# YODA: Enabling computationally intensive contracts on blockchains with Byzantine and Selfish nodes

Sourav Das*, Vinay Joseph Ribeiro† and Abhijeet Anand‡
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India
{*souravdas1547, ‡abhijeetanand98765}@gmail.com, †vinay@cse.iitd.ac.in



***Abstract*—One major shortcoming of permissionless blockchains such as Bitcoin and Ethereum is that they are unsuitable for running Computationally Intensive smart Contracts (CICs). This prevents such blockchains from running Machine Learning algorithms, Zero-Knowledge proofs, etc. which may need non-trivial computation.**

**In this paper, we present YODA, which is to the best of our knowledge the first solution for efficient computation of CICs in permissionless blockchains with guarantees for a threat model with both Byzantine and selfish nodes. YODA selects one or more execution sets (ES) via Sortition to execute a particular CIC off-chain. One key innovation is the MultI-Round Adaptive Consensus using Likelihood Estimation (MIRACLE) algorithm based on sequential hypothesis testing. MIRACLE allows the execution sets to be small thus making YODA efficient while ensuring correct CIC execution with high probability. It adapts the number of ES sets automatically depending on the concentration of Byzantine nodes in the system and is optimal in terms of the expected number of ES sets used in certain scenarios. Through a suite of economic incentives and technical mechanisms such as the novel Randomness Inserted Contract Execution (RICE) algorithm, we force selfish nodes to behave honestly. We also prove that the honest behavior of selfish nodes is an approximate Nash Equilibrium. We present the system design and details of YODA and prove the security properties of MIRACLE and RICE. Our prototype implementation built on top of Ethereum demonstrates the ability of YODA to run CICs with orders of magnitude higher gas per unit time as well as total gas requirements than Ethereum currently supports. It also demonstrates the low overheads of RICE.**

## I. Introduction

Permissionless blockchain protocols, which originated with Bitcoin [18], allow an arbitrarily large network of *miners* connected via a peer-to-peer overlay network to agree on the state of a shared ledger. More recent blockchains extend the shared ledger concept to allow programs called *smart contracts* to run on them [6], [25]. Smart contracts maintain state that can be modified by transactions. One of the major shortcomings of these blockchains is that they are unsuitable for smart contracts which require non-trivial computation for execution [7]. We call such smart contracts *Computationally Intensive Contracts* (CIC). CICs can potentially run intensive machine learning algorithms [29], zero-knowledge proofs [4], [9] etc.

One reason for this shortcoming is that every transaction is executed *on-chain*, that is by *all* miners, and this computation must be paid using the transaction fee. Hence CIC transactions require very high transaction fees.[1] A second reason is the Verifier's Dilemma [16]. A miner must normally start mining a new block on an existing block only after verifying all its transactions. If the time taken to verify these transactions is non-trivial, it delays the start of the mining process thereby reducing the chances of the miner creating the next block. Skipping the verification step will save time but at the risk of mining on an invalid block, thereby leaving a rational miner in a dilemma of whether to verify transactions or not.

One mechanism to side-step the Verifier's Dilemma is to break a computationally-heavy transaction into multiple light-weight transactions and spread these out over multiple blocks [16]. This mechanism has several shortcomings. First, the total fees of these transactions may be prohibitively high. Second, how to split a single general purpose transaction into many while ensuring the same resulting ledger state is not obvious. Third, the number of blocks over which the light-weight transactions are spread out grows linearly with the size of the total computation.

Another approach is to execute smart contracts *off-chain*, i.e. by only a subset of *nodes*[2] to cut down transaction fees and avoid the Verifiers Dilemma [12], [26]. The off-chain methods proposed so far, however, work under the limited threat model of nodes being rational and honest but not Byzantine. Moreover, they require on-chain computation of a part of a CIC to determine its correct solution in some cases. Note that off-chain CIC computation is not the same as achieving consensus about blocks using shards [2], [10], [14], [15], [31]. Blocks can in general take many valid values and are computationally easy to verify unlike CIC solutions which have only one correct value and are expensive to verify.

**Our Goal.** Our goal is to design a mechanism for off-chain CIC execution with the following properties.

1) *BAR Threat model.* It should work under a Byzantine, Altruistic, Rational (BAR) model which considers both Byzantine and Rational entities. BAR models are more realistic and challenging to analyze than threat models which consider only one of Byzantine or rational entities [1].
2) *Adaptive to Byzantine fraction.* It should make fewer nodes perform off-chain computation if the fraction of Byzantine nodes is smaller.
3) *Scalability*: CICs are never executed or verified on-chain

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23142
www.ndss-symposium.org

[1]Transaction verification is to some extent subsidized by mining fees.

[2]For clarity, we use the term *node* for an entity performing off-chain computation and the term miner for an entity performing all on-chain computation.

either fully or partially. Further, the number of CICs that can be executed in parallel must scale with increasing number of nodes in the system.

4) *Fair and timely reward.* As CIC execution is expensive, all nodes performing off-chain computation correctly must be compensated fairly and in a timely manner.

**Our Approach.** In this paper we present YODA, which is to the best of our knowledge the first solution for efficient computation of CICs in permissionless blockchains which gives security guarantees in a BAR threat model. The threat model allows at most a fraction $f_{max} < 0.5$ of Byzantine nodes in the overall system and the remaining can be *quasi-honest*. Note that the actual fraction $f$ of Byzantine nodes is unknown a priori and can be anywhere between 0 and $f_{max}$. Although YODA is designed for the worst case ($f = f_{max}$), it adapts to smaller values of $f$, by evaluating CICs more efficiently.

Quasi-honest nodes are selfish nodes which seek to maximize their utility by skipping CIC computation using information about its solutions which may already be published on the blockchain by other nodes. We call this a *free-loading attack*. They may also try to collude with each other to reduce their computation. More details about quasi-honest nodes are given in §II-A. YODA is robust to DoS attacks, Sybil attacks, and ensures timely payouts to all who execute a CIC.

YODA's modus operandi is to make only small sets of randomly selected nodes called Execution Sets (ES) compute the CICs. ES nodes submit their solutions, or just a small digest of them, on the blockchain as transactions. YODA then study the counts of various solutions submitted in order to identify the correct solution from among them. While a small ES improves system efficiency, it can occasionally be dominated by Byzantine nodes which may form a majority and submit incorrect solutions. Hence, a simple majority decision does not work even in a setting with only honest and Byzantine nodes.

To determine the correct CIC solution, YODA uses a novel *MultI-Round Adaptive Consensus using Likelihood Estimation* (MIRACLE) algorithm. In MIRACLE, miners compute the *likelihood* of each received digest which primarily depends on the counts of different digests and the fraction $f$ of Byzantine nodes. If the likelihood of any digest crosses a particular threshold, MIRACLE declares its corresponding solution as the correct one. Otherwise, it iteratively selects additional ES sets until the likelihood of a digest crosses required threshold. We call the selection of each such ES a *round*.[3] MIRACLE is adaptive, that is the expected number of rounds it takes to terminate is smaller the smaller $f$ is. MIRACLE guarantees selection of the correct digest with probability at least $1 - \beta$ for a design parameter $\beta$. Moreover, for the special case of $f = f_{max}$, if all Byzantine node submit the same incorrect digest, MIRACLE optimally minimizes the expected number of rounds. Interestingly, the strategy for Byzantine nodes to make MIRACLE accept an incorrect solution with highest probability is to submit the same incorrect solution (refer VII-A).

This analysis for MIRACLE, however, assumes that all quasi-honest nodes submit correct solution. Since MIRACLE itself does not enforce honest behaviour, other mechanisms are necessary to make quasi-honest nodes submit correct solutions. Without additional mechanisms, a quasi-honest node may be tempted to free-load on solutions already submitted in earlier rounds, thus saving on computational power. In case quasi-honest nodes free-load on incorrect solutions, MIRACLE has a higher probability of terminating with an incorrect solution.

To mitigate the free-loading attack of quasi-Honest nodes, we design the Randomness Inserted Contract Execution (RICE), an efficient procedure to change the digest from one round to the next. We achieve this by making the digest dependent on a set of pseudo-randomly chosen intermediate states of a CIC execution. This ensures, that despite digests changing from one round to the next, the miners running MIRACLE are able to map digests from different rounds to the same CIC state they represent. We prove that RICE adds little computational overhead to CIC execution. To be precise, if $T$ denotes the total computation for a transaction execution without RICE, then RICE adds computation overhead of $O((log_2 T)^2)$. In the presence of free-loading attacks, we show via a game theoretic analysis that honest behavior from all quasi-honest nodes is an $\epsilon-$Nash equilibrium with $\epsilon \geq 0$.

We have implemented YODA with MIRACLE and RICE, in Ethereum as a proof-of-concept and provide many experimental results supporting our theoretical claims.

## II. Theat Model, Assumptions and Challenges

In YODA, a blockchain is an append-only distributed ledger consisting of data elements called blocks. A blockchain starts with a pre-defined genesis block. Every subsequent block contains a hash pointer to the previous block resulting in a structure resembling a chain. The blockchain contains accounts with balances, smart contracts, and transactions. A transaction is a signed message broadcast by a account owner which can be included in a block provided it satisfies certain validity rules. For example, transactions modifying an account balance must be signed by the corresponding private key to be valid. YODA assumes that the underlying blockchain provides guarantees about its Safety and Availability. *Safety* means that all smart contract codes are executed correctly on-chain, and *availability* means that all transactions sent to the blockchain get included in it within bounded delay and cannot be removed thereafter.

We refer to any entity performing off-chain CIC execution in YODA as a *node*. We call the set consisting of all nodes the *Stake Pool* (SP).[4] Without loss of generality each node in SP controls an account in the ledger with its private key. The account itself is identified by the public key. We assume that the network is synchronous, i.e., transactions broadcast by nodes get delivered within a known bounded delay. However, unlike [15] we do not assume the existence of an overlay network among nodes. Also, we do not assume the presence of a secure broadcast channel or a PKI system. We abstract the source of randomness required for RICE to a function $\mathsf{RandomGen}()$ (given in §VI) which can be accessed by all nodes in YODA. This can be built as a part of YODA or as an external source using techniques from [10], [15], [24].

For the rest of the paper, unless otherwise stated, if some event has *negligible* probability, it means it happens with probability at most $O(1/2^{\lambda})$ for some security parameter $\lambda$. Any event whose complement occurs with negligible probability is

[3]Rounds are different from block-generation epochs and are specific to CICs. A round may span multiple blocks.

[4]The choice of the name will be discussed when we discuss blockchain specifics.

said to occur with high probability or $w.h.p.$

### A. Threat Model and Assumptions.

Systems like permissionless blockchains cannot be assumed to have all honest nodes. They rely heavily on incentives and the rationality of nodes in order to work correctly. Rational nodes are those which seek to maximize their utility. However, assuming that all nodes are rational is not practical either. Real systems may contain Byzantine nodes, that is those which do not care about their returns.

We consider two kinds of nodes: *Byzantine* and *quasi-Honest*. Byzantine nodes are controlled by an *adversary* and these nodes can deviate arbitrarily from the YODA protocol. The adversary can make all Byzantine nodes collude with perfect clock synchrony. They can add or drop messages arbitrarily and not execute CICs correctly. We assume that at most $f_{max} < \frac{1}{2}$ fraction of nodes in SP are Byzantine. Additionally the adversary has state information of the CIC from all previous rounds and can successfully communicate this (potentially false) state information about previous rounds to any node in SP. However, we assume cryptographic primitives are computationally secure.

Modeling rational nodes in these systems, taking into account all possible means of profits, costs, and attacks is non-trivial and is beyond the scope of the paper. However to bring our model close to reality we work with quasi-honest nodes which deviate from the protocol in the following manner.

**Quasi-Honest.** Quasi-honest nodes will skip execution of a CIC either completely or partially, for example by not executing some of its instructions, if and only if the expected reward in doing so is more than that for executing the transaction faithfully. They do not share information with any other node if that information can lead to reduction of their reward. They are *conservative* when estimating the potential impact of Byzantine adversaries in the system, i.e. a quasi-honest node while computing its utility assumes that the Byzantine adversary acts towards minimizing its (quasi-honest node's) rewards [1].

Quasi-honest nodes may skip computation using one of two methods. The first is "free-loading" where they attempt to guess the correct state of a CIC after execution of a transaction from the information of the corresponding transaction already published on the blockchain. Free-loading also includes the case where a quasi-honest node tries to guess the state when an adversary presents the pre-image of hashes among this information already published on the blockchain.

The second is by colluding with other ES nodes of the same round to submit an identical CIC solution without evaluating it. A quasi-honest node only colludes with nodes whose membership in the ES it can verify. YODA has checks (refer § VI ) which prevent nodes from directly proving their ES membership. Hence nodes must use Zero-Knowledge-Proof techniques like zk-SNARK [4] to establish their membership in ES. YODA allows use of smart-contracts as shown in [11] to establish rules of collusion. However we assume that a quasi-Honest node does not know for sure if the node it is colluding with is quasi-honest or Byzantine. Additionally, both free-loading and collusion have costs associated with them. These cost are due to processing of information available on the blockchain or received from peers, producing and verifying zk-Proofs, bandwidth and computation costs etc. In case neither free-loading nor collusion gives a better expected reward than executing CICs correctly, a quasi-honest node will execute the CIC correctly.

### B. Challenges

Enabling off-chain execution of CICs in the presence of a Byzantine adversary is fraught with many challenges. Allowing non-Byzantine nodes to deviate from the protocol makes the problem more interesting and even more challenging. Apart from recently studied challenges like preventing *Sybils* [18], [31] and generating an unbiased source of randomness in the distributed setting [10], [15], [24], our system must tackle the following challenges:

- to prevent quasi-honest nodes from *Free-loading and collusion*.
- since the size of any ES is small, an ES becomes vulnerable to *Lower cost DoS Attacks* than a DoS attack on the set of all nodes taken together.
- to provide guarantees of correctness without requiring re-execution of any part of the CIC on-chain.

## III. MIRACLE: MULTI-ROUND ADAPTIVE CONSENSUS USING LIKELIHOOD ESTIMATION

In this section we describe MIRACLE as an abstract consensus protocol and later get into its blockchain specifics.

**Problem Definitions.** Let $\Psi$ be a deterministic function that when given arbitrary input $x$ produces output $y$. We denote this as $y \leftarrow \Psi(x)$.[5] Let SP contain at most $f_{max}$ fraction of Byzantine nodes. All other nodes are honest, i.e. they strictly adhere to the protocol. Let $n_i$ be a node in SP where $i = 1, 2, \ldots, |SP|$. Let $\Psi_i$ be the function $n_i$ executes when asked to execute $\Psi$ and let $y_i \leftarrow \Psi_i(x)$ be the corresponding result. For all honest nodes, clearly $\Psi_i = \Psi$.

Our goal is to achieve consensus on the true value of $\Psi(x)$ by making only one or more small randomly chosen subsets called Execution Sets (ES) of nodes evaluate $\Psi(x)$. Further, nodes $n_i\ \forall i$ after executing $\Psi_i(x)$ broadcast a *digest* of $y_i$, say $\mathsf{hash}(y_i)$ to all other nodes. MIRACLE proceeds in *rounds* where in each round a new ES is selected. We require MIRACLE to correctly reach consensus on $\Psi(x)$ with probability greater than $1 - \beta$ for any given user-specified parameter $\beta$, given $f_{max}$ and $\mathbb{E}[|ES|]$, while minimizing the expected number of rounds to terminate. Formally, MIRACLE must guarantee the following properties.

- **(Termination)** For any $f_{max} < 1/2$, MIRACLE must terminate within a finite number of rounds.
- **(Agreement)** All honest nodes in SP, agree on the result that MIRACLE returns on terminating.
- **(Validity)** MIRACLE must achieve consensus on the true value of $\Psi(x)$ with probability $1 - \beta$.
- **(Efficiency)** When the fraction of Byzantine nodes is $f_{max}$ and given a particular $\mathbb{E}[|ES|]$, MIRACLE must terminate in the optimal number of rounds. Further, for any given $f \leq f_{max}$, if $N_f$ denotes the expected number of nodes performing off-chain execution then $N_f \leq N_{f_{max}}$

[5] We use $\leftarrow$ for function executions, with the function and its inputs on its right and the returned value on its left.

### A. Overview and Simplistic algorithms

We motivate MIRACLE by describing two simplistic algorithms for achieving consensus regarding $\Psi(x)$. In these two algorithms each node in SP belongs to a particular ES with probability $q$ independent of other nodes, thus $\mathbb{E}[|ES|] = q|SP|$. Note that MIRACLE in general need not have the same $\mathbb{E}[|ES|]$ in every round, although in this paper we present a version which does. Studying other possible MIRACLE algorithms is part of our future work.

**Naive Solution 1 (NS1):** Suppose we use a single subset ES from SP to compute $\Psi(x)$. If more than 50% of nodes in the ES publish the same execution result then this is chosen as $\Psi(x)$. One shortcoming of this scheme is that for lower $\beta$ (more security), the size of ES must be a large fraction of SP. A second shortcoming is that if the actual fraction of Byzantine nodes $f$ is much smaller than $f_{max}$ then we end up using an ES much larger than required. For example, with $\beta = 10^{-20}$ as the error, starting with $|SP| = 1600$ and $f_{max} = 0.35$, NS1 will always pick $|ES| \approx 900$ independent of $f$.

**Naive Solution 2 (NS2):** In this solution we relax the requirement of achieving consensus in one round. If in an ES, the fraction of nodes submitting a particular solution exceeds some threshold then we terminate with that solution. This threshold should be high enough to ensure the correct solution $w.h.p.$ In NS1 for example, the threshold is $1/2$. In general, the smaller $q$ is the larger will the threshold be. If we do not reach consensus then a new round is triggered.

The advantage is that we can use an ES in each round of size smaller than the ES used in NS1. In NS2, in certain instances such as when $f = 0$, a single round may still be sufficient to reach consensus. One shortcoming is that the number of rounds to terminate can be large because NS2 does not optimally combine the results of all rounds in order to reach consensus. Results of one round are forgotten in future rounds.

### B. Design and Algorithm

In MIRACLE, we employ the multi-round strategy of NS2 to achieve gains in case $f \ll f_{max}$. In contrast to NS2, each round uses all hitherto published results to decide whether to terminate or not. For a given $\Psi(x)$, let $d_1, d_2, ..., d_m$ be the $m$ unique digest values broadcast up to and including the $i^{th}$ round. Let $c_{k,i}$ denote the number of times $d_k$ is repeated in the $i^{th}$ round. Let $C_i$ denote the total number of submissions (ES nodes) in the $i^{th}$ round, i.e. $C_i = \sum_{k=1}^{m} c_{k,i}$.

The problem we are addressing is to decide among one of may solutions broadcast. We present a novel model of this problem as a multiple hypothesis testing problem where we have one hypothesis for each solution submitted and the test must decide which hypothesis is true.

**Primer on Hypothesis Testing.** For the reader unfamiliar with Hypothesis testing, we now describe a standard example. Consider a communication system in which a source is transmitting one symbol selected from a known small master set to a receiver over a noisy channel. In the simplest case, only two symbols are allowed, one each for communicating bit 0 and bit 1. The receiver's task is to decide which symbol (and hence which corresponding bit(s)) was transmitted given the observation. To solve the problem, one proposes a hypothesis for each potential symbol which claims that the corresponding symbol was transmitted. The goal is to determine which hypothesis is true. To do so, the receiver computes the probability of the observation conditioned on every hypothesis being true. Only if one of these probabilities is much larger than the others can one say with confidence that the corresponding hypothesis is true with high probability.

One of our novel contributions in MIRACLE is to formulate the problem of determining the true $\Psi(x)$ as a hypothesis testing problem. This is not obvious because traditional hypothesis tests are designed to handle real-world phenomena such as signals in noise. In our problem we have an intelligent adversary which is hard to model as there is no restriction on what solution it can submit. It can submit any of $2^n$ digests if the digest is $n$ bits long. Hence unlike the communication problem described above, there is no small master set of potential correct solutions known a priori to YODA.

However, in the worst case when the fraction of Byzantine nodes is maximum, i.e. $f = f_{max}$, we do have a probability distribution on the *total number of Byzantine nodes* in an ES. Similarly we have a probability distribution of the total number of quasi-honest nodes in an ES. These probability distributions are sufficient for us to compute a likelihood and perform a hypothesis test. In the case the adversary submits only a single incorrect solution, MIRACLE is optimal in the number of rounds it takes to converge. However, if it submits many solutions then our assumed distributions for different hypotheses are not exact. Fortunately, if the adversary submits more than one solution, it is to his own detriment as MIRACLE will converge to the correct solution with higher probability than if it submitted only a single solution.

MIRACLE uses multiple parallel Sequential Probability Ratio Tests (SPRT) to choose the correct solution [28] whose details are given next.

MIRACLE **as Parallel SPRT:** We model the problem as $m$ simultaneous two-hypotheses Sequential Probability Ratio Tests (SPRT) [28]. The $k^{\text{th}}$ SPRT is given by:

- Null Hypothesis, $\mathcal{H}_k : d_k$ is the solution.
- Alternative Hypothesis, $\mathcal{H}_k^* : d_k$ is not the solution.

The log-likelihood ratio is defined as the log of the ratio of probabilities of the observations ($c_{k,i}$) conditioned on the two hypotheses. We approximate this log-likelihood ratio after $i$ rounds by a quantity we loosely call the *likelihood*. We denote it by $L_{k,i}$, and proceed as follows. We give a formula for the likelihood subsequently. For appropriately chosen threshold $\mathbb{T}$, in round $i$ we perform

- If $L_{k,i} \leq \mathbb{T}$, make no decision.
- If $L_{k,i} > \mathbb{T}$, decide in favor of $\mathcal{H}_k$.

When any one SPRT, say the $k^{\text{th}}$, terminates in favor of its Null Hypothesis $\mathcal{H}_k$, we halt all other SPRTs and declare $d_k$ as the digest. If no SPRT terminates, we proceed to the next round. Algorithm 1 demonstrates the general MIRACLE algorithm for any given $L_{k,i}$ and $\mathbb{T}$.

MIRACLE **and YODA.** We now present our specific choices for the likelihood $L_{k,i}$ and threshold $T$ which we use in YODA. MIRACLE can in general use other choices for the same quantities, and such generalisations are part of future work.

Recall that nodes are selected randomly and with the same

**Algorithm 1** MIRACLE

1: $i \leftarrow 1$
2: **while** $L_{k,i} \leq \mathbb{T}\ \forall k$ **do**
3: $\quad i \leftarrow i+1$
4: $\quad$ Pick next ES to execute $\Psi(x)$
5: **end while**
6: **declare** $d_{k'}$ to be correct where $L_{k',i} > \mathbb{T}$

probability $q$ for any ES. We set

$$L_{k,i} = \sum_{j=1}^{i} \left(c_{k,j}^2 - (C_j - c_{k,j})^2\right) = \sum_{j=1}^{i} \left((2c_{k,j} - C_j)C_j\right) \tag{1}$$

and the threshold to:

$$\mathbb{T} = \left(\ln \frac{1-\beta}{\beta}\right) \frac{2q(1-q)M(1-f_{max})f_{max}}{(1-f_{max}) - f_{max}}. \tag{2}$$

The above choice of $L_{k,i}$ is indeed the log-likelihood ratio when the adversary submits a single incorrect solution in all rounds, assuming a Gaussian distribution for the number of quasi-honest and Byzantine nodes in any ES. Under the same assumptions, in Section VII-A we describe how this choice of threshold gives an optimal solution in terms of number of rounds to terminate along with required security.

## IV. RICE: Randomness Inserted Contract Execution

MIRACLE by itself does not force quasi-honest nodes to behave honestly. In fact, a *free-loading attack* by quasi-honest node is a real possibility. Here quasi-honest nodes in an ES of one round may simply replay the digest with highest likelihood of previous submissions in earlier rounds. Even though this enables a quasi-honest node to save on heavy CIC computation, this attack can make increase the probability of accepting an incorrect solution to larger than $\beta$. Specifically, in scenarios where the corresponding digest is an incorrect solution, as an adversary can sometime dominate a large fraction in ES, free-loading can lead to acceptance of an incorrect solution.

To address this in this section we describe Randomness Inserted Contract Execution (RICE), a procedure to pseudo-randomly change the digest of $\Psi(x)$ from one round to the next to mitigate the free-loading problem. Other attacks, such as collusion of quasi-honest nodes within the same ES and copying digests submitted by nodes in the same round are addressed in §VI.

So far we have looked at $\Psi$ as a very abstract function without describing any of its details. We now formally define the semantics of $\Psi$ required to understand RICE.

### A. Design of RICE

**Setup.** We assume $\Psi$ to be a stateful function similar to a smart contract. Let $\sigma$ be the state on which $\Psi$ operates by taking an input $x$. The output of $\Psi(\sigma, x)$ is the modified state $\sigma^*$. Call $root(\sigma)$ (or simply $root$) a unique digest of $\sigma$. For example, $root$ can represent the root of a Merkle tree where leaves of the tree correspond to the contents of $\sigma$.

Let $j$ ($\geq 1$) be the MIRACLE round number. We wish to generate a pseudorandom digest in each round. At the same time, to compute likelihoods, we must be able to map digests across different rounds to each other. To solve the paradox, we generate a digest $(seed^{(j,.)}, root)$, where $seed^{(j,.)}$ is a pseudorandom number which changes from one round to the next. The $seed$ is initialized as:

$$seed^{(j,0)} \leftarrow \begin{cases} \mathsf{RandomGen}() & \text{if } j = 1 \\ \mathsf{hash}(seed^{(j-1,0)}) & \text{otherwise} \end{cases} \tag{3}$$

**Array Model for RICE.** Consider an execution model in which all machine level instructions that $\Psi(\sigma, x)$ executes are stored in an imaginary "instruction array", that is the $i^{\text{th}}$ instruction executed is stored in the $i^{\text{th}}$ array position. RICE then interrupts execution[6] of $\Psi(\sigma, x)$ at certain intermediate indices of the array where state of the CIC is $\sigma'$ and updates the $seed$ as follows:

$$seed^{(j,l+1)} \leftarrow \mathsf{hash}(seed^{(j,l)} || root(\sigma')) \tag{4}$$

By choosing these different indices pseudorandomly in different rounds, RICE produces a different $digest$ every round. ES nodes then submits $(seed^{(j,\phi)}, root(\sigma^*))$ as the $digest$ after executing $\Psi(\sigma, x)$, where $\phi$ is the total number of times the $seed$ has been updated.

Due to the deterministic nature of the $\Psi$, all nodes computing $\Psi(\sigma, x)$ correctly will have the same $root$ across rounds. The $seed$ values of all honest nodes will be identical within any round, but will differ from one round to the next. Malicious nodes may submit the correct $root$ but the wrong $digest$, an attack we guard against in §VI

**Details.** Let $t$ denote the indices in the instruction array where $t \in [1 : T]$ where $T$ is the total number of instructions executed as a part of $\Psi(\sigma, x)$. Note that $T$ is unknown a priori, but assumed to be bounded. Specifically, for systems such as Ethereum where CIC transactions have a gas limit, $T$ is guaranteed to be bounded (refer §VI). Thus to update $digest$, instead of executing the entire $\Psi$ array in a single run, RICE progressively executes a subarray of $\Psi$ array between two index $t_i$ (initial) and $t_f$ (final), updates the $seed$, and repeats the process with the next sub-array and so on until it reaches $T$.

Formally, let $\Psi[t_i : t_f]$ denote an arbitrary subarray from $\Psi$ with $t_i$ and $t_f$ its initial and final index. RICE consists of a new deterministic contract execution function $\Psi'$ with the following semantics. Inputs to $\Psi'$ are two indices $t_i, t_f$, an intermediate CIC state $\sigma'$ and $x$. Given input $(t_i, t_f, \sigma', x)$, $\Psi'$ executes subarray $\Psi[t_i : t_f]$ (both $t_i, t_f$ inclusive) with state $\sigma'$ and data $x$. After execution, $\Psi'$ returns a modified state and the last successfully executed index. In the special case where $T < t_f$ for some $(t_i, t_f)$, $\Psi'(t_i, t_f, \sigma', x)$ runs only till $\Psi[t_i : T]$ and returns $\sigma^*, T$ as its output. Formally,

$$\left.\begin{array}{l}(\sigma', t_f), \text{ if } t_f < T \\ (\sigma^*, T), \text{ if } t_f \geq T\end{array}\right\} \leftarrow \Psi'(t_i, t_f, \sigma', x), \text{ where } \sigma^* = \Psi(\sigma, x)$$

After executing $(l+1)^{\text{th}}$ subarray of $\Psi(\sigma, x)$, RICE updates the seed via (4).

Algorithm 2 gives the pseudocode of RICE.

[6] Blockchains such as Ethereum count gas used after each instruction. Hence additional interrupts are not required for Ethereum-like blockchains.

**Algorithm 2** RICE

**input** $seed^{(j,0)}, \sigma, x$
$\sigma' \leftarrow \sigma, l \leftarrow 0$
$t_i, t_f \leftarrow$ NEXT ▷ Next subarray indices
**while** true **do**
  $\sigma', t_l \leftarrow \Psi'(\sigma', t_i, t_f, x)$
  **if** $t_l$ is $T$ **then**
    **return** $(seed^{(j,l)}, root(\sigma'))$
  **end if**
  $seed^{(j,l+1)} \leftarrow \mathsf{hash}(seed^{(j,l)}||root(\sigma'))$
  $l \leftarrow l + 1$
  $t_i, t_f \leftarrow$ NEXT
**end while**

**Algorithm 3** Next subarray indices

**Let** $K = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, \ldots]$
**procedure** NEXT($seed, \sigma, index, t_f$)
  $k \leftarrow K[index]$
  $pivot \leftarrow t_f - \mathsf{int}(seed[1:k]) + 2^k$
  $nextSeed \leftarrow \mathsf{hash}(seed||root(\sigma'))$
  $t_i \leftarrow t_f + 1$
  $t_f \leftarrow pivot + \mathsf{int}(nextSeed[1:k])$
  **return** $t_i, t_f$
**end procedure**

*B. Choosing the indices.*

A naive strategy is to choose indices $t_f$ as multiples of a fixed number, say $\Delta$. Note that $\Delta$ cannot be a function of $T$ which is not known prior to computing $\Psi$. This strategy leads to overheads of $O(T)$.

Another problem arises because indices do not change from one round to the next. Suppose a quasi-honest node of the current wants to free-load the $root$ values at these indices from an earlier round. It can ask any node from an earlier ES to provide these root values and use (4) to verify that they indeed corresponded to the $digest$ submitted by that node, thereby giving it confidence that these root values are correct. It can then reuse these root values to create its own digest without performing the computation. In case the ES node queried is malicious, the quasi-honest node will submit an incorrect solution.

A second naive strategy is to choose the sub-array sizes randomly but with mean size exponentially increasing as $\Psi$ progresses. For example, choose $t_f - t_i$ randomly from $[2^k : 2^{k+1}]$ where $k$ increments by 1 from one sub-array to the next. On the positive side, this will lead to $O(log_2 T)$ seed updates (and consequently overheads of that order) and also will produce a different set of indices from one round to the next with large probability.

However, there remains the problem of skipping computing the last sub-array of the instruction array. Suppose a quasi-honest node in the current round's ES has learned the value of $T$ from ES nodes of earlier rounds. It can perform computation of $\Psi$ for all sub-arrays except for the last one. Then it can use a value of $root$ submitted in an earlier round in (4) to obtain the final $seed$ value, without computing the last sub-array. For this strategy the last sub-array can be as large as $T/2$, leading to nodes skipping as much as half of the computation. Hence although overheads have reduced to $O(log_2 T)$, the computation skipped at the end is $O(T)$. We seek to find a sweet spot between the two with our choice of indices for RICE.

**Our Approach.** RICE uses a hybrid of the two index locating procedures described above. The idea is to divide the array $\Psi'$ into sub-arrays of size $2^k$ where $k = 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, \ldots$. In other words, every value of $k$ repeats $k$ times. Thus, like the second naive scheme the sub-array size increase, but much more gradually (sub-exponentially) so that the last sub-array which might be skipped is smaller. More precisely, for a sub-array of size $2^k$ we choose the index to update the $seed$ as $\mathsf{int}(seed[1:k])$ away from the beginning of the sub-array, where $\mathsf{int}(seed[1:k])$ denotes the integer whose binary representation is identical to the first $k$ bits of $seed$. Algorithm 3 demonstrates our approach.

## V. CIC Preliminaries

**Smart Contracts and its Execution.** A smart contract in YODA is denoted by its state $\sigma = (cid, code, storage)$. Here $cid$ denotes its immutable globally unique cryptographic identity, and $code$ represents its immutable program logic consisting of functions. The state can be modified by a transaction invoking its code and its execution can only begin at a function. Functions may accept data from sources external to the blockchain which must be included in the transactions invoking them. In YODA smart contracts are stateful and state is maintained as $(key, value)$ pairs which together we refer to as $storage$.

A transaction $\tau$ in YODA is the tuple $(tid, funId, data, \xi)$. Here $tid$ is a globally unique transaction identity and $funId$ is the function it invokes. All external inputs required for the function are part of $data$ and $\xi$ consists of meta-information about the account that generated the transactions along with a cryptographic proof of its authenticity. Hereafter we assume all transactions are validated using $\xi$ before being included in a block and hence we drop $\xi$.

Executions of functions in YODA are modeled as *transaction driven state transitions*. We use $\Psi$ to denote a *Deterministic State Transition Machine* (Possibly Turing complete). Formally, we denote this as $\sigma^* \leftarrow \Psi(\sigma, \tau)$ where $\sigma^*$ is the state of the contract after executing $\Psi$.

**Intensive Transactions.** Intensive Transactions (IT) are transactions which cannot be executed on-chain due to either of two problems: its execution time exceeds the typical interspacing between blocks, or it competes with PoW time (the Verifier's Dilemma). The first problem can occur in permissioned ledgers such as Hyperledger, Quorum, R3-Corda etc., and both problems in permissionless blockchains such as Ethereum and Bitcoin. The exact definition of an IT will depend on parameters of the blockchain system under consideration. Transactions which are not ITs are called *non-ITs*.

We give one example of a IT for Ethereum using the concept of $gas$, a measure of cost of program execution [6]. Ethereum associates a fixed cost with each machine level instruction that a smart contract executes and enforces the constraint that all transactions included in a block can consume a maximum combined gas of *blockGasLimit* which is set to prevent the Verifier's Dilemma. Every time a transaction $\tau$ is broadcast, its

creator specifies $\tau.gasLimit$, an upper bound on the gas it is expected to consume. Clearly $\tau.gasLimit < blockGasLimit$ for any transaction to be included in a block. Transactions which violate this condition are thus ITs.

**Computationally Intensive Contracts.** We term all smart contracts that execute ITs as Computationally Intensive Contracts. Since YODA allows transactions of different CICs as well as on-chain transactions to run in parallel we make some assumptions as well as provide special mechanisms to prevent race conditions from occurring. First, we assume that transactions of CIC that are run off-chain cannot modify the storage of any other smart contract whether CIC or non-CIC. Second, we assume that on-chain transactions cannot modify the state of any CIC if any off-chain transaction execution of the CIC is in progress. Third, we ensure that all transactions of all CICs are ordered on-chain before their execution by a special on-chain smart contract called the *Master Contract*[7] (MC).

**Master Contract.** The MC maintains a queue $Q_\sigma$ in which all transactions (ITs and non-ITs) of CIC $\sigma$ are stored before being executed. The transaction at the queue's head is executed first and a new incoming transactions added to its tail. In case a non-IT is at the head of $Q_\sigma$, miners execute the transaction on-chain otherwise the transaction is executed off-chain.

The MC performs many tasks in YODA. Since each CIC has its own queue, ITs of different CICs can be executed in parallel off-chain. In addition, the MC embodies the rules of YODA like creating CICs, ordering their transactions and initiating off-chain execution, running MIRACLE, distributing rewards to the ES node, enabling YODA nodes to join SP by collecting their deposits etc.

**Stake Pool.** YODA prevents sybil entries in SP [8]. To join SP, a node needs to deposit stake $\mathcal{D}_{sp}$ by sending a transaction to MC. This deposit acts as insurance for misbehavior of SP nodes. The SP once selected remains valid for a system defined interval of time denoted by $\delta_{sp}$ beyond which YODA re-initiates the SP selection procedure. Note that SP could potentially include all miners in the entire network, especially in a small blockchain network maintained by several hundred nodes such as blockchains that are built using Hyperledger [3].

**Execution Set.** YODA selects a subset of nodes from SP known as the Execution Set (ES) to execute $\Psi(\sigma, \tau)$.

**CIC creation and deployment.** To deploy a CIC with state $\sigma$ on the blockchain, anyone can broadcast a transaction requesting creation of CIC containing the tuple $(code, storage)$. Miners use $\mathsf{RandomGen}()$ to generate a unique identity $cid$ for the CIC. An entry is registered in the Master Contract (MC) corresponding to the new CIC along with an empty transaction queue $Q_\sigma$. Then miners deploy $\sigma$ on the blockchain like any other smart-contract.

**External Functions.** The following functions are used in YODA and are assumed to be accessible to all nodes.

- $(o, \pi) \leftarrow \mathsf{CheckSort}(pk, sk, nonce, threshold)$. This function on invocation internally runs Secret Cryptographic Sortition (SCS) [10]. The $threshold$ is used to set the probability $q$ that an SP node is in ES. If $\mathsf{CheckSort}()$ returns $\perp$ it implies the node was not selected. Otherwise the node is selected and $o$ is indistinguishable from a truly random number to anyone without $sk$ [17]. However, it is easy to prove that $\mathsf{CheckSort}() \neq \perp$, given $\pi$ and $pk$.
- $r \leftarrow \mathsf{RandomGen}()$ on invocation produces an unbiased distributed random string. It can be practically built using the Randhound protocol given in [24] or using block headers of a sufficiently long set of blocks. A simple and efficient block nonce generation procedure is using $k$ historical block hashes for sufficiently large $k$ as in [10], [15]. Alternatively, one can use a NIST randomness beacon relayed through a data feeding mechanism such as Town Crier [32].

[7]Systems can be built where all rules in MC are part of the basic System protocol instead of making it a smart contract. Our implementation makes MC a smart contract.

## VI. Enabling CICs in Blockchain

In this section we describe how on receiving an IT $\tau$, YODA executes it off-chain. We have described two of YODA's key ingredients in detail: MIRACLE, which enables efficient CIC computation with small sets of nodes, and RICE which makes guessing the seed of one round difficult from submitted digests in earlier rounds. The other mechanisms we describe here address the other challenges mentioned in §II-B, such as preventing sybil attacks, collusion, DoS, and certain variants of free-loading attacks.

**S1. CIC Transaction Deployment.** On receiving an IT, $\tau$, miners generate string $nonce$ using $\mathsf{RandomGen}()$ which is used in $\mathsf{CheckSort}()$ to elect an ES. It is important that the $nonce$ be created *during or only after* inclusion of $\tau$ on-chain. Otherwise, if the $nonce$ is known a priori, the node generating $\tau$ can perform the following attack to dominate the ES formed. It can enroll with key-pairs $(pk, sk)$ in SP such that the Sortition Check §VI results for the key-pairs will guarantee it a large membership in ES. It then broadcasts the transaction, dominates the resulting ES, and submits false solutions which may be accepted.

The creator of $\tau$ deposits $(\mathcal{D}_{min} + gasPrice * gasLimit)$ in the MC where $\mathcal{D}_{min}$ is the minimum amount to pay for the fixed costs of S4 and S5, and $gasPrice$ and $gasLimit$ denote the gas price and gas limit respectively specified by $\tau$. Any extra deposit after execution of $\tau$ is refunded.

**S2. Sortition Check and RICE of CICs.** Nodes decide if they are in ES corresponding to $\tau$ by computing (5) given below. The node is part of the ES if and only if its $sort\_res \neq \perp$. All nodes selected for the ES then execute the corresponding CIC in RICE as given in §IV, generate the corresponding RICE digest and proceed to S3.

$$sort\_res \leftarrow \mathsf{CheckSort}((pk, sk), \tau.nonce, threshold) \quad (5)$$

By using SCS, YODA prevents Byzantine nodes from joining an ES at will. Also SCS protects ES nodes from DoS attacks since their selection is secret until they reveal the fact. Because YODA uses a commit-reveal mechanism (see below), ES nodes are not vulnerable to DoS attacks before they submit their commit transaction. After the commit step, ES nodes may be easier to identify and hence the DoS attack can be more effective. In the second "reveal" step, an ES node broadcasts a single on-chain transaction after a certain number of blocks have been generated. We assume that a node is sufficiently DoS resilient to be able to receive block headers in a timely manner and also to broadcast a small transaction.

**S3. Commitment and Release.** Let $digest_k$, and $sort\_res_k$ denote the digest of $\Psi(\sigma, \tau)$, and the result of $\mathsf{CheckSort}()$

respectively for node $n_k \in ES$ for $k = 1, 2, \ldots, |ES|$. The commitment $n_k$ generates is $se_k$ and is given by

$$se_k \leftarrow \mathsf{hash}(digest_k || sort\_res_k)$$

Assuming existence of a VRF and Ideal Hashing, $w.h.p$ $sort\_res_k \neq sort\_res_i$ for $i \neq k$ and hence $se_i \neq se_k$ even if $digest_k = digest_i$. Nodes in ES then broadcast $se_k$ to the blockchain as a transaction which miners include on-chain if the node is in SP.

ES nodes must broadcast their commitment within a time window $w_{src}$ or commitment period starting from the block that includes $\tau$. Window $w_{src}$ is transaction dependent, recorded in MC and measured in number of blocks. It is set based on the gas limit mentioned in $\tau$ as by design $\Psi(\sigma, \tau)$ will run for at most $\tau.gasLimit$ instructions.

A node is required to keep $sort\_res_k$ secret during this period and forfeits its deposit if it fails to do so. This deters ES nodes from colluding. However, as mentioned in §II-A, ES nodes can use of ZK-proofs to prove that they are in an ES without revealing $sort\_res_k$. We perform a game theoretic analysis of such an attack in §VII-D.

After $w_{src}$, nodes in ES wait for a buffer period $w_{buf}$ before sending their unhashed digests and sortition results to the blockchain. Nodes in ES which have submitted commitments earlier, are required to submit a transaction containing $(digest_k || sort\_res_k)$ within a time window $w_{sr}$ or release period and failure to do so results in their forfeiting deposits and being removed from SP.

The reason for keeping this buffer period of length $w_{buf}$ is to prevent an adversary from launching a DoS attack which we term the *Chain Forking Attack* (CFA).[8] CFA can occur in blockchains where block creation does not guarantee block finalization and nodes need to wait for certain number of blocks before becoming certain about a block's finality $w.h.p$. Assume the absence of this buffer period. If this buffer did not exist then if an honest node publishes its opened commitment after $w_{src}$ and expects its inclusion in a future block, an adversary can create an alternate chain where it includes this transaction before the end of $w_{src}$ and can thereby penalizes the honest node. To prevent this from happening, the introduction of $w_{buf}$ between $w_{src}$ and $w_{sr}$ ensures that the attacker will have to create a fork which is long enough to be prohibitively expensive to create.

**S4.** MIRACLE **for CICs.** The blockchain miners then execute one round of MIRACLE using the submitted digests. All digests with the same $root$ are considered by MIRACLE to be the same solution irrespective of which $seed$ they contain. Steps S2-S4 are repeated if necessary till MIRACLE converges.

**S5. State Update, Reward Distribution and Cleanup.** Once MIRACLE terminates, all nodes in the ES broadcast one or more transactions to the blockchain miners containing information required for updating the state of the CIC to $\sigma^*$, the state corresponding to the winning digest along with corresponding proof.[9] Any miner, on receiving such a transaction validates it using the root contained in the winning $digest$ in MIRACLE and then gossips it to other miners. To avoid flooding the network, it does not gossip any other transactions about the same state.

To disincentivize nodes from submitting either false $seed$ or $root$ values, YODA rewards ES nodes as follows. Let $I$ denote the round in which MIRACLE terminates with $root$. The deposits of all ES nodes who submitted a digest with different root from the winning one are forfeited. For a round $i | i \leq I$, let $seed_k | k \in \{1, 2, \ldots K\}$ be the $K$ different $seed$ values submitted in digests containing $root$ and let $e_k$ be their count. YODA then rewards only the ES nodes corresponding to the $seed_i$ for which $\frac{e_i}{\sum_{j=1}^{K} e_j} > th_1$ where $th_1 > 0.50$. YODA confiscates the deposit of all ES nodes for which $\frac{e_i}{\sum_{j=1}^{K} e_j} < th_2$ and YODA neither rewards nor punishes the rest. These forfeited deposits are either burned or transfered to the MC.

The intuition behind using thresholds is as follows. Although MIRACLE identifies the correct root with probability $1-\beta$, it cannot say which of many digests (with different $seed$ values), both containing the correct root in a particular round, is correct. Rewarding both would encourage free-loading. A naive solution would be to reward the set of nodes corresponding to $\max_k\{e_k\}$ and punish the rest. There are, however, rare instances in which Byzantine nodes can exceed quasi-honest nodes in a round. If Byzantine nodes publish the correct $root$ but with an incorrect seed, the naive method would severely punish the honest nodes. The set of quasi-honest nodes will however not be a very small fraction of an ES. Hence threshold $th_2$ is chosen small enough to ensure that quasi-honest nodes which behave honestly will not be punished $w.h.p$, while at the same time punishing lone quasi-nodes which try to guess the correct seed. Quasi-nodes have to collude in large numbers to cross the $th_1$ threshold, an attack which is non-trivial and analyzed in §VII-D

Lastly, blockchain miners perform a cleanup, where they deallocate space used for execution of $\Psi(\sigma, \tau)$. This includes the space for storing commitments, sortition results etc. Following this miners check whether the transaction queue $Q_\sigma$ is empty or not. If it is non-empty, SP nodes to initiates the protocol for off-chain execution of the transaction at the head of the queue and the cycle continues.

## VII. Security Analysis

In this section we analyze the security properties of YODA. We first analyze MIRACLE and prove many results, the most important being that it is optimal in the expected number of rounds under certain constraints. Incidentally, given MIRACLE, Byzantine nodes maximize the probability of choosing an incorrect solution by all submitting the same incorrect solution. Ironically, MIRACLE is optimal given this particular strategy of Byzantine nodes.

We then analyze RICE, proving bounds on the number of update indices, and the amount of computation that can be skipped at the end. We also prove that w.h.p. every round will have update indices which have not been encountered in previous rounds. This makes free-loading difficult.

We then present a Game-theoretic analysis of our incentive schemes proving them to have Nash Equilibria [19]. We finally stitch together all our results to show how they meet the goals mentioned in §I. Lastly we discuss why guarantees in YODA

[8] According to the blockchain safety and liveness assumptions of our threat model, CFA will never arise and we can safely consider $w_{buf} = 0$. However for blockchains reminiscent to Ethereum, CFA is a practical concern, thus we have added a non-zero $w_{buf}$ in our implementation §VIII.

[9] For a Merkle tree implementation of $\sigma$ it will suffice to send only the Merkle paths of all modified variables in $\sigma$.

are likely to work in an even more realistic setting with a stronger adversary and where quasi-honest nodes are allowed more protocol deviations.

### A. MIRACLE *Analysis*

In this section we present the security analysis and guarantees provided by MIRACLE. Let $M$ be the total size of SP, i.e. $M = |SP|$ containing $f$ fraction of Byzantine nodes. Let the probability of any node in SP getting chosen for an ES be $q$. Let $N^b, N^h$ denote the total number of Byzantine and quasi-honest (here assumed to be honest) nodes in an ES. Let $N_i^b, N_i^h$ denote $i.i.d$ Bernoulli random variable indicating if the $i^{\text{th}}$ Byzantine or honest node is selected for the ES or not. Then

$$N^b = \sum_{i=1}^{fM} N_i^b, \text{ and } N^h = \sum_{i=1}^{(1-f)M} N_i^h$$

We approximate $N^h, N^b$ by a Gaussian distribution, since they are a sum of large number of $i.i.d$ random variables. Let $\mu_h, \mu_b$ denote the mean of $N^h, N^b$ respectively and $\nu_b^2, \nu_h^2$ denote their variances. These are $\mu_h = E[N^h] = q(1-f)M$, and $\nu_h^2 = Var[N^h] = q(1-f)M(1-q)$. Similarly, $\mu_b = E[N^b] = qfM$, and $\nu_b^2 = Var[N^b] = qfM(1-q)$.

**Theorem VII.1.** *If $f = f_{max}$ and the adversary submits only a single incorrect digest, then* MIRACLE *reduces to an optimal Sequential Probability Ratio Test (SPRT) [28]. The threshold (see (2)) provides an optimal expected number of rounds for a given $\beta$. The expected number of rounds is given by*

$$\mathbb{E}[\#\ of\ rounds] \approx \frac{(1-\beta)\ln\frac{1-\beta}{\beta} + \beta\ln\frac{\beta}{1-\beta}}{\frac{(\mu_h-\mu_b)^2+\nu_h^2-\nu_b^2}{2\nu_b^2} + \ln\frac{\nu_b}{\nu_h}} \tag{6}$$

*Proof:* Consider the case where all Byzantine nodes consistently provide the same solution. Let the solutions be $d_1$ and $d_2$. The problem of determining the correct solution boils down to choosing between two hypotheses over multiple rounds: $\mathcal{H}_k$ : $d_k$ is the correct solution; $k = 1, 2$. Let $c_{k,j}$ denote the number of solutions equal to $d_k$ in round $j$. Then the optimal solution [28] is given by an SPRT in which the log-likelihood ratio after $i$ rounds is

$$L_i = \sum_{j=1}^{i} -\frac{(c_{1,j}-\mu_h)^2}{2\nu_h^2} - \frac{(c_{2,j}-\mu_b)^2}{2\nu_b^2} + \frac{(c_{1,j}-\mu_b)^2}{2\nu_b^2} + \frac{(c_{2,j}-\mu_h)^2}{2\nu_h^2}$$

$$= \frac{1}{2q(1-q)M}\left[\frac{(1-f_{max})-f_{max}}{(1-f_{max})f_{max}}\right]\sum_{j=1}^{i}(c_{1,j}^2 - c_{2,j}^2)$$

If $L_i > \ln((1-\beta)/\beta)$, then the SPRT chooses $\mathcal{H}_i$. This is equivalent to MIRACLE. ■

**Remark 1:** MIRACLE is optimal in case $f = f_{max}$. In case $f < f_{max}$, the expected number of rounds will be less than that specified in ( 6), while still ensuring that the probability of incorrectly deciding $\sigma^*$ is less than $\beta$. Specifically, let $R(f)$ be the expected number of rounds MIRACLE takes to terminate as a function of $f$. In Figure 1 , for $\beta = 10^{-20}$, we choose $q$ such that, $R(f_{max}) = 5$. We then empirically evaluate the number of rounds MIRACLE takes to terminate when $f \leq f_{max}$ over 10000 runs of MIRACLE. Key points to observe in the Figure 1 is that MIRACLE terminates early for $f \leq f_{max}$. Specifically, with $f_{max} = 0.45$, on average MIRACLE terminates in one round up to $f \approx 0.30$.

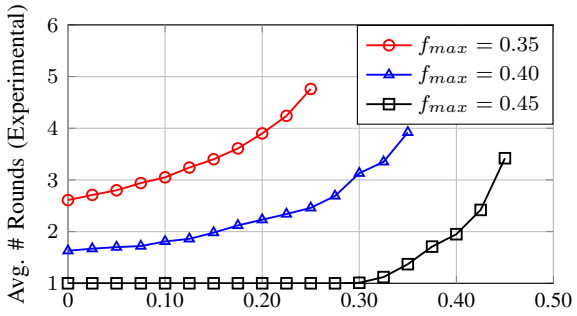


Figure 1: Number of rounds MIRACLE takes to terminate with $f \leq f_{max}$ when designed for worst case ($f = f_{max}$) expected number of rounds for termination to be 5 i.e $R(f_{max}) = 5$. For the case $f_{max} = 0.45$, for all $f \leq 0.30$, MIRACLE terminates in one round.

**Remark 2:** The probability of choosing a node to belong to the ES, $q$, can be set to any value which fixes the expected size of ES in any round. We recommend that $q$ be chosen such that if all nodes in the first ES are honest then the likelihood crosses the threshold $T$ in that round itself. This ensures that more than one round can occur only if there are some Byzantine nodes in SP. Solid line from graph in Figure 2 demonstrates minimum required $|ES|$ with $|SP|$=1600 and $f = 0$ for one round termination of MIRACLE.

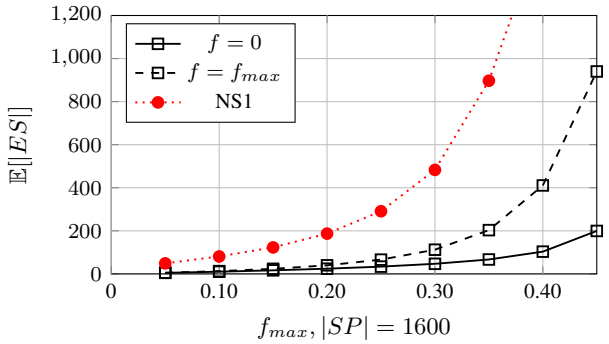


Figure 2: Dashed line shows $\mathbb{E}[|ES|]$ required for $R(f_{max}) = 1$. Solid line shows $\mathbb{E}[|ES|]$ when $R(0) = 1$. For the case $f_{max} = 0.35$, $|ES| \approx 200$ guarantees $R(f_{max}) = 1$. In addition with $\mathbb{E}[|ES|] \approx 60$, for the same $f_{max}$, MIRACLE terminates in 1 round when $f = 0$ and $\beta = 10^{-20}$.

Although we have proved that MIRACLE is optimal if an adversary chooses a single solution, the question arises as to whether the adversary has a better strategy for MIRACLE in which it chooses more than one solution. The next theorem states that this is not the case. Indeed, the best strategy for the adversary, given that YODA uses MIRACLE is to choose only a single incorrect solution.

**Theorem VII.2.** *With* MIRACLE *as the consensus algorithm, the best strategy for an adversary controlling all byzantine nodes is to only submit a single incorrect solution. Any other strategy reduces the probability of choosing incorrect solution by the system.*

*Proof:* Consider a strategy in which adversary submits

solution $d_2, d_1, ..., d_m$ for $m \geq 2$ and the solution submitted by honest nodes is $d_1$. Let us call this strategy ST1. Consider another strategy ST2, in which the adversary only submits a single solution denoted by $s'_2$ and the honest nodes submits $s'_1$. For an round $i$ in ST2, let the corresponding likelihood be $L'_{2,i}$ and the number of solutions submitted is $c'_{2,i}$. We assume that total number of submissions for an round $j \leq i$ are the same in both ST1 and ST2. Hence, it is trivial to see that $c_{k,j} \leq c'_{2,j} \forall\ k \geq 2$. Also,

$$L_{k,i} = \sum_{j=1}^{i} (2c_{k,j} - C_j)C_j \leq \sum_{j=1}^{i} (2c'_{k,j} - C_j)C_j$$

$$= \sum_{j=1}^{i} (2c'_{k,j} - C'_j)C'_j$$

$$= L'_{2,i}, \forall\ k \geq 2, \forall i$$

Thus

$$\max_{k \geq 2} L_{k,i} \leq L'_{2,i} \tag{7}$$

Hence the result follows. ∎

For each solution submitted MIRACLE has one likelihood which is compared with $\mathbb{T}$. The question arises as to whether or not more than one likelihood can simultaneously exceed the threshold, thus leading to multiple solutions for the transaction. The following theorem proves that this cannot happen.

**Theorem VII.3.** MIRACLE *can terminate with only a single solution being adjudged correct.*

*Proof:* Assume there are multiple solutions $d_1, d_2, .., d_t$ with $t \geq 2$ for which $\sum_j (2c_{k,j} - C_j)C_j > \mathbb{T} > 0$. Then summing them

$$\sum_{k=1}^{t} \sum_{j} (2c_{k,j} - C_j)C_j > 0$$

$$\Rightarrow \sum_{k} \sum_{j} 2c_{k,j}C_j > t \sum_{j} C_j^2. \tag{8}$$

Consider the left hand side of the previous equation.

$$2 \sum_{j} C_j \sum_{k} c_{k,j} \leq 2 \sum_{j} C_j(C_j) \leq t \sum_{j} C_j^2 \tag{9}$$

Equation 8 and 9 contradict each other. Hence our assumption was wrong. ∎

### B. RICE Analysis

In this section we prove that RICE adds low overhead and is secure.

**Lemma VII.1.** *Given RICE terminates in an subarray of size $2^k$, let $\phi$ be the number of times $\Psi(\sigma, \tau)$ is interrupted to update the seed in RICE, then*

$$\frac{(k-1)k}{2} < \phi \leq \frac{k(k+1)}{2} \tag{10}$$

*Proof Sketch:* Due to the slow $k$ increase strategy, the total number of times storage root is updated i.e $\phi$ is

$$\sum_{j=1}^{k-1} j < \phi \leq \sum_{j=1}^{k} j$$

which proves the lemma.

Given the bounds on number of times $seed$ is updated in RICE, we now proceed to find a relationship between the $T$ and $\phi$. Thus we first find relationship between $T$ and $k$ and then proceed further.

**Lemma VII.2.** *The relationship between $k$ and $T$ is*

$$2^k(k-2) + 2 < T \leq 2^{k+1}(k-1) + 2 \tag{11}$$

*Proof Sketch:* From the slow $k$ increase strategy we have

$$\sum_{j=1}^{k-1} j2^j < T \leq \sum_{j=1}^{k} j2^j \tag{12}$$

Simplifying further we have the result.

**Theorem VII.4. (RICE Efficiency)** *The number of times seed is updated in a single RICE run i.e $\phi$ is $\Theta((\log_2 T)^2)$.*

*Proof Sketch:* Taking log on both sides of the result of Lemma VII.2 we have $k$ is $\Theta(\log_2 T)$. This combined with equation 10 from Lemma VII.1 proves the theorem.

We now identify the possible point where last $seed$ update happens in RICE for $\Psi(\sigma, \tau)$

**Lemma VII.3.** *With $T$ as the length of array representing $\Psi(\sigma, \tau)$, the last seed update happens at $O(\frac{1}{log_2 T})$ fraction prior to $T$.*

*Proof:* Let $t_l$ be the index of last seed update and let $t_l$ lies inside a segment of length $2^k$ then $T \leq t_l + 2^k + 2^{k+1}$. Hence we want $\frac{T - t_l}{T}$ is $O\left(\frac{1}{log_2 T}\right)$

$$T > \sum_{i=1}^{k} i2^i = k2^{k+2} + 2$$

$$\frac{T - t_l}{T} \leq \frac{2^k + 2^{k+1}}{T} < \frac{3}{4k} \leq \frac{3}{4log_2 T}$$

∎

As discussed earlier, it is important for different rounds to use a different set of indices for updating the seed to prevent free-loading attacks. We thus prove how RICE prevents free-loading attacks in YODA.

**Definition VII.1. (Unmatched index)** Let RICE$_i$ and RICE$_j$, with $i \neq j$ be two RICE runs in distinct rounds in YODA. Then $M_i = \{t_i^m |\ m \in [1 : \phi]\}$ and $M_j = \{t_j^m |\ k \in [1 : \phi]\}$ denote the set of indices where $seed$ is updated in rounds $i$ and $j$. An index $t_i^m \in M_i$ is said to be an *Unmatched index* with respect to RICE$_j$ iff $t_i^m \notin M_j$.

**Definition VII.2. Strong Unmatched Index.** An index in RICE$_i$ is called Strong Unmatched Index if it is an unmatched index $\forall$ RICE$_j$ where $j < i$.

We now evaluate the distribution of number of strong unmatched index in RICE$_i$. The presence of even a single strong unmatched index implies that even if an adversary assists a quasi-honest node in a free-loading attack, by revealing root values corresponding to indices in earlier rounds, these prove insufficient to compute the digest of RICE$_i$.

**Theorem VII.5.** *Let $X_k$ denote the number of strong unmatched indices in RICE$_i$ where $\Psi'$ terminates in an segment of size $2^k$. Then $X_k$ is strictly smaller than a Poisson Binomial Distribution [30] with mean $\mu = \frac{(k-1)k}{2} - 2(i-1)$ and variance $\nu^2 \approx \sum_{n=1}^{k-1} n(1-\frac{i-1}{2^n})\frac{i-1}{2^n}$.*

*Proof:* The occurrence of a strong unmatched index in a segment of size $2^m$ in RICE$_i$ is a Bernoulli random variable with mean lower bounded by $1 - \frac{(i-1)}{2^m}$. This is a tight bound and the event corresponding to the lower bound occurs when all previous rounds have strong unmatched indices in this segment. There are $m$ such segments of length $2^m$. In the case where *all* RICE$_j$ have strong unmatched indices in *all* segments, $X$ is random variable with *Poisson binomial distribution* [30] with mean and variance given in the statement of the theorem. ∎

Given the above we proceed to find a lower bound on the probability that the number of strong unmatched index $X_k$ is greater than some $x_k$. We achieve this by finding a lower bound on $P(X \geq x)$ in the $i^{\text{th}}$ round with the tail of a Binomial Random variable. As CIC size $T \to \infty$ we prove that for a series $x_k \to \infty$, the tail of the binomial and hence $P(X_k \geq x_k)$ goes to 1. This means that the number of strong indices increases without bound w.h.p. as $T$ increases. This in turn reduces the chances of success of a free-loading attack as it becomes virtually impossible for a node to guess the $root$ values at an increasingly large set of strongly unmatched indices.

**Theorem VII.6.** *Let $\Psi(\sigma,\tau)$ in the $i^{th}$ round of* MIRACLE *ends in a segment of size $2^k$. Given $X_k$ defined as in Theorem VII.5, we can lower bound the tail probability of $X_k$ i.e $P(X_k \geq x_k)$ for any $x_k \leq k(k-1)/2$ with the tail probability of $\mathcal{B}\left(\frac{(k-1)k}{2} - \frac{(b_2-1)b_2}{2}, x_k, 1-\frac{i-1}{2^{b_2}}\right)$ for any $b_2 \leq k$ where $\mathcal{B}(n,.,p)$ is a binomial distribution with $n$ trials with $p$ as success probability of each trial.*

*Proof Sketch:* Call the occurrence of a strong unmatched index in a segment of size $2^m$ in RICE$_i$ as a trial in that segment. The trial is a Bernoulli random variable with mean lower bounded by $1 - \frac{(i-1)}{2^m}$. If $m > b_2$ then the mean has lower bound $1 - (i-1)/2^{b_2}$ and if $m \leq b_2$ the mean is lower bounded by 0. Hence $X_k$ which is the sum of all trials has tail distribution strictly higher than the tail of the sum of $(b_2-1)b_2/2$ i.i.d. Bernoulli random variables with mean $1-(i-1)/2^{b_2}$. There are $\frac{(k-1)k}{2} - \frac{(b_2-1)b_2}{2}$ number of segments with $m > b_2$. The result follows.

**Lemma VII.4.** *As $k \to \infty$, $P(X > \sqrt{k}) \to 1$.*

*Proof Sketch:* Choose $x_k = \sqrt{k}$. The result follows from the above Theorem and the use of the well-known bound on the tail distribution of $\mathcal{B}(n,l,p)$ given by

$$P\left(\mathcal{B}(n,.,p) > l\right) \geq 1 - e^{-2\frac{(np-l)^2}{n}}$$

**Remark 1:** Recall that MIRACLE allows the system designer to choose an appropriate $|ES|$ size to achieve an expected number of rounds. In this way, the number of rounds can be limited to less than a constant $i$ $w.h.p.$

**Remark 2:** Since $\sqrt{k}$ grows unboundedly with $k$, it follows that for large sized ITs, and some finite round $i$, the number of strong indices grows unboundedly w.h.p. Since the $root$ values at these strong indices are not known w.h.p. (except for trivial CICs where storage does not change over indices) the final $seed$ also cannot be known w.h.p.

The next result shows that the probability of occurrence of any particular seed value is vanishingly small assuming the roots at different strong indices are mutually independent.

**Theorem VII.7.** *Let the probability mass distribution of the $root$ at all strongly unmatched indices in round $i$ be upper bounded by $1-\lambda$ for some $\lambda > 0$. Let the last segment of the CIC be of size $2^k$. Then as $k \to \infty$ the probability mass function of the $seed$ at the end of RICE$_i$ is negligible assuming an ideal hash function, and that the root at different unmatched indices are mutually independent.*

*Proof Sketch:* Let $j \in [1 : X]$ denote the $X$ strongly unmatched indices, and $r_j$ and $s_j$ the corresponding root and seed. Since the hash function is ideal, it maps unique inputs to unique outputs.

Thus $P(\mathsf{hash}(s_j||r_j)) = P(s_j, r_j) = P(s_j)P(r_j)$. The last equality is due to the independence assumption. We assume that $s_1$, the seed at the first strong unmatched index, is known to the node and hence $P(s_1) = 1$. Denoting $seed_{X+1}$ as the final seed, we have $P(seed_{X+1}) = \prod_j P(r_j) \leq \prod_j \max P(root_j) \leq (1-\lambda)^X$. Since $X$ is larger than $\sqrt{k}$ w.h.p. as $k \to \infty$ we have $P(s_{X+1}) \to 0$.

**Remark:** The roots of indices "far apart" being independent is not unrealistic, except for trivial CICs. Strong unmatched indices are in different segments and hence except for neighboring indices, they are separated by whole segments, and hence we conjecture that the independence assumption is a good approximation in practice. We also conjecture that the same result holds for weaker assumptions than stated in the theorem and leave the proof for future work.

### C. Free-loading attack

We now analyze a free-loading attack where a quasi-honest node skips computation of the CIC by using information available on the blockchain and/or state information of $\Psi(\sigma,\tau)$ from previous rounds received from an adversary §IV. We consider the best case scenario for the free-loading node where it knows the correct $root$ of the $digest$ w.h.p. but has to guess the $seed$. We analyze the case where Byzantine nodes have maximum fraction $f_{max}$ in SP and all submit the same incorrect $root$ with the same $seed$ in order to maximize the probability of MIRACLE selecting their solution, and where quasi-honest nodes do not collude.

Denote the profile where all quasi-honest nodes execute the CIC as $\overrightarrow{a}$ and the profile where only a single quasi-honest node $n_i$ free-loads as $\overrightarrow{a}_{-i}$. With $\overrightarrow{a}$ the analysis of MIRACLE with honest and Byzantine nodes holds. Hence quasi-honest nodes win reward $\mathcal{R}$ with probability $1-\beta$, and lose their deposits $\mathcal{D}$ with probability $\beta$. The cost of computing the CIC is $c_1$. Hence the utility for $n_i$ with this profile is

$$\mathcal{U}_i(\overrightarrow{a}) = (1-\beta)\mathcal{R} - \beta\mathcal{D} - c_1 \tag{13}$$

Let $\gamma$ be the probability of $n_i$ guessing the correct seed while free-loading. If it guesses the correct seed then its probabilities of winning a reward and losing its deposit are $(1-\beta)$ and $\beta$ as

above. If it guesses the wrong seed then it loses its deposit. We denote by $c_2$ the cost of bandwidth consumed for downloading intermediate $storage$ of previous rounds from an adversary and analyzing them to predict the $seed$. Then the utility

$$\mathcal{U}_i(\overrightarrow{a}_{-i}) = \gamma((1-\beta)\mathcal{R} - \beta\mathcal{D}) - (1-\gamma)\mathcal{D} - c_2 \quad (14)$$

From (13) and (14) we obtain $\mathcal{U}_i(\overrightarrow{a}) - \mathcal{U}_i(\overrightarrow{a}_{-i}) > 0$ iff

$$\mathcal{R} + \mathcal{D} > \frac{c_1 - c_2}{(1-\beta)(1-\gamma)} \approx c_1 - c_2 \quad (15)$$

where the last approximation is due to the fact that $\gamma$ is vanishingly small in practice and $\beta$ is a design parameter chosen to be small. Since $\mathcal{R} > c_1$, that is the reward must be more than the cost of computation, we see that (15) is true. Hence profile $\overrightarrow{a}$ is a Nash equilibrium [19].

### D. Collusion Attack

We now consider the case where a group $\mathcal{C}$ of ES nodes collude to submit a common seed. We assume they know the correct root $w.h.p$, that Byzantine nodes all submit the same incorrect $root$ with the same $seed$ in order to maximize the probability of MIRACLE selecting their solution, and that all other quasi-honest nodes execute the CIC correctly. Since $|ES|$ is random, nodes in $\mathcal{C}$ cannot be entirely sure if $|\mathcal{C}|$ is larger than $th_1|ES|$ or less than $th_2|ES|$. Suppose $|\mathcal{C}| > th_1|ES|$ with probability $\gamma_1$ and $|\mathcal{C}| < th_2|ES|$ with probability $\gamma_2$. The computation cost of colluding requires solution of ZK-proofs since nodes need to prove they belong to ES without revealing their Denote the associated costs by $c_3$ and this profile by $\overrightarrow{a}_{-\mathcal{C}}$.

In case the Byzantine nodes win MIRACLE, $\mathcal{C}$ lose their deposits. In case the correct root is selected by MIRACLE, $\mathcal{C}$ win a reward with probability $\gamma_1$, and lose their deposits with probability $\gamma_2$. Hence utility for node $n_i \in \mathcal{C}$

$$\mathcal{U}_i(\overrightarrow{a}_{-\mathcal{C}}) = \gamma_1((1-\beta)\mathcal{R} - \beta\mathcal{D}) - \gamma_2\mathcal{D} - c_3 \quad (16)$$

In case $\gamma_1 = 1$, $U_i(\overrightarrow{a})$ is a $\epsilon$-Nash equilibrium [22] with $\epsilon = c_3 - c_1$. In this special case, if the $c_3$ is larger than the CIC computation cost itself, the nodes are better off being honest. Note that higher $|\mathcal{C}|$ increases $\gamma_1$, but also increases $c_3$ because of more ZK proofs, and related communication costs.

### E. Meeting the Requirements.

We here summarize how various mechanisms in YODA meet the goals listed in §I. Most requirements are met due to MIRACLE. For all $f_{max} < 0.50$, MIRACLEterminates and thus off-chain execution of a CIC also *Terminates*. MIRACLE allows a system parameter $\beta$ which is the probability of accepting a incorrect solution, thus achieving *Validity* with tunable high probability. *Agreement* on off-chain CIC execution trivially follows from the *safety* guarantees of the blockchain. Recall in YODA, to initiate the reward mechanism process ES nodes need to submit the correct storage root in the blockchain and the miners verify it before its inclusion ensuring *Availability* of the post-execution state $\sigma^*$.

Since YODA never requires the blockchain to verify the CIC execution on-chain, YODA is *Oblivious*. YODA requires ES to be much smaller than naive approaches discussed in §III thus making YODA *Efficient*. MIRACLE is *Adaptive* to the fraction of Byzantine nodes in SP, and terminates earlier the smaller this is §III. For appropriate choice of $th_1, th_2$ in §VI YODA ensures *weak-Fairness*.

### F. Stronger Adversary Scenarios with Rational Nodes

We now discuss how YODA may perform if we replace quasi-honest nodes by *rational* nodes and also allow stronger adversaries. Unlike quasi-honest nodes, rational nodes are not conservative (ref. §II-A) towards Byzantine nodes. The stronger adversary is allowed to try to reveal information about the current round to a rational node, and not just information about past rounds.

Consider the case where an adversary $\mathcal{A}$ is actively providing information about the $root$ at intermediate RICE indices for the current round. Recall from RICE §VII-B that for each round we will have a large number of strong unmatched indices with large probability, and thus for all such unmatched indices, a rational node $\mathcal{N}$, has to obtain the $root$ from $\mathcal{A}$. How will $\mathcal{A}$ convince $\mathcal{N}$ about the correctness of the states at these intermediate points? One mechanism is that $\mathcal{A}$ commits its digest in the current round and gives a zk-Proof to $\mathcal{N}$ about it. A challenge for $\mathcal{N}$ is that it does not know whether $\mathcal{A}$ is Byzantine or rational. Even if we consider that rational nodes commit correct digest, when $\mathcal{A}$ is Byzantine then its commitment could be false.

Consider the case where two or more rational nodes in an ES want to collaborate to generate the correct $digest$. Consider one rational node $\mathcal{N}_R$ interacting with another node $\mathcal{N}_U$ which might potentially be Byzantine. First, both nodes must prove that they belong to ES. Ruling out a non-Interactive zk-Proof, $\mathcal{N}_R$ and $\mathcal{N}_U$ have to produce an interactive zk-Proof, and this may be vulnerable to DoS attacks. Once the proofs are established, one strategy could be to split the execution among them. This situation again boils down to trusting the result claimed by $\mathcal{N}_U$ about its execution.

Finally we discuss pragmatic concerns related to a *collusion attack*. In our utility analysis we considered the case where all nodes in the colluding set $\mathcal{C}$ are all quasi-honest but in practice it will possibly contain Byzantine nodes as well. This will possibly lower the probability $\gamma_1$ of successful collusion. Since the success of the collusion attack depends on $|ES|$, imperfect knowledge of $|ES|$ during the commitment phase of RICE lowers $\gamma_1$ even more. Interestingly even with knowledge of $|ES|$ prior to collusion and $|\mathcal{C}| \geq th_1|ES|$, the success truly depends on the behavior of Byzantine nodes in $\mathcal{C}$.

## VIII. Implementation and Evaluation

To experimentally evaluate the scalability of YODA, we have implemented a prototype which includes all parts of YODA except SP selection, on top of the popular Ethereum geth client version `1.8.7` and evaluate them in a private network. The SP selection procedure is independent of CIC transaction gas limits and hence does not impact scalability. Our implementation consists of ∼500 lines of code (LOC) in Solidity (for the Master Contract and sample CICs) and ∼2000 LOC in python on top of the interface for the modified client whose task is detailed below.

**Experimental Environment.** We use 8 physical machines each with a $2.80 \times 8$ GHz intel Core-i7 processor with 8GB RAM and running Ubuntu 17.10 to simulate 16 off-chain clients. Each client emulates 100 YODA nodes thus making $M = |SP| = 1600$. Since off-chain CIC execution requires considerable computations resources, we restrict the number of clients per machine to two. All these 16 clients are connected to an Ethereum network created using geth which we consider

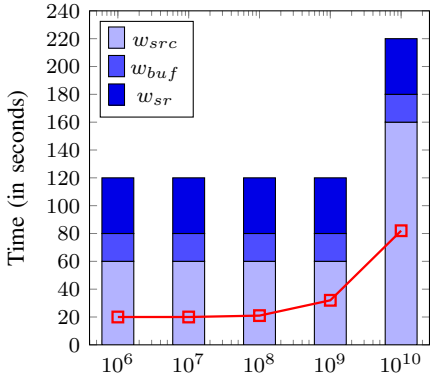


Figure 3: Measured CIC execution time with varying gas usage.

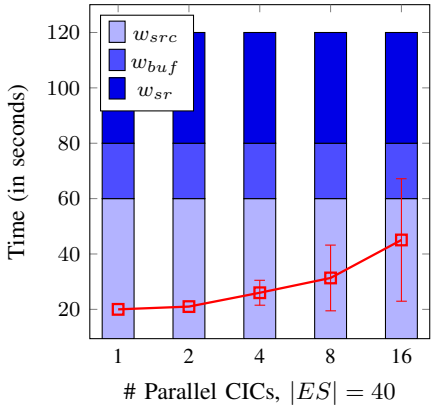


Figure 4: Average digest commit time with increasing number of parallel ITs.

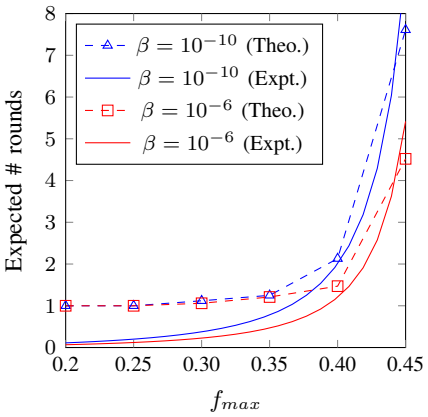


Figure 5: Change in Expected number of rounds vs. $f_{max}$ for $|SP| = 1600, q = 0.125$.

to be the blockchain network. The blockchain runs on Proof of Authority in geth which is a developer mode option in the Ethereum private network to keep the block mining time fixed. A separate physical machine with similar specifications handles transactions and mines blocks. To achieve a required $|ES|$ we allow a single client to execute an IT only once and submit commitments for multiple SP nodes based on how many of its nodes are selected by Sortition. For each execution we regulate $q$ to achieve an appropriately sized ES.

**Executing CICs in the EVM.** To execute ITs off-chain, each client in our system runs a modified EVM supported with an implementation of RICE. This EVM provides the interface for off-chain execution of a CIC. The clients deploy the contract in their local EVM whenever a CIC is created on the blockchain. When a IT is broadcast on-chain, clients download the transaction and execute them locally in the EVM based on the sortition result. After execution each client broadcasts a transaction containing the $digest$ for each of its ES nodes. An identical mechanism is used to reveal the commitment.

### A. *Scalability of CICs.*

We start with a CIC containing a parameterizable function called `Compute()` with parameter $\eta$ which sets the amount of gas to be used. Internally `Compute()` runs iterations modifying a CIC state variable after performing arithmetic operations. We start `Compute()` with gas usage $5.3 \times 10^6$ and increase this by a factor of 10 rising up to $5.3 \times 10^{10}$. Note that each Ethereum block can only accommodate up to 8M gas in 15 seconds (avg. block generation time) at the time of writing and hence we conclude that its maximum gas usage per second is $5.3 \times 10^5$.

**Varying Gas Usage.** In Figure 3 we plot the average of measured time from the start of the round up to the commitment (S1-S2) as a red line and the range of values as error bars. Observe that the average time remains constant at about 20 secs till gas usage of $5.3 \times 10^8$ after which it increases to about 30 secs for $5.3 \times 10^9$ and to about 100 secs after that. The total submission time includes off-chain computation time and in addition Ethereum on-chain delays, such as the time for a transaction to be included in a block on-chain. Clearly for gas $5.3 \times 10^9$ or less the on-chain delays dominate after which off-chain delays dominate.

We also plot a bar graph depicting the windows for committing the storage root $w_{src}$, the buffer period $w_{buf}$, and the window to reveal the storage root value $w_{sr}$. Among the three only $w_{src}$ depends on $gas$ usage since the computation of the IT happens during this time. The total time for one round is $w_{src} + w_{buf} + w_{sr}$.

From the experiment with CIC gas usage equal to $5.3 \times 10^{10}$, we see that YODA consumes 240 Million gas per second. This amount is $450\times$ more than the existing amount of gas Ethereum can use per second. Note that this speedup is when only a single IT is running. With parallel execution of ITs this scales up further as we demonstrate in our next experiment.

**Parallel CICs.** We further test YODA by running up to 16 parallel ITs. Figure 4 shows the time taken for executing different number of concurrent ITs. All ITs are invoked at once in a single block on-chain and the $gas$ usage of all are kept identical. The red line in Figure 4 records the average of the storage root commitment times and error bars are used to indicate the range of these. Observe that the minimum commit time remains almost constant, indicating that the time for off-chain execution is the same. However the maximum value increases. This is because more blocks are needed to include the increased number of commitment transactions. As a result the average commit time increases gradually. As future work we will devise mechanisms to automatically provision $w_{src}$ taking this phenomenon into account.

**Evaluation of MIRACLE.** We next evaluate the performance of MIRACLE in the presence of a Byzantine adversary. In our experiments the adversary uses the best strategy, that is it submits a single incorrect solution for all nodes it controls.

For system design the expected number of rounds is a crucial parameter. We determined this quantity experimentally and compared it to its theoretical approximation. In Figure 5 we plot $\mathbb{E}$[# Rounds] versus the fraction of Byzantine nodes $f$ for different values of the probability of accepting an incorrect storage root $\beta$ in the range $10^{-3}$ to $10^{-10}$. For this experiment we fixed parameter values $q = 0.125, M = 1600$ giving us an ES of expected size 200. Notice that $\mathbb{E}$[# Rounds] increases with an increase in $f$ and largely agrees with the theoretical approximation. The theoretical approximation has an artifact in that it can be less than 1 which is impossible because the number of rounds is at least 1 always.

**Evaluation of RICE.** We next evaluate the overheads associated with RICE when implemented on the EVM geth client. For this experiment, we measure CIC execution time on the

unmodified EVM and then perform the same experiment in a EVM modified with RICE implementation. For each gas usage, we aggregate the results over 200 repetitions. Figure 6 shows the time difference of CIC execution with RICE and without RICE. As expected, the absolute difference increases as gas increases due to the presence of more update indices. More interesting to observe is Figure 7 where we plot gas usage vs. relative execution time i.e ratio of absolute time difference and CIC execution time without RICE. First observe that the relative overhead due to RICE is extremely low. As gas increases, the relative time decreases because the RICE indices become sparse in later segments and hence add less overhead. During the early part of the graph wee see a small aberration. This is because time with and without RICE are small and hence minor variations in the absolute time difference get magnified relative to time without RICE.

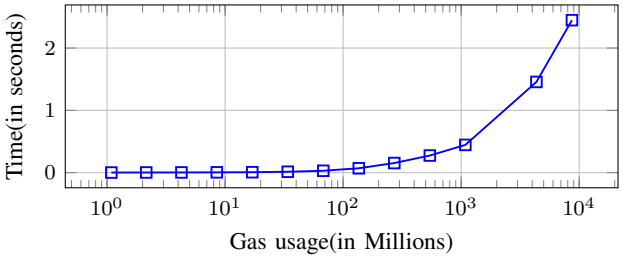


Figure 6: Absolute overhead of RICE in CIC execution plotted against increasing gas usage.

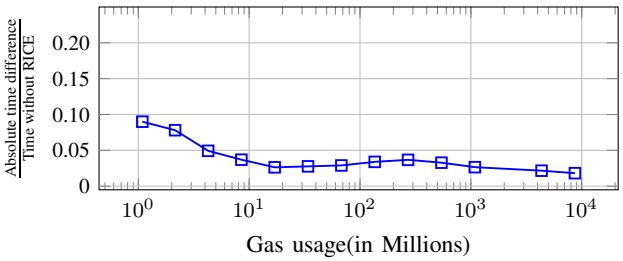


Figure 7: Relative overhead of RICE in CIC execution plotted against increasing gas usage.

## IX. Related Work

The threat model of combining Byzantine and selfish nodes in distributed systems dates back to Aiyer et al. [1] Prior to their work threat models in distributed systems either considered the presence of perfectly honest nodes and Byzantine nodes, or only selfish nodes. The *Byzantine Altruistic Rational* (BAR) considered a more realistic scenario combining the two models in a permissioned cooperative service with PKI [1]. Our threat model is of the BAR variety.

Most analysis of blockchain consensus protocols in the permissionless case limit themselves to a threat model of Byzantine with honest nodes which are not BAR models. These include works on Fruitchains [20], Algorand [10], and the sleepy model of consensus [21]. In the sleepy model, honest nodes may go offline and not participate in the protocol. Ouroboros [13] introduces $\epsilon$-Nash Equilibrium for a proof-of-stake protocol. Selfish mining [23] shows in case a non-zero fraction miners in PoW blokchains behave selfishly, honest behaviour is no longer an equilibrium, because individuals unilaterally benefit by joining hands with the attacker. All these works solve a very different problem from YODA, namely that of block consensus. Blocks can potentially take many values and are easy to verify. In contrast, CIC computation can have only one correct value and are computationally intensive to verify.

Truebit is a proposal to enable CICs on permissionless blockchains, in the presence of selfish nodes [26]. Truebit requires a single *Solver* to execute and upload the results of the transaction, and any number of volunteer verifiers to verify the Solver's solution. There is no bound on the number of verifiers unlike in YODA. Moreover Truebit does not claim to provide guarantees for probability of correct CIC computation under a threat model. In fact, recent work shows that Truebit is susceptible to a Particpation Dilemma, where if all participants are rational, an equilibrium exists with only a single verifier which can cheat at will [12]. It also makes payouts to verifiers rare events, unlike YODA which pays ES members rewards immediately.

Arbitrum is a system for scalable off-chain execution of private smart contracts developed concurrently with our work [12]. In Arbitrum, each smart contract can assign a set of managers who execute its transactions off-chain. Any one manager can submit a hash of the updated state on-chain. In addition, any other manager can submit a challenge if this earlier submitted state is incorrect. Arbitrum works under a threat model with at least one honest manager and the rest of the managers being rational. It has not been proved to work in the presence of Byzantine managers, or with all managers being rational. In contrast, YODA works in the presence of both Byzantine and selfish nodes, none of which need to be honest. YODA is also not restricted to private smart contracts.

Several other papers focus on sharding for improving performance of permissionless blockchains [2], [14], [15]. None of these, however, focus on the specific problem of executing CIC transactions efficiently. They instead increase throughput in terms of number of non-IT transactions executed over time. The execution (or verification of correct execution) is implicitly assumed to take little time, and all miners verify all transactions.

## X. Discussion and Conclusion

We have presented YODA which enables permissionless blockchains to compute CICs efficiently. Experimental results show that individual ITs with gas usage 450 times the maximum allowed by Ethereum can be executed using the existing EVM. YODA uses various incentives and technical mechanisms such as RICE to force rational nodes to behave honestly. Our novel MIRACLE algorithm uses multiple rounds to determine the correct solution and shows great savings in terms of number of rounds when the actual Byzantine fraction of nodes is less than the assumed worst case.

One advantage of YODA is its modular design. Several modules can be left intact, while replacing the others. Examples of such modules are RICE, MIRACLE, SP selection, and ES selection, which can in future be replaced by alternatives.

Several issues need to be addressed before YODA can become a full fledged practical system. One open problem we have not addressed is the issue of data. Often, large CICs are likely to have large state and each IT can potentially modify many state variable. Broadcasts of every update for every IT can be costly in terms of communication. A possible alternative to this state update could be storage of data in a Distributed File

System like IPFS [5] and using Authenticated Data structures such as Versum [27] to store a succinct representations of it in the blockchain.

Another concern is regarding the number of additional transactions needed to achieve consensus on a CIC. For each round, MIRACLE requires each ES node to submit two short transactions. Also the count of such transactions depends on $f$. Observe from Figure 2 that in the best case YODA requires only $\approx 70$ transactions with $f = 0$ and $\beta = 10^{-20}$. With contemporary blockchain solutions that claims to scale up to 1000s transaction per second [10], [31] these transactions consume relatively small bandwidth.

The periods $w_{src}, w_{sr}$ chosen for execution of CICs off-chain will in practice also depend on the number of simultaneous ITs being currently executed by YODA. This is because, as the number of simultaneous ITs increase, the average CIC workload on any node increases as well, since each node may belong to multiple ES sets simultaneously. As CICs are computationally expensive, the MC must further keep a limit on ITs at any instant of time to reduce the maximum load on an ES node.

## ACKNOWLEDGMENTS

The authors would like to thank Aditya Ahuja, Cui Changze, Aashish Kolluri, Dawei Li, Sasi Kumar Murakonda, Prateek Saxena, Ovia Seshadri, Subodh Sharma, and anonymous reviewers for their feedback on the early draft of the paper.

## REFERENCES

[1] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services," in *ACM SIGOPS operating systems review*, vol. 39, no. 5. ACM, 2005, pp. 45–58.

[2] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.

[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.

[4] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture." in *USENIX Security Symposium*, 2014, pp. 781–796.

[5] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

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

[7] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 106–125.

[8] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[9] J. Eberhardt and S. Tai, "Zokrates-scalable privacy-preserving off-chain computations."

[10] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.

[11] A. Juels, A. Kosba, and E. Shi, "The ring of gyges: Investigating the future of criminal smart contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 283–295.

[12] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *Proceedings of the 27th USENIX Conference on Security Symposium*. USENIX Association, 2018, pp. 1353–1370.

[13] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[14] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[15] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 17–30.

[16] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 706–719.

[17] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999, pp. 120–130.

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

[19] J. F. Nash *et al.*, "Equilibrium points in n-person games," 1950.

[20] R. Pass and E. Shi, "Fruitchains: A fair blockchain," in *Proceedings of the ACM Symposium on Principles of Distributed Computing*. ACM, 2017, pp. 315–324.

[21] ——, "The sleepy model of consensus," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 380–409.

[22] R. Radner, "Collusive behavior in noncooperative epsilon-equilibria of oligopolies with long but finite lives," in *Noncooperative Approaches to the Theory of Perfect Competition*. Elsevier, 1982, pp. 17–35.

[23] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 515–532.

[24] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *Security and Privacy (SP), 2017 IEEE Symposium on*. Ieee, 2017, pp. 444–460.

[25] N. Szabo, "Smart contracts: building blocks for digital markets," *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996.

[26] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," 2017.

[27] J. van den Hooff, M. F. Kaashoek, and N. Zeldovich, "Versum: Verifiable computations over large public logs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1304–1316.

[28] A. Wald, *Sequential analysis*. Courier Corporation, 1973.

[29] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1386–1393.

[30] Y. H. Wang, "On the number of successes in independent trials," *Statistica Sinica*, pp. 295–312, 1993.

[31] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 931–948.

[32] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*. ACM, 2016, pp. 270–282.