\min_{k=1}^{m} \sum_{j \in J_k} r_j w_j} \leq \gamma_2 \tag{1b}$$

For simplicity, we define the workload and the profit of each subset $J_i$ as the total workload and total profit of tasks it contains, respectively. Without loss of generality, we assume $w_j, r_j$ are integers (if not, we simply scale up the instance).

In this model, each task corresponds to a smart contract. Note that all the smart contracts will be partitioned and assigned to $m$ sub-committees, therefore, it is required that such a partition is fair in the sense that the total workload assigned to different sub-committees, as well as the total profit of the tasks assigned to different sub-committees, are more or less the same. To measure the fairness, we introduce two parameters $\gamma_1$ and $\gamma_2$, which are defined as the maximal workload over the minimal workload, and the maximal profit over minimal profit, respectively. Obviously the best partition in terms of fairness satisfies that $\gamma_1 = \gamma_2 = 1$. The following theorem, however, implies that the fair load balancing problem is a challenging problem. Indeed, even determining whether there exists a feasible solution satisfying that $\gamma_1 = \gamma_2 = 1$ is NP-hard.

**Theorem 1.** *The fair load balancing problem is NP-hard.*

We have shown our proof of theorem in the full version of this paper [15]. In the following part we show how to solve the fair load balancing problem efficiently by utilizing integer linear programming.

*1) A Bi-objective Optimization Formulation:* We can formulate the fair load balancing problem as the following bi-objective optimization programming.

$$\min(\gamma_1, \gamma_2) \tag{2a}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad 1 \leq j \leq n \tag{2b}$$

$$\sum_{j=1}^{n} w_j x_{ij} \leq \gamma_1 \sum_{j=1}^{n} w_j x_{i'j}, \qquad 1 \leq i, i' \leq m \tag{2c}$$

$$\sum_{j=1}^{n} r_j w_j x_{ij} \leq \gamma_2 \sum_{j=1}^{n} r_j w_j x_{i'j}, \qquad 1 \leq i, i' \leq m \tag{2d}$$

$$x_{ij} \in \{0, 1\}, \qquad 1 \leq i \leq m, 1 \leq j \leq n \tag{2e}$$

We explain the variables. Here $x_{ij}$ denotes whether task $j$ is in subset $J_i$, i.e., $x_{ij} = 1$ if task $j$ is in $J_i$ and $x_{ij} = 0$ otherwise.

We explain the constraints. Constraint (2b) implies that every task must be assigned to one subset, whereas the solution of the ILP produces a feasible partition. Constraint (2c) implies that the total workload of tasks in one subset is no more than $\gamma_1$ times the total workload of tasks in another subset, therefore, the maximal workload is no more than $\gamma_1$ times the minimal workload. Similarly, constraint (2d) implies that the total profit of tasks in one subset is no more than $\gamma_2$ times the total profit of tasks in another set, whereas the maximal profit is no more than $\gamma_2$ times the minimal profit. Given $\gamma_1$ and $\gamma_2$, we can test whether $ILP(\gamma_1, \gamma_2)$ admit a feasible solution or not.

Note that the two parameters $\gamma_1$ and $\gamma_2$ may contradict each other, i.e., a solution that partitions tasks fairly in terms of workload may result in an unfair distribution of the profits. A common approach for bi-objective optimization is to compute its *pareto front* [18], which is composed of all the points $(\gamma_1^*, \gamma_2^*)$ such that by fixing $\gamma_1 = \gamma_1^*$, the minimal value of $\gamma_2$ is $\gamma_2^*$.

To compute the pareto front, we can set $\gamma_1$ to be different values and change the objective into $\min \gamma_2$. We remark that, by minimizing $\gamma_2$, we establish a *non-linear* integer programming.

We observe that, if we fix the values of $\gamma_1$ and $\gamma_2$, then the bi-objective optimization program reduces to an integer linear programming (ILP). We denote such an integer programming as $ILP(\gamma_1, \gamma_2)$. In general, integer linear programming can be solved significantly faster than a non-linear integer programming. Therefore, we can compute the pareto front by applying binary search on $\gamma_2$. Refer to the full version of this paper [15] for an overview of the proposed algorithm.

*2) Solving the ILP Faster.:* We observe that, $ILP(\gamma_1, \gamma_2)$ has $mn$ different 0-1 integral variables, and $2m^2 + n$ constraints (including constraints (2b), (2c) and (2d)). Discussion about how to solve the integer programming faster by reducing the number of contraints to $0(m+n)$ can be found here [15]. This modified ILP can be solved efficiently using ILP-solvers like Lingo or Fico Xpress. Detailed discussion about the experimental results are presented in Section VII.

### B. `RandomAssign`: Random Assignment of Pseudonyms

According to the design given in Algorithm 1, each subset of smart contracts generated by `Partition` is handled by a sub-committee. Here we discuss the process to randomly assign each pseudonym to one of the sub-committees. More precisely, our scheme should ensure that for each pseudonym, the probability that it is assigned to a sub-committee is exactly $1/m$. We show how to achieve this random assignment problem in a decentralized system.

First, we prove that the random assignment problem is equivalent with generating a random permutation of all users (Theorem 2) [15].

**Theorem 2.** *Let $\ell$ be the number of pseudonyms, $m$ be the number of sub-committees, $m|\ell$ and $m < \ell$. Let $U = \{u_1, \cdots, u_\ell\}$ be the set of pseudonyms and $T = \{T_1, \cdots, T_m\}$ be the set of sub-committees. Assigning each user randomly to a sub-committee is equivalent with applying a random permutation $p$ to $U$ and assigning $(u^{(p)}_{i\frac{n}{m}+1}, \cdots u^{(p)}_{(i+1)\frac{n}{m}})$ to $T_{i+1}$, where $i = 0, \cdots, m-1$ and $u^{(p)}_j$ is the $j$-th user after the permutation.*

In practice, it is not easy to select a real random permutation efficiently, and we use pseudo-random permutation in our scheme. The effect of this replacement is analyzed in the following section. In a nutshell, the assignment process works in three steps: first the system selects a random number; then the random number is used to construct a pseudo-random permutation; and finally smart contract subsets are assigned accordingly to the constructed sub-committees.

**Random number generation.** A randomness source is the precondition for pseudo random permutation construction. Because our scheme works in a fully decentralized environment and there is no trusted party, we cannot rely on any single party for random number generation.

Instead of running a dedicated protocol between all users to generate a random number [2], [6], we use the blockchain itself as the randomness source [5]. As we consider the scenario where the blockchain is constructed using proof-of-work [25], the nonce calculated by the miner can provides some randomness. The time stamp is also a source of randomness. We then apply a random bits extractor [13] to the source to build the random number.

**Random permutation construction.** A block cipher is intended to be computationally indistinguishable from random permeation [3]. Roughly speaking, given a block cipher scheme with random keys and a random permutation, an attacker with limited computation resources cannot distinguish them by observing input/output pairs. For our scheme, we use AES as the underlying block cipher, which has received intensive study and is believed to be a secure pseudo random permutation [26]. The AES encryption/decryption key is derived by applying HMAC [13] to the random number generated in the previous step. After the random number is generated, all users in the smart contract system can learn the value of the key. This is not a problem for our scheme as we utilize AES for permutation, not confidentiality protection.

Without loss of generality, let each pseudonym $u$ be identified by a public key $pk_u$, where $|pk_u| = \beta$. If $128 \nmid \beta$, we pad $pk_u$ with zeros to make sure the padded result is a multiple of 128. The padded public key is denoted as $pk'_u$ and the new length is $\beta'$. The padded public key is then encrypted using AES CBC mode, denoted as $E(pk'_u)$.

**Smart contract subsets assignment.** The encrypted public key can be treated as an integer, and we divide the cipher-text space to $m$ parts: $[0, \frac{2^{\beta'}}{m}], [\frac{2^{\beta'}}{m} + 1, 2 \cdot \frac{2^{\beta'}}{m}], \cdots, [(m-1) \cdot \frac{2^{\beta'}}{m}, 2^{\beta'}]$. The cipher-text of a public key falls into one of the $m$ parts with probability $1/m$. The $i$-th smart contract subset is assigned to pseudonyms with encrypted public keys that fall into the $i$-th slot. Because public keys, the AES key, and the dividing information are public, it is easy for one to check which subset of contracts is assigned to a specific pseudonym.

### C. `Execute`: Smart Contracts Execution and Results Collection

After each subset of smart contracts is assigned to a sub-committee, pseudonyms in each sub-committee run the assigned contracts one by one and submit corresponding results to the system. We discuss execution of a single smart contract $c$.

A pseudonym $u$ who is assigned to $c$ executes it locally and obtains the result $r$. $u$ also generates a digital signature $sig$ on $r$. Other pseudonyms in the same sub-committee do the same thing and they can run a BFT protocol to determine the final result; and broadcast it to the whole system. A straightforward approach to build the final result is to attach all digital signatures together with the execution result. However, the size of the broadcast message increases as more pseudonyms are included in the sub-committee. To mitigate this problem, we leverage multi-signature scheme [27] to compress multiple signatures for the same message (i.e., the execution result) into a single signature.

One who receives the broadcast result first checks whether the result is generated by legitimate pseudonyms (by checking the random assignment of pseudonyms) and supported by enough number of pseudonyms in the sub-committee (by checking the aggregated digital signature). If the result passes these tests, one can start the mining process and try to embed the result in a new block.

## V. Security Analysis

In this section, we analyze the security of the proposed scalable smart contract execution scheme. As a user can create multiple pseudonyms, Theorem 3 shows that he/she cannot generate enough number of pseudonyms to dominate a sub-committee unless he/she controls more than half of the computation resources.

**Theorem 3.** *Let $\epsilon > 0$ be an arbitrarily small positive number and $\lambda \in (0, 1)$ be any constant. Let $\ell \in \mathbb{Z}^+$ be such that $\lambda\ell \geq \max\{\frac{1}{\epsilon^2} \ln \frac{1}{\epsilon}, cm \log m\}$ where $c$ is some constant, then with sufficiently high probability (at least $1 - 2\epsilon$), for any user that controls $\lambda$ fraction of the total computation power, the number of his/her pseudonyms in each of the $m$ sub-committee does not exceed $(1+\epsilon)^2 \frac{\lambda\ell}{m}$, i.e., he/she cannot own more than $(1+\epsilon)\lambda$ fraction of the pseudonyms in every sub-committee.*

Here $c$ is some universal constant that follows from the constant appears in the fundamental problem of "Balls into Bins"as we described in Section II-C. Note that $c$ is independent of $\lambda, \ell, m$ and only depends on $\epsilon$.

Briefly speaking, the theorem states that with high probability our scheme ensures that the fraction of pseudonyms belonging to one user in each sub-committee does not exceed the fraction of computational power owned by him/her, consequently, taking $\lambda = 1/2 - 2\epsilon$, no user can control more than $50\%$ pseudonyms in each sub-committee as long as he/she does not control more than $(50-\epsilon)\%$ of the total computational power, and the security of our scheme is ensured. Note that it is required by the theorem that $\ell = \Omega(m \log m)$, that is, the number of sub-committees needs to be bounded by roughly $O(\ell / \log \ell)$. A complete proof can be found in the full version of this paper [15].

## VI. Fairness Analysis

Fairness is critical for the new scalable smart contract execution scheme. If smart contract partition is not fair, users may not have the incentives to run certain smart contracts, which will affect the overall scalability. In this section, we show that the proposed scheme ensures both the long term and short term fairness for all users.

**Long term fairness.** We prove the following theorem, which shows that in expectation, workload and profit of smart contracts are distributed among users in fair way, that is, it is proportional to the computational power of a user. This is reasonable as users with more computational power are able to process more tasks, and consequently get more reward. Note that this is also the case with the current public blockchain system such as Bitcoin or Ethereum.

**Theorem 4.** *In expectation, any user who controls $\lambda \in (0, 1)$ fraction of the total computational power will complete $\lambda$ fraction of the total workload and get $\lambda$ fraction of the total profit.*

We remark that the total profit is $\sum_{j=1}^{n} r_j w_j$, but the total workload is not $\sum_{j=1}^{n} w_j$ but rather $\ell/m \cdot \sum_{j=1}^{n} w_j$ since in each sub-committee, execution of every smart contract is repeated by every pseudonym, hence the workload is increased by $\ell/m$ times.

Towards the proof, we need the following lemma which follows readily as a consequence of the random assignment of pseudonyms to sub-committees.

**Lemma 3.** *If every pseudonym is randomly assigned to one of the $m$ sub-committees, then the expected workload finished by him/her is $\frac{\sum_{j=1}^{n} w_j}{m}$, and the expected profit got by him/her is $\frac{\sum_{j=1}^{n} r_j w_j}{\ell}$.*

Proof of theorems and lemmas in this section can be found here [15].

**Short term fairness.** Note that the total workload finished by a user and the reward received converges to the expectation in the long run. However, if a user joins the system and leaves after a short time, then his/her workload and profit may vary. Though the expectation is fixed, it makes much difference if he/she is lucky and assigned to more profitable smart contracts vs. if he/she is unlucky and assigned to less profitable smart contracts. Indeed, users who are unlucky may also have less incentives to continue participating in the system. Therefore it is also important to ensure short term fairness among users, that is, the total workload and profit of smart contracts in each sub-committee should be similar. Our scheme handles this by its ILP-based load balancing algorithm that returns a solution such that the difference of workload and profit among sub-committees are bounded by $\gamma_1$ and $\gamma_2$, respectively. We provide experiment results in Section VII, which shows the values of these two parameters by taking as input the real data collected from Ethereuam. The experimental results suggest that, $\gamma_1$ and $\gamma_2$ can both be close to 1 for, e.g., 5 sub-committees and 100 smart contracts. Therefore, even if a user joins the system and leave shortly afterwards, he/she can be treated in an approximately fair way no matter he/she is lucky or unlucky.

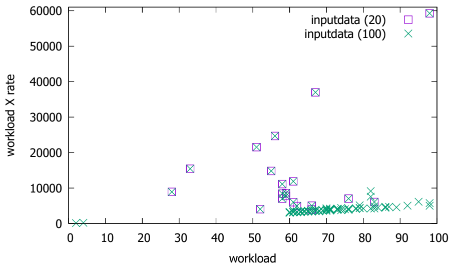


Fig. 1: The input data is collected and normalized from the first 100 transactions of Ethereum on May 31, 2016. The subset of 20 inputs are marked by square symbols.

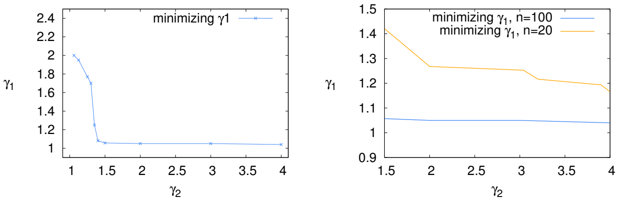
## VII. Experiments and Evaluation

We have proved in the previous sections that our scheme ensures the security and long term fairness among users (Note that correctness of Theorem 3 and Theorem 4 do not rely on partition of smart contracts). The short term fairness, however, relies on how evenly smart contracts can be divided among sub-committees, which is guided by the two parameters $\gamma_1$ and $\gamma_2$. Ideally, if $\gamma_1 = \gamma_2 = 1$, then perfect short term fairness is achieved. However, if the data is not distributed nicely, then it may be impossible to partition smart contracts in a good fashion. For example, consider two smart contracts, one of profit 1 and the other of profit 100. Partitioning them into two subsets yields a solution with $\gamma_2 = 100$, which may be too large to be acceptable. In reality, however, the data is neither too good nor too bad. The goal of this section is to calculate the values of $\gamma_1$ and $\gamma_2$ by taking as an input the real data collected from Ethereum, thereby illustrating that our scheme can achieve approximate short term fairness if it is applied to a smart contract system such as one similar Ethereum.

Throughout this section, we use the data taken from Ethereum on May 31, 2016 (see Fig. 1). As the ILP-solver for the sub-routine $\texttt{Partition}(c_1, \ldots, c_n)$, we run LINGO 11 on Windows 10 64bit with 16 GB total memory and an Intel®Core$^{\text{TM}}$i7-4790S CPU. LINGO is a comprehensive tool to solve integer linear optimization problem.

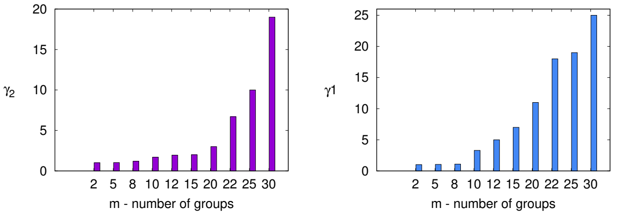
Let $m = 5$. We fix the value of $\gamma_1$ and compute the minimal value of $\gamma_2$. Each pair of the values $(\gamma_1, \gamma_2)$ is a pareto optimal solution. We compute all the pareto optimal solutions and derive a pareto front, as is illustrated by Fig. 2a. From the figure, we see that $\gamma_1$ drops to almost 1 when $\gamma_2$ is around 1.4, hence 5 sub-committees works well for 100 smart contracts.

We further discuss the possibility of increasing $m$ with respect to the 100 smart contracts. In Fig. 3a, we ignore the fairness of workload in the partition by setting $\gamma_1$ to be some large value (here we use $\gamma_1 = 100$), and focus on the value of $\gamma_2$ with respect to $m$. We observe that, $\gamma_2$ remains around 2 until $m = 15$, and increases quickly to nearly 20 when



(a) (b)

Fig. 2: (a) $m = 5$, $n = 100$, the pareto front for $(\gamma_1, \gamma_2)$; (b) n=100 and 20, m=5, the two pareto fronts.



(a) (b)

Fig. 3: (a) $n = 100$, $\gamma_1$=100, the minimal $\gamma_2$ with respect to $m$; (b) $n = 100$, $\gamma_2$=100, the minimal $\gamma_1$ with respect to $m$.

$m = 30$. In Fig. 3b, we ignore the fairness of profit in the partition by setting $\gamma_2 = 100$, and focus on the value of $\gamma_1$ with respect to $m$. We observe that, $\gamma_1$ remains slightly larger than 1 until $m = 8$, and increases fast to nearly 25 when $m = 30$. The trend of increasing on both parameters is not difficult to understand. Generally, dividing a fixed number of smart contracts into more sub-committees resulting less number of smart contracts within each sub-committee, causing the workloads and profits among sub-committees to differ more. This phenomenon is further illustrated in Fig. 2b, where we compute the pareto front of $(\gamma_1, \gamma_2)$ for $m = 5$ by taking as an input a subset of the 100 smart contracts (see the squared data in Fig. 1). We can see that the whole pareto front for $n = 100$ is completely below the pareto front for $n = 20$.

Note that $m$ measures the scalability of our scheme. While ensuring the short term fairness, the scalability of our scheme increases with more smart contracts. If there are sufficiently many smart contracts, we may expect that most of them have very similar workloads and profits (as we can already see from Fig. 1), allowing for a good partition into much more sub-committees.

## VIII. Related Works

**Related work on blockchain based transaction systems.** The study of e-cash systems is not new. There are various well-known online payment systems, including Visa, Mastercard, Paypal, and Moneygram. Theoretical studies on anonymous e-cash systems date back to 1983 by Chaum [8], which is followed by a series of subsequent researches, e.g. [31]. One property that is shared by all of such systems is that they are centrally or quasi-centrally administrated in the sense that there exists a central controlling authority (e.g., bank) who has access to the information of all transactions carried out within the system. A well-known exception, Bitcoin, was introduced by Nakamoto [25] in 2008. Instead of appointing a central authority, Bitcoin uses a public ledger, which is also known as a *blockchain*, to record transactions carried out between users. Following this line of research, various alternative blockchain based transaction systems are proposed [23], [32], further improving the performance of Bitcoin. All of these transaction systems focus on mining and transactions between users. Ethereum [36] goes further by introducing smart contracts in a blockchain based system. We remark that, the concept of smart contract is actually not new. In 1997, Szabo [34] first introduced smart contract. It could be viewed as a counterpart of a contract in an online system. However, implementing smart contracts in a decentralized blockchain based transaction system is a major challenge. Ethereum proves to be a successful attempt that brings smart contracts into a blockchain based system, yet it fails to achieve scalability, that is, smart contracts have to be executed serially in the system, yielding a major problem once there are a large number of smart contracts. In this paper, we focused on the scalability problem.

**Related work on partitioning and load balancing.** We give a brief overview on algorithms for the load balancing problem. In the load balancing problem, we are given a set of $n$ integers $\{a_1, a_2, \cdots, a_n\}$, and the goal is to partition them into $m$ disjoint subsets $S_1$, $S_2$, $\cdots$, $S_m$ such that the ratio between the largest load and the smallest load of subsets is minimized, where the load of a subset is defined as the summation of the integers in this subset. More precisely, the goal is to find a partition such that $\frac{\max_{i=1}^{m} \sum_{j:a_j \in S_i} a_j}{\min_{i=1}^{m} \sum_{j:a_j \in S_i} a_j}$ is minimized.

The load balancing problem is a fundamental problem in computer science and has received much study under different context, e.g., scheduling, resource allocation, bin packing. It is a classical NP-hard problem [16]. Exact and approximation algorithms are presented and analyzed in [19], [9]. Various heuristics are analyzed in [41], [20], [33].

We remark that, in our model, we need to solve a problem more general and harder than the classical load balancing problem. In addition to the integers, we also associate a profit for every integer and require that the total profit of subsets are also balanced. Thus, the sub-problem of partitioning smart contracts in our model falls into the category of bi-objective optimization problems that builds upon the load balancing problem. We adopt the common approach [18] for bi-objective optimization by computing all pareto optimal solutions for the sub-problem.

## IX. Conclusion

We design a scalable smart contract scheme that allows smart contracts to be executed much more efficiently in a blockchain based transaction system. Our scheme collects a set of smart contracts to be executed, and then creates a special smart contract to partition them and assign them to users. Our approach relies on two important techniques – A fair partition of smart contracts based on integer linear programming (ILP), and a random assignment protocol that allows each subset of the produced partition to be assigned randomly among users. Then we evaluate our approach by applying it to the data collected from Ethereum.

A potential limitation of our approach is that it relies on the efficiency of the ILP solver. The current solver, e.g., GUROBI [28] or SCIP [1], is able to solve an ILP with up to 1,000 binary variables in a short time. This means, in certain scenario, our scheme may create a special smart contract whenever it collects 1000 or less smart contracts, otherwise

ILP optimization could take too much time and reduce the time saved by introducing divide-and-conquer. While this may be acceptable in practice, it is desirable that the scheme can work for an even larger set of smart contracts. It is an interesting open problem for future research whether we can achieve the more efficient and scalable partition without resorting to ILP solvers.

## Acknowledgment

This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 2015-ST-061-BSH001. This grant is awarded to the Borders, Trade, and Immigration (BTI) Institute: A DHS Center of Excellence led by the University of Houston, and includes support for the project Secure and Transparent Cargo Supply Chain: Enabling Chain-of-custody with Economical and Privacy Respecting Biometrics, and Blockchain Technology awarded to University of Houston. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

## References

[1] T. Achterberg, "Scip: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[2] B. Awerbuch and C. Scheideler, "Robust random number generation for peer-to-peer systems," in *International Conference On Principles Of Distributed Systems*. Springer, 2006, pp. 275–289.

[3] G. V. Bard, S. V. Ault, and N. T. Courtois, "Statistics of random permutations and the cryptanalysis of periodic block ciphers," *Cryptologia*, vol. 36, no. 3, pp. 240–262, 2012.

[4] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings," in *Proceedings of the 20th ACM Conference on Computer and Communications Security - CCS 2013*. ACM, 2013, pp. 967–980.

[5] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1015, 2015.

[6] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Computer Networks*, vol. 53, no. 13, pp. 2340–2359, 2009.

[7] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, 2014.

[8] D. Chaum, "Blind signatures for untraceable payments," in *Advances in cryptology*. Springer, 1983, pp. 199–203.

[9] L. Chen, K. Jansen, and G. Zhang, "On the optimality of approximation schemes for the classical scheduling problem," in *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, 2014, pp. 657–668.

[10] L. Chen, L. Xu, N. Shah, N. Diallo, Z. Gao, Y. Lu, and W. Shi, "Unraveling blockchain based crypto-currency system supporting oblivious transactions: a formalized approach," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 2017, pp. 23–28.

[11] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "Decentralized execution of smart contracts: Agent model perspective and its implications."

[12] N. T. Courtois, P. Emirdag, and D. A. Nagy, "Could bitcoin transactions be 100x faster?" in *Security and Cryptography (SECRYPT), 2014 11th International Conference on*. IEEE, 2014, pp. 1–6.

[13] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, "Randomness extraction and key derivation using the cbc, cascade and hmac modes," in *Annual International Cryptology Conference*. Springer, 2004, pp. 494–510.

[14] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 45–59.

[15] Z. Gao, L. Xu, L. Chen, N. Shah, Y. Lu, and W. Shi, "Scalable blockchain based smart contract execution (full version)," 2017. [Online]. Available: http://i2c.cs.uh.edu/tiki-download_wiki_attachment.php?attId=72&download=y

[16] M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman New York, 2002, vol. 29.

[17] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," *Advances in Cryptology–CRYPTO 2010*, pp. 465–482, 2010.

[18] S. Greco, J. Figueira, and M. Ehrgott, "Multiple criteria decision analysis," *Springer's International series*, 2005.

[19] K. Jansen, K. Klein, and J. Verschae, "Closing the gap for makespan scheduling via sparsification techniques," in *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, 2016, pp. 72:1–72:13.

[20] R. E. Korf and E. L. Schreiber, "Optimally scheduling small numbers of identical parallel machines." in *ICAPS*, 2013.

[21] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[22] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena, "Scp: A computationally-scalable byzantine consensus protocol for blockchains." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1168, 2015.

[23] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 397–411.

[24] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[25] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[26] *FIPS PUB 197: Advanced Encryption Standard*, NIST Std., November 2001.

[27] T. Okamoto, "A digital multisignature scheme using bijective public-key cryptosystems," *ACM Transactions on Computer Systems (TOCS)*, vol. 6, no. 4, pp. 432–441, 1988.

[28] G. Optimization *et al.*, "Gurobi optimizer reference manual," *Gurobi*, vol. 2, pp. 1–3, 2012.

[29] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption." in *TCC*, vol. 7194. Springer, 2012, pp. 422–439.

[30] M. Raab and A. Steger, "balls into bins - a simple and tight analysis," in *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 1998, pp. 159–170.

[31] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash," in *Annual International Cryptology Conference*. Springer, 1999, pp. 555–572.

[32] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.

[33] E. L. Schreiber and R. E. Korf, "Improved bin completion for optimal bin packing and number partitioning." in *IJCAI*, 2013.

[34] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[35] M. Tibouchi, "Elligator squared: Uniform points on elliptic curves of prime order as uniform random strings," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 139–156.

[36] N. P. Triantafyllidis and T. Oskar van Deventer, "Developing an ethereum blockchain application," 2016.

[37] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[38] L. Xu, L. Chen, Z. Gao, Y. Lu, and W. Shi, "Coc: Secure supply chain management system based on public ledger," in *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 2017, pp. 1–6.

[39] L. Xu, L. Chen, N. Shah, Z. Gao, Y. Lu, and W. Shi, "Dl-bac: Distributed ledger based access control for web applications," in *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 2017, pp. 1445–1450.

[40] L. Xu, N. Shah, L. Chen, N. Diallo, Z. Gao, Y. Lu, and W. Shi, "Enabling the sharing economy: Privacy respecting contract based on public blockchain," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 2017, pp. 15–21.

[41] J. Zhang, K. Mouratidis, and H. H. Pang, "Heuristic algorithms for balanced multi-way number partitioning," 2011.