MultiVAC: A High-Throughput Flexible Public Blockchain Based on Trusted Sharding Computation

core@mtv.ac
MultiVAC Foundation
June, 2018, version 0.1

Abstract: MultiVAC is a next-generation high-performance public blockchain for industrial-scale decentralized applications. Its trusted sharding technology allows for unlimited and sustainable scalability, and it provides a novel approach towards solving the blockchain scalability problem currently preventing mainsteam blockchains from reaching industrial capability. MultiVAC is the first to propose a sharding model based on Verifiable Random Functions (VRF) and applies this model to transactions, computation, and storage. We confirm transactions in our network through a classic UTXO model with miners dynamically selected through a probability model. MultiVAC allows for the high levels of safety and reliability needed by industrial applications while only requiring processing on a small number of nodes, producing significant speed improvements. On top of our fast and scalable blockchain model, MultiVAC is the first in the industry to provide a computational model for smart contracts which allows developers to flexibly decide for themselves the tradeoff between consistency, availability, and partition tolerance, parameters that are often stiffly fixed by the designs of many public blockchains. We achieve this by providing a general-purpose virtual machine MVM equipped with a specially designed blockchain instruction set (BISC) and a powerful method to validate the correctness of smart contract executions (PoIE). With this suite of breakthroughs, MultiVAC is extremely fast, totally scalable, and robustly allows for the development of extremely complicated business logic on its application layer, an ideal blockchain to serve as the foundational layer of a public diversified blockchain ecosystem.

Keywords: blockchain; shard; reliability; probability model; flexibility

1. Problems with Traditional Blockchain

Blockchains must be scalable to achieve their full economic use in society. This statement necessarily entails compromises. It is known that blockchain protocols suffer from the so-called "impossible triangle": it is impossible for a single blockchain to have at once the three desiderata of security, decentralization, and scalability. The largest public blockchains today compromise between the three features; for example, Bitcoin[1] and Ethereum[11] are secure and decentralized but are also completely unscalable. The computational power of their entire network is stuck at the level of a single miner. On the other hand, sharding models such as Zilliqa[16] and Dfinity[19] abandon security guarantees for scalability, and models like EOS[13] abandon decentralization and attempt to solve the performance bottleneck using supernodes. State channel technologies such as Plasma[28] take yet another approach, by taking the transactions off-chain when attempting to solve the scaling issue.

Massive amounts of research and development are already being invested into scaling blockchain, leading to recent throughput levels of several thousand transactions per second (e.g. EOS[13] and Seele[18] reach 1000-3000 tps in some experiments, tps = transactions per second). Despite this, the low hardware processing capacity of a single miner is still the major bottleneck. As these networks' speed do not improve as they scale in node number, there is no significant incentive pushing them to enlarge, resulting in network growth at underwhelming rates: as of 12:00 Noon (UTC time) on May 13, 2018, there were only 10,424 full nodes on the ten-year-old Bitcoin blockchain [7] and only 14,383 on Ethereum [8].

MultiVAC believes that the viability of blockchain for businesses today depend on whether or not blockchains can provide general-purpose computation, whole-network transactions, and network-wide contract processing in a scalable, expandable, and adaptable way.

We propose a trusted sharding model that solves the scalability problem, allowing for the unlimited accumulation of transaction power from nodes worldwide, i.e. scalability without limit. We construct and exploit a relationship between the network division of labor and consensus reliability to make our model effective. MultiVAC performs sharding independently for transaction processing and for smart contract execution, creating an incredibly supportive and flexible base-layer blockchain platform. DApps on MultiVAC can realize general-purpose computational business logic and can flexibly decide according to their own security needs how many nodes on which they wish to run their code.

MultiVAC solves three fundamental problems:

1. How to create shards from network nodes for transaction and smart contract processing in a trustworthy manner, allowing the network to scale.
2. How to process transactions and update records using trusted shards in the use case of transaction processing.
3. How to verify the correct and honest execution of smart contract code by network nodes in the use case of smart contract processing.

We summarize the presentation of our system in this paper: MultiVAC creates shards through a novel probability model based on Verifiable Random Functions (VRF), solving the problem of how to safely, efficiently, and randomly shard the network. We use the Byzantine consensus family to reach internal consensus within a shard, achieving the construction of trustworthy shard-based consensus. We ready our blockchain for smart-contract deployment by designing an optimized virtual machine MVM capable of general-purpose computation, which is equipped with a special blockchain instruction set BISC, and which verifies correctness of contract execution through Proof of Instruction Execution (PoIE). This creates a not only trusted but also flexible execution environment that allows for the execution of complicated general-purpose business logic.

2. Verifiable Random Functions

A consensus algorithm is a mechanism for selecting bookkeeping nodes. In this process, overall consideration should be paid to the efficiency and equitability of the selection, with

each honest node ideally given equal opportunity to participate in the bookkeeping process. This was achieved in Bitcoin and Ethereum by using Proof of Work (PoW), which guarantees that the node selection process is sufficiently random and that the network regulations can only be broken with the collusion of over 51% of the network's total compute power. PoW is an elegant solution that embodies the equitability inherent in the paradigm of decentralization, but it is also massively inefficient. Another approach to node selection is the DPoS algorithm represented by Graphene, which improves the throughput of a PoW system but abandons randomness by giving up the ability for common nodes to participate in consensus, thus sacrificing decentralized equitability for efficiency. Yet other consensus algorithms i.e. PBFT [2] ($O(n^2)$ complexity) and RAFT [21] are equitable but difficult to realize on public blockchains in large scale due to high communication costs.

MultiVAC believes that Bitcoin and Ethereum have a desirable degree of equitability by allowing all ordinary nodes to have a say in the verification process. This property is the bedrock of blockchain's future development. Ethereum has even designed a custom hash function called ETHash to keep its network equitable and to combat ASIC mining, returning bookkeeping power from specialized miners to ordinary users. However, the PoW system remains to be extremely wasteful and thus we choose a better node-selection process based on Verifiable Random Functions (VRF).

The ideal node-selection algorithm integrates both equitability (randomness) and efficiency. Decentralization is the most basic value proposition of blockchains, but the future of blockchains also depends on substantial efficiency improvements to current systems. An optimal method for resolving this contradiction is the usage of Verifiable Random Functions (VRF), a framework also used in Algorand[4], Dfinity's BLS [5], and Cardano's Ouroboros Praos algorithm [6]. Verifiable Random Functions satisfy at the same time the requirements for equitability and efficiency.

Intro to VRFs. VRFs are pseudorandom functions such that the functions' user can produce a proof allowing all parties to verify the function was calculated correctly without ever needing to disclose the random function itself.

In our case, a satisfactory VRF has the following desirable characteristics:

1. It can be used to verify that a random number generator has provided a rigorous level of randomness.
2. It is impossible to predict or control.
3. It is a non-interactive algorithm and so can be implemented with lower-cost and higher efficiency.

Definition and Properties. Formally, a VRF consists of three polynomial-time functions:

$$\text{VRF} = \{\text{Generate}, \text{Evaluate}, \text{Verify}\}.$$

These functions perform the following operations:

$$\text{VRF.Generate}\left(1^{\lambda}\right) \rightarrow \{PK, SK\}.$$

VRF.Generate generates the pair PK (public key) and SK (secret key) according to a security parameter λ.

$$\text{VRF.Evaluate}(SK, x) \rightarrow \{\delta(SK, x), \pi(SK, x)\}.$$

VRF.Evaluate produces an encrypted output value δ and a proof π according to the private key and some input x.

$$\text{VRF.Verify}(PK, x, \delta, \pi) \rightarrow (\text{true}|\text{false}).$$

VRF.Verify is able to verify whether or not the encrypted output value did indeed come from the VRF.Evaluate calculation given the proof, the public key, and the value of x. Both VRF.Generate and VRF.Verify are probabilistic functions while VRF.Evaluate is deterministic.

Now, given any three polynomial-time functions a, b, and s over integers such that

$$a: N \rightarrow N \cup \{*\}$$
$$b: N \rightarrow N$$
$$s: N \rightarrow N$$

We say $\text{VRF} = \{\text{Generate}, \text{Evaluate}, \text{Verify}\}$ is a *verifiable pseudorandom function* with input length $a(\lambda)$, output length $b(\lambda)$, and security level $s(\lambda)$ if the following three conditions are satisfied:

1. *Probabilistic Correctness*: The probability of the following two conditions is each not less than $1 - 2^{-\Omega(\lambda)}$:
   a) Domain Range Correctness:
   For any $x \in \{0,1\}^{a(\lambda)}$, we have $\delta(SK, x) \in \{0,1\}^{b(\lambda)}$.
   b) Complete Provability:
   For any $x \in \{0,1\}^{a(\lambda)}$, if $(\delta, \pi) = \text{VRF.Evaluate}(SK, x)$, then
   $\text{Prob}[\text{VRF.Verify}(PK, x, \delta, \pi) = \text{true}] > 1 - 2^{-\Omega(\lambda)}$,
   where the left side of the greater-than sign is over coin tosses of VRF.Verify.

2. *Unique Provability*:
   For any $PK, x, \delta_1, \delta_2, \pi_1, \pi_2$ such that $\delta_1 \neq \delta_2$, for all i,
   $Prob[\text{VRF.Verify}(PK, x, \delta_i, \pi_i) = true] < 2^{-\Omega(\lambda)}$.

3. *Residual Pseudorandomness*:
   For any algorithm $T = (T_E, T_J)$ with original input 1^{λ} taking total execution count less than or equal to $s(\lambda)$, and for any $\cdot \neq x$, let
   $$(x, \tilde{\pi}) \leftarrow T_E^{VRF.Evaluate(SK, \cdot)}(1^{\lambda}, PK)$$
   where PK, SK are generated through $VRF.Generate$.

Now, we define a random event X which takes on two states with equal probability 0.5 each. Depending on the state of X, we determine a value for $\tilde{\delta}$ either randomly or from $\delta(SK, x)$:

$$\text{Prob}\left[X: \tilde{\delta} = \delta(SK, x)\right] = 0.5$$
$$\text{Prob}\left[X: \tilde{\delta} \xleftarrow{Random} \{0,1\}^{b(\lambda)}\right] = 0.5$$

We require that no prediction algorithm T_J is able to accurately predict within the margin of safety the actual state of X that generated $\tilde{\delta}$:

$$Prob\left[T_J^{VRF.Evaluate(SK, *)}\left(1^{\lambda}, \tilde{\delta}, \tilde{\pi}\right) = X\right] \leq 0.5 + s(\lambda)^{-1}.$$

VRF defines a complete random number generator that can be used to select bookkeepers as well as to generate validation challenges. We need to make a modification to VRF to make it work in our framework: in addition to the above three properties (probabilistic correctness, unique provability and residual pseudorandomness), we also require that the random numbers in our blockchain system be unpredictable, because if the random function can be predicted, then a miner's identity can exposed before he is finished verifying transactions, allowing him to be the subject of attacks which can result in the failure of bookkeeping.

There exists a concept called Verifiable Unpredictable

Functions (VUF) that has the same definition as VRF, and which satisfies properties 1 and 2 but modifies property 3 into property 4 below:

4. *Unpredictability*: for any algorithm T, for $* \neq x$:
$\text{Prob}\left[T^{\text{VUF.Evaluate}(SK,*)}(1^{\lambda}, PK) = \delta(SK, x)\right] \leq s(\lambda)^{-1}$.

In our case, we use a VRF that is also a VUF, that is, it satisfies condition 4 as well as 1-3. The method of adapting VRF to be unpredictable is found in [3] and is beyond the scope of this paper.

3. Sharding using VRF probabilities

We apply VRFs to node selection by using them in our sharding process. Assuming there are N nodes in the whole network, we attempt to select shards with m nodes. A random number R that is generated on the MultiVAC main chain is encrypted by node i according to each node's VRF private key, producing a 256-bit random number R_i. A node is picked into the shard if the following condition holds:

$$\frac{R_i}{2^{256}} \leq \frac{m}{N}$$

Thus, the probability of a node being selected as an in-shard node is:

$$p = \frac{m}{N}$$

Due to node selection being completely probabilistic, it is highly likely that the number of nodes in an actual shard is not equal to m. The probability that there are exactly k nodes in the shard is exactly:

$$p_{(k,m)} = \binom{N}{k} p^k (1-p)^{N-k}$$

$$= \frac{N!}{k!\,(N-k)!} \left(\frac{m}{N}\right)^k \left(1 - \frac{m}{N}\right)^{N-k}.$$

Note that for $k = 0$, this value is always greater than zero, thus there always exists a tiny non-zero chance that we produce empty shards, a probability that should not affect practical usage but which should be minimized. We can use the value $p_{(k,m)}$ to analyze the influences of the shard size on the reliability of the consensus in the shard.

In the most common case, N is very large and in particular, far larger than m and k. For this case we can simplify the above formula somewhat:

$$p_{(k,m)} = \frac{N \cdot (N-1) \cdots (N-k+1)}{N^k} \cdot \frac{m^k}{k!} \cdot \left(1 - \frac{m}{N}\right)^{N-k}$$

Since N is far larger than k,

$$\frac{N \cdot (N-1) \cdots (N-k+1)}{N^k} \approx 1.$$

Finally, as $N \to \infty$,

$$lim_{N\to\infty} \left(1 - \frac{m}{N}\right)^{N-k} = lim_{N\to\infty} \left(1 - \frac{m}{N}\right)^{N} = e^{-m}.$$

Thus, when N is very large,

$$\tilde{p}_{(k,m)} \approx \frac{m^k}{k!} e^{-m}.$$

Since this value is independent of N, a network with a sufficiently large node count has a shard structure only dependent on the desired shard size m, irrelevant of the number of nodes in the whole network.

Blockchain as a shard of the real world: As an aside, our definition of shard and the above formulation gives us another way to look at blockchains.

1) ***Blockchains are networked consensus systems which are subsets of the wider network of all connected things (the internet) and thus can be considered shards of the entire internet.***

2) ***The reliability of a blockchain's internal consensus is mainly related to its internal node count, and not related to what its size is in proportion to the wider internet.***

Any blockchain, including Bitcoin and Ethereum, has a reliability value directly positively related to the participant node number, i.e. the number of full nodes in the network: 10424 for Bitcoin [7] and 14383 for Ethereum [8]. We can consider all networked entities including people, objects, and machines as nodes in a massive 'real-world' network, with a blockchain connecting only being a subset of them. Compared with node counts in any particular blockchain, the size of the wider internet (the true value of N) is clearly infinitely larger. Our preliminary model applies directly to the wider internet and permits us to see any particular blockchain as a 'shard' of the real world internet, from which we also derive that the reliability of a blockchain is primarily related to its node number.

Conditions for consensus: We define a consensus algorithm's *margin of safety* ρ as follows: if a reliable consensus among m nodes is required then the proportion of honest nodes must not be less than ρ. We list some reference values for ρ below: In PBFT systems with sufficiently large node number, $\rho = 0.667$. In Algorand's BA$\star$ algorithm [4], the proposed value used in the consensus of each interim step is $\rho = 0.685$, and in the final step a stronger $\rho = 0.74$ is used.

We can now discuss conditions for consensus and also quantify the degree of reliability obtained by the network. Byzantine consensus algorithms use ρm as the threshold for successful consensus. Let μ be the proportion of honest nodes in the entire network and τ be the proportion of honest nodes in a shard. Then to reach reliable consensus in a shard we require:

$$constraint\ 1\colon \tau k \geq \rho m$$

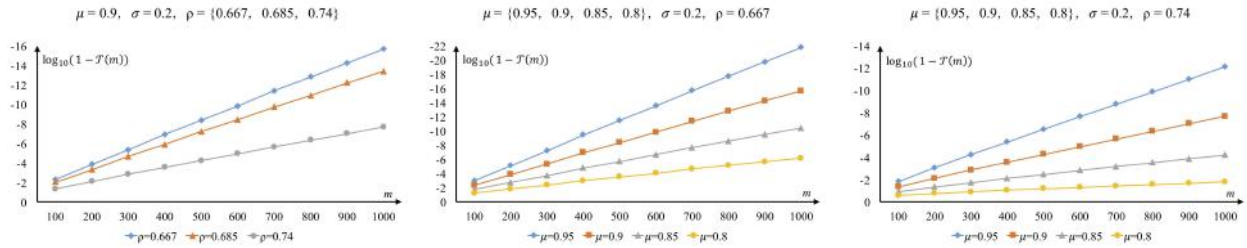
that is, the number of honest nodes is sufficient to reach consensus. We also require that

$$(1-\tau)k < \rho m$$

that is, the number of malicious nodes are too few to reach consensus.

Yet, the above inequality assumes an immediately synchronized network. When the network faces fluctuations or DDoS attacks, some honest nodes may fail to produce signals in time. Considering this, let σ be the proportion of non-responsive honest nodes, which can also be interpreted as the degree to which the network is severed, with $\sigma = 0$ implying a strongly synchronized network and $\sigma = 1$ implying complete network paralysis. We now refine our second constraint to prevent non-responsive nodes and malicious nodes from together causing the next block formation to fail:

$$constraint\ 2\colon (1 - \tau + \sigma\tau)k < \rho m$$



(a) (b) (c)

Fig. 1: The relationship between the network bifurcation probability $1-\mathcal{T}(m)$ and the number of nodes within the shard; the horizontal variable is the number of in-shard nodes m and the vertical variable is the logarithm of the bifurcation probability. (a) When $\mu = 0.9$, $\sigma = 0.2$, the effect on forking probabilities for the different algorithms under different shard sizes is shown. (b) When $\sigma = 0.2$, the effect of the proportion of malicious nodes in the network under a BFT algorithm is shown. (c) When $\sigma = 0.2$, the effect of the proportion of malicious nodes in the network under the BA * algorithm is shown.

In other words, a trusted consensus shall simultaneously satisfy:

$$\begin{cases} \tau k \geq \rho m \\ (1-\tau+\sigma\tau)k < \rho m \end{cases}$$

Note that when we have k nodes in a shard, the probability of having τk honest nodes and $(1-\tau)k$ malicious nodes, which we define as $P_{\tau,k}$, can be directly calculated from the probability $\tilde{p}_{(k,m)}$ above:

$$P_{\tau,k} = \tilde{p}_{(\tau k,\mu m)} \cdot \tilde{p}_{((1-\tau)k,(1-\mu)m)}$$

$$= \frac{(\mu m)^{\tau k}}{(\tau k)!} \cdot e^{-\mu m} \cdot \frac{[(1-\mu)m]^{(1-\tau)k}}{[(1-\tau)k]!} \cdot e^{-(1-\mu)m}$$

This is simplified to:

$$P_{\tau,k} = \frac{(\mu m)^{\tau k}}{(\tau k)!} \cdot \frac{[(1-\mu)m]^{(1-\tau)k}}{[(1-\tau)k]!} \cdot e^{-m}$$

Quantifying Reliability: In a shard built with size m, the reliability $\mathcal{T}(m)$ of reaching a consensus can be expressed as follows:

$$\mathcal{T}(m) = [1 - \text{Prob(constraint 1 fails)}] \cdot [1 - \text{Prob(constraint 2 fails)}]$$

We expand out $\mathcal{T}(m) =$:

$$\mathcal{T}(m) = \left[1 - \int_{\tau k=0}^{\rho m} \tilde{p}_{(\tau k,\mu m)}\right] \cdot \left[1 - \int_{(1-\tau)k=0}^{\infty} \int_{\tau k=\max\left\{\frac{k-\rho m}{1-\sigma},0\right\}}^{\infty} P_{\tau,k}\right]$$

which is integrated only in terms of τ and k as μ, ρ, σ are parameters taken as constants.

To solve for $\mathcal{T}(m)$, note that τk, $(1-\tau)k$ and ρm are all nonnegative integers and so the integrals in the above computation can be transformed into discrete summations. Note that $\mathcal{T}(m)$ is monotonic and thus invertible: knowing $\mathcal{T}(m)$ we can quickly calculate $m(\mathcal{T})$ and effectively estimate $m(\mathcal{T})$ through binary search.

As shown in Fig.1, when the node number increases continuously, the log of the network bifurcation probability $log_{10}(1-\mathcal{T}(m))$ is almost linear, showing that that reliability improves exponentially in m. In an example use case, suppose that the honest node proportion in the entire network is $\mu = 0.9$ and we adopt a PBFT or asynchronous BFT consensus $(\rho = 0.667)$ within the shard. If we assume that the proportion of nodes failing to respond is $\sigma = 0.2$, we find that $\mathcal{T}(200) = 0.9998$ and $\mathcal{T}(300) = 0.999995$. For reference, in a totally synchronized Bitcoin network with $\mu = 0.9$, Bitcoin has a reliability value [1] of 0.99976 after six confirmation blocks, slightly lower than $\mathcal{T}(200)$ under the above parameters.

Again suppose that the honest node proportion is $\mu = 0.9$, and that we adopt the BA⋆ consensus (with a more powerful $\rho = 0.74$) within the shard, maintaining the network severity parameter at $\sigma = 0.2$. We then obtain $\mathcal{T}(500) = 0.99994$. BA⋆ can also operate at $\rho = 0.685$ which gives $\mathcal{T}(300) = 0.99998$ and $\mathcal{T}(500) = 0.99999994$. With this comparison we see that a PBFT or asynchronous BFT algorithm reaches higher reliability with fewer nodes, at the cost of the higher communication cost of $O(m^2)$ required for consensus.

4. Transactions and Consensuses

Using our reliability model to pick m and using VRF to generate shards with random nodes, we can decompose the entire blockchain network into several shards with each transaction designated to a specific shard for execution. However, as with all sharding implementations it is challenging to design an appropriate mechanism to sync up all the shards' decisions and realize inter-shard coordination. A sharding solution needs to comprehensively consider the questions of how a ledger should be generated from in-shard transactions, whether the consensus reached within a shard is adequately secure, and how to handle transactions that straddle multiple shards.

Existing sharding technologies including Elastico[15] and Zilliqa[16] utilize a unified shared ledger. These are able to handle transactions in a sharded network but incur a heavy cost to synchronize the shards throughout the network, failing to optimally solve the sharding problem at its root. On the other hand, the Byzantine Shard Atomic Commit (Atomix) protocol designed by OmniLedger[17] conducts atomized processing on each transaction but uses logic that is complex and difficult to engineer.

MultiVAC's UTXO mechanism solves the synchronization problem. Each transaction is distributed by the network into different shards according to its account number, such that all the transactions of any given account are executed on the same shard. As shown in Fig.2, in the UTXO transaction mechanism, confirmation of the availability of funds is conducted only when funds are spent, not when funds are received. Each UTXO transaction takes one or more previously confirmed transactions as input and produces output that requires no processing by the recipient. By using a UTXO mechanism and always processing a particular user account within the same shard, we guarantee that the write operation is only performed on data within the same shard and that all cross-shard data interactions in our system are read-only, thereby preventing complicated cross-shard logic that inhibits other technologies' effectiveness. Our method is simple, easy to implement, and highly effective.

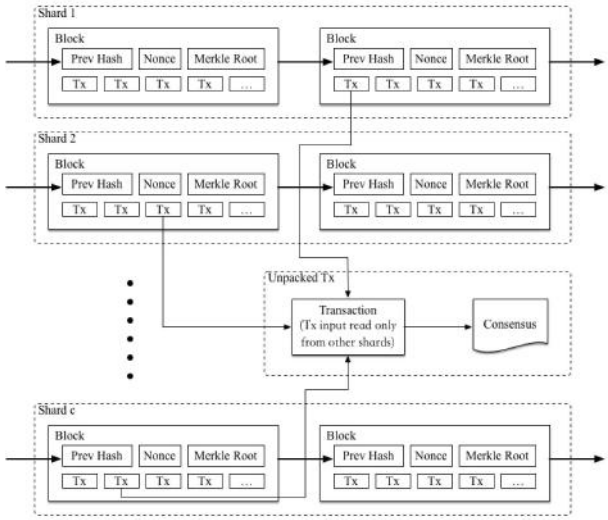


Fig. 2: The UTXO model in MultiVAC. The transactions are distributed into different shards for execution according to the payers' addresses. The inputs to UTXO are transactions that have already been confirmed on other shards, so cross-shard data interactions in our system are all read-only operations.

There is a potential problem with the shard UTXO method: attackers attempting to tamper with transactions or to perform double payment would only need to attack specific shards, as opposed to the network as a whole. This increases the chances of a successful attack. A common method to prevent this is *dynamic shard adjustment*, this is, to keep the users (or miners) on the same shard and randomly move the miners (or users) to different shards in a continuous fashion. In our implementation, we choose to dynamically adjust the miners of a shard. This makes attacks on any shard as difficult as attacking the network as a whole. MultiVAC additionally makes attacks harder by selecting in-node consensus algorithms that will not (or are very unlikely to) produce network forks, such as PBFT, asynchronous BFT or BA⋆. Erroneous blocks affected by malicious nodes would thus be left with a cryptographic trace. In this light, the PoW algorithm in Bitcoin is not applicable to in-shard consensus because the weak computational power of any single shard compared to the entire network makes it easier for the attacker to occupy the majority of computational power in the shard and create a fork.

Consensus for Transactions: Supposing the reliability requirement of each shard is q, then the shard size m must satisfy:

$$\mathcal{T}(m) \geq q.$$

Upon satisfying the reliability condition, we also wish to keep the cost to reach consensus in the entire network as low as possible. Suppose in every epoch the total transaction volume is t and the total number of shards is c, and suppose further that the communication time complexity to reach consensus within a shard is the function $f(m)$ of m and the cost for a single communication is $g(t/c)$. Then in terms of the average number of nodes in a shard, we wish our sharding plan over the entire network to satisfy:

$$\begin{aligned} \text{minimize} \quad & c \cdot f(m) \cdot g\left(\frac{t}{c}\right) \\ \text{subject to} \quad & \mathcal{T}(m) \geq q \end{aligned}$$

As $\mathcal{T}(m)$ is monotonic, the above optimization has a deterministic solution.

Consensus for Smart Contracts: We now extend our above analysis to include the case of smart contracts, which is more complicated than the case of transactions.

Considering a series of computational tasks $\{\Gamma_i\}$, $i = 0,1,2,3\ldots$, such that the corresponding runtime cost for each task is $|\Gamma_i|$, the required reliability level for each task is q_i, the size of the shard that each task is executed on is m_i, and the total communication cost $f_i \cdot g_i$, then the sharding plan within the entire network shall optimize:

$$\begin{aligned} &\text{minimize:} \\ &c \cdot f(m) \cdot g\left(\frac{t}{c}\right) + \sum \left(m_i \cdot |\Gamma_i| + f_i(m_i) \cdot g_i(\Gamma_i)\right) \\ &\text{subject to:} \\ &\mathcal{T}(m) \geq q \\ &\mathcal{T}(m_i) \geq q_i \\ &c \cdot m + \sum m_i \leq N. \end{aligned}$$

There is no global polynomial time solution for the above optimization problem. However, we can derive a qualitative conclusion from intuition: for a task with larger computational volume $|\Gamma_i|$, we would select a consensus algorithm with a higher communication cost but which uses fewer nodes within the shard to arrive at stronger consensus (i.e. asynchronous BFT). For a computation task with smaller computation volume $|\Gamma_i|$, we choose an in-shard consensus mechanism with a lower communication cost, such as BA⋆.

In summary, MultiVAC uses VRF to construct a probability model that splits user transactions and miner nodes into shards and then uses UTXO and the Byzantine consensus family to reach in-shard consensus. This completes the construction of our trusted sharding model. Together with the basic principles of security and decentralization, the trusted sharding model also has large scalability implications for public blockchains, because it allows for blockchain throughput to increase without limit with the number of nodes.

For ordinary public-chain transactions, the consensus strength in a single MultiVAC shard is adequate to achieve a high level of reliability. However, for DApps and smart contracts, it is quite wasteful to require each line of code to run on hundreds or thousands of different nodes. Is there a method that can use even fewer or an optionally limited number of nodes and still achieve trustworthy smart contract executions in a decentralized trustless network? On the basis of our VRF sharding mechanism, we achieve this by creating a MVM virtual machine equipped with a custom BISC instruction set and PoIE consensus.

5. On VMs and Instruction Sets

Virtual Machines provide an excellent sandbox environment for executing smart contracts. For public chains that should be capable of general computation and unlimited scalability, the design of the virtual machine's instruction set is of vital importance. Mainstream virtual machines and instruction sets are rather unoptimized for complicated business logic in smart contracts. We thus create our own specialized blockchain-dedicated instruction set, the BISC (Blockchain Instruction Set Computer). On this basis we create our general-purpose virtual machine, the MultiVAC Virtual Machine (MVM).

5.1 Design Goals

Virtual machines need not stay virtual. In the long term, a blockchain virtual machine may be implemented directly as a specialized hardware CPU. This would make blockchain transactions faster and immensely more powerful. For this to

happen, the blockchain instruction set used in the virtual machine should be mature and efficient, able to support complicated top-layer contract logic with a complicated and robust base-layer architecture.

Based on this long-term vision, we design the MVM and the BISC instruction set with the following features:

1. **Support for General-Purpose Computation.** Blockchain VMs today are rather limited in handling complicated general-purpose computation. Future smart contracts and DApps require VMs to not only be Turing-complete but also for their instruction sets to support more complicated logic.
2. **Support for Compilation from Multiple High Level Languages.** MultiVAC is an open-source ecosystem designed to be highly friendly to developers, providing a robust compilation environment for many high level languages to support smooth migration of existing programs onto our platform.
3. **Effective Use of Hardware, Allowing for Implementation of our Instruction Set as a Hardware Computer**
   Present-day blockchain systems cause low-level hardware to suffer a large loss in potential performance when compiling or interpreting VM bytecode. MVM redesigns and upgrades a mature CPU instruction set, holding the potential to one day be directly installed as a hardware computer. This makes it possible for computers to naturally become MultiVAC nodes while still being computers used for desktop or mobile purposes, and would allow for a seamless switch between personal computer and miner.

5.2 The BISC Instruction Set

The MVM uses a flexible and custom-made instruction set BISC. BISC is based on the outstanding reduced instruction set RISC-V [22], which has a mature instruction architecture and an excellent open-source compilation environment. BISC customizes RISC-V for blockchain by adding 256-bit instruction processing plus signature and hashing instructions for public blockchains. The development of BISC will be in line with global open-source principles.

BISC supports a complete and tidy set of instruction sequences as shown in Table 1. There are multiple sets of instructions, named as follows: Instructions labeled with prefix RV are from the standard extensions defined by RISC-V, while those labeled BRV are newly defined for BISC. The numbers following RV and BRV refer to the instruction bitwidth and the suffix signifies the instruction's functions. The suffix G is a joint label that covers RISC-V's base pack I and the four standard extensions MAFD. These instructions, especially RV32G and RV64G, have the strong support of the RISC-V community. Additional RISC-V extensions have suffix L and B whereas instructions newly defined for BISC have suffix H and X.

Table 1: BISC Instruction Pack

Instruction Computer	Instruction Extension Pack	Instruction Description	BISC Instructions
G instructions Standard RISC-V set contains the basic I instruction and four kinds of extension packs of MAFD.	I instructions	Basic access, computation and controlling operation of integers	RV32G RV64G BRV256I BRV256M BRV256A
	M instructions	Multiplication and division operations of integers	
	A instructions	Trans-processor atom manipulation instructions such as synchronous reading and writing etc.	
	F instructions	Single-precision floating number operation instructions	
	D instructions	Double-precision floating number operation instructions	
L instructions		Decimal integer operation instructions	BRV256L
B instructions		Bit manipulation instructions	BRV256B
H instructions		Signature and hash instructions	BRV256H
X instructions		Encryption and decryption instructions	BRV256X

The BISC instruction set framework currently supports C compilation based on LLVM, the GDB debugger, and the glibc standard library. LLVM (Low Level Virtual Machine) is a compiler framework whose purpose is to construct a compile-time, link-time and run-time executor for any programming language. The LLVM compilation framework with RISC-V as the back-end will eventually support high-level languages such as Java and Go. Its overall architecture is shown in Fig. 3.

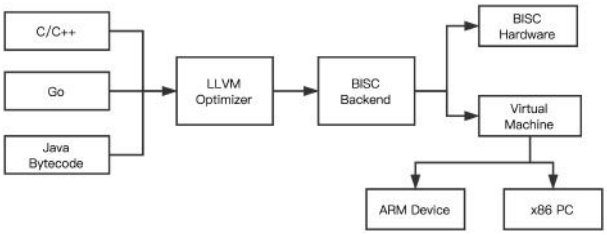


Fig. 3: LLVM compilation framework based on BISC

5.3 The MVM Virtual Machine

The MVM Virtual Machine is a blockchain VM designed to support flexible computational models, capable of providing an efficient and verifiable execution environment for smart contracts sourced from high-level Turing-Complete programming languages. MVM provides applications with static code optimization, storage allocation, run-time inspection, and execution-time verification.

To prevent infinite-loop attacks, MVM adopts gas charges similar to Ethereum for each BISC instruction executed. Because each executed instruction in a smart contract incurs a charge, smart contracts must be executed in the most computationally efficient way possible, requiring code optimization. To do this, MVM will include for developers a targeted suggestion and optimization engine in its test environment that will provide breakdowns of executed instructions and their gas costs, and it will also provide in the compilation environment suggestions for code optimization.

Other than completing execution in limited time, smart

contracts in our flexible computational system must also be verified by honest work. The PoIE consensus algorithm directly embedded into the MVM platform achieves this, performing computation, gas charging, and verification concurrently upon every executed instruction. Note that gas charges will only be levied on smart contract execution steps and not on the computational steps required for the verification logic or for gas charging itself. When an instruction sequence with sufficient gas is completed and verified, the node will issue the computational results through the consensus mechanism and will receive a gas reward.

To facilitate processing, MVM provides a BISC-compatible memory model that isolates a computer's physical memory, providing flexible run-time support through our built-in stack and heap space. The stack space provides sufficient call depth to support various types of complicated data structures and may also provide batch IO stack operations. The heap space is capable of being freely allocated and supports random addressing and also provides a monitoring mechanism to recover used resources, in sum guaranteeing memory allocation for general-purpose computation.

MVM can operate on all the network nodes, allowing the nodes providing computational services to schedule tasks by adding them to their priority queues in order of their gas price and to execute them in order of priority.

6 PoIE Consensus

Existing sharding technologies such as those proposed by Ethereum [20], Zilliqa[16] and Elastico[15] require a large number of nodes per shard, usually in the hundreds to low thousands. DApps are composed of smart contracts on the public blockchain, and requiring all DApp code to run on hundreds or thousands of nodes is clearly too expensive. In a system of untrusted nodes such as the blockchain, is there a way to execute the computation task on only a tiny number of nodes such that that the soundness of both the execution process and that of the obtained result are verifiable by the network as a whole?

6.1 Theoretical and Realistic Basics for PoIE

Proposed by researchers at Tel Aviv University and MIT, zk-SNARKs[14] can verify the execution of a program without first divulging the program's data, via solving the program's zero-knowledge proofs. zk-SNARKs create concise non-interactive zero knowledge proofs by flattening the program (a transaction or smart contract) into base expressions functioning much like logic gates in a circuit. By encoding the program code into a circuit and providing a proof statement to the verifier, zk-SNARKs can verify non-interactively whether or not a computation task has actually been executed.

One might design a shard-based internal consensus algorithm based on zk-SNARKs. The benefit of this is that the number of nodes within a shard is very small but they can still reach a high degree of consensus, one that is easily verifiable by the out-of-shard nodes. This is a very important quality to have in an effective consensus system: the nodes which did not participate in program execution can still verify that they were executed correctly. However, zk-SNARKs suffer from extremely high time complexity. For any program $\mathbb{P}$ and a time bound T, the time complexity to execute zk-SNARK verification is $O(|\mathbb{P}| \cdot T)$ [9][10][14], and thus they are not practically applicable to public blockchain systems.

MultiVAC introduces a brand-new consensus algorithm called PoIE (Proof of Instruction Execution), a proof on the base layer of instruction sequences. zk-SNARKs are purely mathematical algorithms for verification, but PoIE is based on physical computational constraints. The basic design principle is that malicious nodes must incur a high real-world physical cost in order to defraud, and that even if they defrauded they would be able to receive a reward but would still not be able to overturn the computation's verified correctness. From the perspective of costs, malicious nodes thus have a great incentive to honestly execute computational tasks.

The physical cost used by PoIE is as follows: We treat a program to be executed as a base-layer instruction sequence. For modern computers, the cost of executing this sequence is far less than the cost of storing this sequence in memory, the physical constraint we use to ensure reliability. In reality, the processing speed of modern computers is often equal to that of their CPU cache and far greater than their read/write speed on memory. Even though CPU cache can reach the same processing speed as the CPU itself, even in high-end CPUs (i.e. the Intel Core i7 series) the cache is only 8-12MB but consists of 1/4 to 1/3 of its computational costs (in terms of number of transistors).

Many technologies in the world are designed from similar insights. The physical foundations of PoIE have some similarity with Ethereum's mining mechanism ETHash. ETHash was made to resist ASIC mining and avoid Bitcoin-level mining-pool centralization by requiring miners not only to perform hashing but also to randomly and frequently read large amounts of data from memory. This memory read requirement creates a bottleneck for specialized ASIC miners, preventing mining from becoming a highly specialized and centralized activity. Similarly to ETHash, PoIE uses the physical discrepancy between computation and storage in modern computers to penalize malicious behavior.

6.2 The PoIE algorithm

PoIE is an instruction set based consensus embedded into the virtual machine. Its design philosophy is to consider the program execution as a string of execution instructions. PoIE can verify if this instruction sequence has been honestly executed in a network with untrustworthy nodes and can distribute appropriate economic rewards for honest execution.

6.2.1 Preliminaries

First, we define an anti-collision hash function with safety parameter λ:

$$hash: \{0,1\}^* \to \{0,1\}^{O(\lambda)}$$

MultiVAC uses the Merkle Tree data structure to perform verification. A Merkle Tree is a tree-based data structure used for efficient verification of contents. For a data set $S = \{a_i\}, i = 1,2 \ldots n$, we build a binary Merkle Tree on S denoted $M(S: 1 \to n)$ as follows:

$$M(a_i) = hash(0x00 \cup a_i)$$
$$M(S: i \to i+1) = hash(0x01 \cup M(i) \cup M(i+1))$$
$$M(S: 1 \to n) = hash(0x01 \cup M(S: 1 \to 2^{\lfloor log_2^{n-1} \rfloor}) \cup M(S: 2^{\lfloor log_2^{n-1} \rfloor} + 1 \to n))$$

The classic application of Merkle Trees in blockchain are their uses in packaging transactions in Bitcoin as well as in the proof of replication in Filecoin.

PoIE requires a computationally complete hidden verification function, Scalable Computational Integrity and Privacy (SCIP) [9][10][14]. SCIP is a triad:

$$\text{SCIP} = (\text{Setup}, \text{Prove}, \text{Verify})$$

and is a process of zero knowledge verification that hides the execution proof of PoIE to prevent a third party from copying

the instruction set sequence.

6.2.2 Homomorphic Hiding

For any program decomposed into an instruction set sequence Γ, PoIE allows the instruction executor (Prover, $\mathcal{P}$) to generate a proof $\pi(\Gamma)$ in linear $O(|\Gamma|)$ time which enables the verifier (Verifier, $\mathcal{V}$) to verify in $O\big(log(|\Gamma|)\big)$ time that the instruction set sequence has been correctly executed. To simplify our presentation, we combine the output Θ (if any) of the instruction set sequence into Γ so Γ considers both instruction set sequence and its result.

A node owns a public-secret key pair $\{PK, SK\}$ in addition to another pair of public-secret keys $\{P_{HH}, S_{HH}\}$ used to hide information. First, we conduct Homomorphic Hiding (HH) on the instruction set sequence of each executor, a triad expanded below,

$$\text{HH} = (\text{Generate}, \text{Prove}, \text{Verify}).$$

We describe each operation in HH as follows:

$$\text{HH.Generate}(\{PK, SK\}, S_{HH}, \Gamma) \rightarrow \{\Lambda, \pi_{HH}\}.$$

HH.Generate generates a hidden version Λ of the instruction set Γ and a proof π_{HH} provided to the executor $\mathcal{P}$ to generate a proof about Λ and Γ. Λ and Γ need to be doubly generated below in section 6.2.3.

The pseudo-code for HH.Generate is below:

HH.Generate

INPUTS:
- Key pair $\{PK, SK\}$
- Hide key S_{HH}
- Instruction list Γ

OUTPUTS:
- Encrypted Instruction List Λ
- Prove π_{HH}

PROCEDURE:
- Compute: $\Lambda \leftarrow Encrypt(\Gamma, SK)$
- Set: $\vec{x} \leftarrow \{PK, \Lambda\}$
- Set: $\vec{w} \leftarrow \{SK, \Gamma\}$
- Compute: $\pi_{HH} \leftarrow \text{SCIP.Prove}(S_{HH}, \vec{x}, \vec{w})$
- Output: Λ, π_{HH}

Observe that encryption of Λ requires only verification and not reverse decryption. Thus, we may use easily computable one-way encryptions instead of high-cost encryptions such as elliptic curves or RSA.

$$\text{HH.Prove}(S_{HH}, \Lambda, \varepsilon) \rightarrow \pi^{\varepsilon}_{POST}$$

Using Λ, HH.Prove generates for the executor $\mathcal{P}$ a proof corresponding to the challenge ε proposed by the verifier $\mathcal{V}$.

$$\text{HH.Verify}(PK, P_{HH}, \Lambda, \pi_{HH}, \varepsilon, \pi^{\varepsilon}_{POST}) \rightarrow (\text{true}|\text{false})$$

HH.Verify is used by the verifier $\mathcal{V}$ to check the authenticity of π^{ε}_{POST}. HH.Prove and HH.Verify are both generated using SCIP.

6.2.3 The Main Algorithm

Now we present the full PoIE algorithm. This is also a triad:

$$\text{PoIE} = (\text{Generate}, \text{Prove}, \text{Verify})$$

We expound on each of the individual functions below.

$$\text{PoIE.Generate}(\{PK, SK\}, S_{HH}, \mathbb{P}) \rightarrow \{\text{Root}_{\text{M}(\Gamma)}, \text{Root}_{\text{M}(\Lambda)}, \pi_{HH}\}$$

$\mathbb{P}$ is the program code to be executed. PoIE.Generate creates a Merkle Tree root node from the instruction sequence Γ and the hidden instruction sequence Λ generated by $\mathbb{P}$'s execution. These operations are executed simultaneously in the CPU without recording Γ or Λ.

The pseudo-code for PoIE.Generate is below:

PoIE.Generate

INPUTS:
- Key pair $\{PK, SK\}$
- Hide key S_{HH}
- Program $\mathbb{P}$

OUTPUTS:
- Root of Merkle Tree $\text{Root}_{\text{M}(\Gamma)}$
- Root of Merkle Tree $\text{Root}_{\text{M}(\Lambda)}$
- Prove π_{HH}

PROCEDURE:

Synchronized:
$$\begin{cases} \Gamma \leftarrow \text{Run } \mathbb{P} \\ M(\Gamma) \leftarrow \text{Merkle Tree of } \Gamma \\ \{\Lambda, \pi_{HH}\} \leftarrow \text{HH.Generate}(\{PK, SK\}, S_{HH}, \Gamma) \\ M(\Lambda) \leftarrow \text{Merkle Tree of } \Lambda \end{cases}$$

- Set: $\text{Root}_{\text{M}(\Gamma)} \leftarrow$ Root of $M(\Gamma)$
- Set: $\text{Root}_{\text{M}(\Lambda)} \leftarrow$ Root of $M(\Lambda)$
- Output: $\text{Root}_{\text{M}(\Gamma)}, \text{Root}_{\text{M}(\Lambda)}, \pi_{HH}$

PoIE.Prove defines an interactive process requiring a two-phase commit protocol, meaning that another execution of $\mathbb{P}$ is performed that generates a new $M(\Lambda)$, constructing proof for the challenge ε given by the verifier, up until the point where all challenges have been queried:

$$\text{PoIE.Prove}(\{PK, SK\}, S_{HH}, \mathbb{P}, \varepsilon) \rightarrow \{\pi^{\varepsilon}_{\Gamma}, \pi^{\varepsilon}_{\Lambda}, \pi^{\Lambda(\varepsilon)}_{POST}, \Lambda(\varepsilon)\}$$

The pseudo-code of PoIE.Prove is below:

PoIE.Prove

INPUTS:
- Key pair $\{PK, SK\}$
- Hide key S_{HH}
- Program $\mathbb{P}$
- Challenge ε

OUTPUTS:
- Prove $\pi(\Gamma) = \{\pi^{\varepsilon}_{\Gamma}, \pi^{\varepsilon}_{\Lambda}, \pi^{\Lambda(\varepsilon)}_{POST}, \Lambda(\varepsilon)\}$

PROCEDURE:

Synchronized:
$$\begin{cases} \Gamma \leftarrow \text{Run } \mathbb{P} \text{ until } \varepsilon \text{ is finished} \\ \text{M}(\Gamma) \leftarrow \text{Merkle Tree of } \Gamma \\ \{\Lambda, \pi_{HH}\} \leftarrow \text{HH.Generate}(\{PK, SK\}, S_{HH}, \Gamma) \\ \text{M}(\Lambda) \leftarrow \text{Merkle Tree of } \Lambda \\ P\text{ath}_{\varepsilon,\Gamma} \leftarrow \text{Merkle path of } \Gamma(\varepsilon) \text{ in M}(\Gamma) \\ P\text{ath}_{\varepsilon,\Lambda} \leftarrow \text{Merkle path of } \Lambda(\varepsilon) \text{in M}(\Lambda) \\ \pi^{\Lambda(\varepsilon)}_{POST} \leftarrow \text{HH.Prove}(S_{HH}, \Lambda(\varepsilon), \varepsilon) \end{cases}$$

- Set: $\vec{x}_{\Gamma} \leftarrow \{\text{Root}_{\text{M}(\Gamma)}, \varepsilon\}$
- Set: $\overrightarrow{w_{\Gamma}} \leftarrow \{P\text{ath}_{\varepsilon,\Gamma}, \Gamma(\varepsilon)\}$
- Compute: $\pi^{\varepsilon}_{\Gamma} \leftarrow \text{SCIP.Prove}(S_{HH}, \vec{x}_{\Gamma}, \overrightarrow{w_{\Gamma}})$
- Set: $\vec{x}_{\Lambda} \leftarrow \{\text{Root}_{\text{M}(\Lambda)}, \varepsilon\}$
- Set: $\overrightarrow{w_{\Lambda}} \leftarrow \{P\text{ath}_{\varepsilon,\Lambda}, \Lambda(\varepsilon)\}$
- Compute: $\pi^{\varepsilon}_{\Lambda} \leftarrow \text{SCIP.Prove}(S_{HH}, \vec{x}_{\Lambda}, \overrightarrow{w_{\Lambda}})$
- Output: $\pi^{\varepsilon}_{\Gamma}, \pi^{\varepsilon}_{\Lambda}, \pi^{\Lambda(\varepsilon)}_{POST}, \Lambda(\varepsilon)$

Finally, the verifier $\mathcal{V}$ verifies the computation given its input using PoIE.Verify:

$$\text{PoIE.Verify}\left(PK, P_{HH}, \text{Root}_{\text{M}(\Gamma)}, \text{Root}_{\text{M}(\Lambda)}, \pi_{HH}, \varepsilon, \pi(\Gamma)\right) \rightarrow (true|false)$$

The pseudo-code for PoIE.Verify is below:

PoIE.Verify

INPUTS:

Public Key of Prover PK

Public Key of HH P_{HH}

Root of Merkle Tree $\text{Root}_{\text{M}(\Gamma)}$

Root of Merkle Tree $\text{Root}_{\text{M}(\Lambda)}$

Prove π_{HH}

Challenge ε

Prove $\pi(\Gamma) = \{\pi_{\Gamma}^{\varepsilon}, \pi_{\Lambda}^{\varepsilon}, \pi_{POST}^{\Lambda(\varepsilon)}, \Lambda(\varepsilon)\}$

OUTPUTS:

True or false

PROCEDURE:

Set: $\vec{x}_1 \leftarrow \{\text{Root}_{\text{M}(\Gamma)}, \varepsilon\}$

Compute: $v_1 \leftarrow \text{SCIP.Verify}(P_{HH}, \vec{x}_1, \pi_{\Gamma}^{\varepsilon})$

Set: $\vec{x}_2 \leftarrow \{\text{Root}_{\text{M}(\Lambda)}, \varepsilon\}$

Compute: $v_2 \leftarrow \text{SCIP.Verify}(P_{HH}, \vec{x}_2, \pi_{\Lambda}^{\varepsilon})$

Compute: $v_3 \leftarrow \text{HH.Verify}\left(PK, P_{HH}, \Lambda(\varepsilon), \pi_{HH}, \varepsilon, \pi_{POST}^{\Lambda(\varepsilon)}\right)$

Output: $v_1 \wedge v_2 \wedge v_3$

This allows for an interactive verification process executed on the instruction set sequence. Since the cost of executing instructions is much lower than that of storing them in memory, an attacker will incur a high cost if they chose to store or copy Γ to construct Λ. This makes it not cost-effective to launch an attack. It also goes without saying that the cost of storing and constructing $M(\Gamma)$ and $M(\Lambda)$ in memory is also extremely high.

6.3 Flexible Sharding Computation

Requirements of consistency, availability and partition tolerance are difficult to equally satisfy in the design of any distributed system. Different contracts and DApps have different levels of requirements for these properties, but almost all public blockchains have a fixed compromise between them. MultiVAC is unique among public blockchains in that its flexible computation model provides infrastructure guaranteeing that DApp designers have space to decide on their own the tradeoff between decentralization, scalability and security. Given the VRF sharding process and the PoIE task verification process, MultiVAC allows the users who submitted tasks to select a required reliability level based on actual business demand, and based on this to select a shard size and corresponding consensus mechanism.

For a computation task Γ, MultiVAC allows the task submitter to choose to run their task inside of a shard with a certain size in order to reach the reliability requirement q. We define the communication complexity to reach consensus within the shard in terms of shard size m as a function $f(m)$. Also we define the cost of a single communication as a function of the proof $g(\Gamma)$. Note that $g(\Gamma)$ is the data volume that the PoIE algorithm needs to interact with and has complexity $O(log|\Gamma|)$. $f(*)$ is fixed by the consensus type chosen (i.e. asynchronous BFT or BA$\star$), so given a consensus algorithm $\circledast$ we denote the consensus-specific communication complexity as $f^{\circledast}(*)$. We also denote the reward that miners are able to receive as $e^{\circledast}(|\Gamma|)$ and the node count involved in distributing the reward as $h^{\circledast}(m)$.

The submitter of the computation task aims to achieve a desired reliability level at the minimum possible cost. MultiVAC's public blockchain aims at using minimum possible system resources to reach the reliability requirements. Therefore, the decision $D(m,\circledast)$ made may be expressed as:

$$\arg\min D(m,\circledast) = \alpha \cdot [f^{\circledast}(m) \cdot\ g(\Gamma) + m \cdot |\Gamma|] + \beta \cdot h^{\circledast}(m) \cdot e^{\circledast}(|\Gamma|)$$
$$\text{subject to } \mathcal{T}(m) \geq q, \quad \text{w.r.t.} \rho^{\circledast}$$

where α and β are weight parameters.

The task submitter would prefer to set $\alpha = 0$, ignoring the needs of the system in order to maximize his or her self-interest. Because of this, the final decision-making power of $D(m,\circledast)$ remains with the MultiVAC public blockchain. The user will be able to request a reliability level and some economic considerations, but the final selection of shard size m and consensus algorithm $\circledast$ are still decided by the MultiVAC program.

7 Storage, Transmission and Computation

It must be noted that computers do not only compute; they also store and transmit data. A robust public blockchain system should be able to achieve the three desiderata of security, scalability, and decentralization not only for computation but also for storage and transmission. This in turn requires well-designed incentive mechanisms to encourage nodes to contribute resources for all three. MultiVAC is the first scalable public blockchain that designs for all three of dimensions of blockchain robustness (computation, storage and transmission).

7.1 Computation

We have already discussed computation in the above sections. MultiVAC is the first system to provide a flexible sharding solution for blockchain computation, using PoIE to verify the correctness of each computation. PoIE provides both the actual sequence of executed instructions Γ as well as the instruction sequence after the homomorphic hide Λ. Based on the execution status inferred by Λ, we can easily design a reward system similar to the gas incentive of Ethereum. Its reward function is:

$$Reward_i(\Lambda_i, \text{PoIE.Verify}) = \begin{cases} |\Lambda_i| & \text{PoIE.Verify} \rightarrow \text{true} \\ \emptyset & \text{PoIE.Verify} \rightarrow \text{false} \end{cases}$$

7.2 Storage

MultiVAC is equipped with high-performance transaction processing that improves with the number of nodes in the network. If the average realized throughout of a public blockchain is >1,000 tps and the average transaction size is 0.4KB, the blockchain ledger will have an annual file size of over 10TB. Clearly, normal PCs are unable to store such large ledgers and so we either require the usage of supernodes or shard storage.

There are many distributed storage projects such as Storj[25], MaidSafe[27] and Siacoin[26] and Filecoin[24]. Filecoin[24] takes IPFS[23] as its base mechanism, which is a complete decentralized and distributed storage system with an addressable, versioned, and peer-to-peer file system. Some well-known blockchains including EOS also adopt IPFS.

Slightly different from IPFS which is based on Hash addressing, MultiVAC also uses a storage and search mechanism based on Merkle Roots. Merkel Roots have many benefits. They not only enable us to search and retrieve data, they have the additional capability of allowing us to search and retrieve only a small chunk of the data while still obtaining

verification of the full data's existence and authenticity. MultiVAC supports file storage and retrieval based on both Hash and Merkle Roots. In addition, MultiVAC also includes a VRF sharded storage mechanism, which is a distributed and decentralized storage system.

Similar to Bitcoin light nodes, the MultiVAC nodes only store block header information, maintaining the full transaction input and output in distributed storage. It is important to note that in MultiVAC, the data storage mechanism is only used as an internal base-layer service for the system, so that the storage mechanism is unable to edit the data. All the rules for data generation, modification, deletion, as well as verification and consensus are delegated to the platform's higher-level functions. The only thing that the base-layer does is to store and retrieve data for the higher-levels. MultiVAC will provide a reward for nodes performing both storage and computational services.

7.2 Transmission

Finally, a blockchain network also must consider data transmission issues. Systems utilizing a sharded storage ledger reduce their storage costs in exchange for increased transmission costs, though this issue may be relieved somewhat as IPFS has proven that the usage of a distributed storage also brings with it distributed transmission capability, which may reduce bandwidth pressures on centralized nodes.

Suppose in a blockchain a node processes t transactions before forming a block. If the entire network stores the ledger then there will be a disc IO time cost of $O(t)$ and a network syncing cost of $O(t)$. If we use shard storage, there will be no disc IO time cost, a network syncing cost of $O(t)$, and an additional network communication cost for verification of $O(t)$. As the transaction process likely takes place over a fragmented network instead of a synchronized network, the time cost of syncing will actually be in practice somewhat higher than that of the local disc IO, however in principle sharding the ledger's storage does not increase the time complexity of the transmission.

Discovering appropriate incentive mechanisms for data transmission remains an open question in academia and industry and no fully effective solutions have been presented as of date. Even in the mechanisms of IPFS and filecoin where storage nodes may receive rewards through two mechanisms PoRep and PoST, the storage nodes may still refuse to transmit data when other nodes require it due to reasons such as bandwidth cost. In addition, data transmission may be so frequent such that it is impossible to generate a corresponding reward transaction for each data request, because the reward transaction itself will result in its own data transmission costs, leading to an infinite recursion. A well-designed incentive mechanism for data transmission would take into consideration issues such as bandwidth, latency, transition volume, and request frequency, and these many variables cause the data transmission reward question to remain an unsolved problem in the near-term. However, this mechanism is not an urgent objective as data transmission is never decoupled from the operations of storage and computation which each already have their own incentive mechanisms.

In summary, MultiVAC comprehensively considers the three dimensions of computation, storage, and transmission in modern blockchains, and we design an incentive mechanism for computation and storage. We are the first scalable public blockchain that designs for all three dimensions.

8. Conclusions

MultiVAC designs a high-performance public blockchain where nodes are randomly sharded based on VRF and where reliability is guaranteed with a probability model. Unlike all public blockchains available today, our flexible platform provides users of smart contracts the ability to self-select the balance between security, decentralization, and scalability. Unlike Bitcoin or Ethereum, the processing capacity of the MultiVAC network will be continuously increased as number of nodes increases and as the total computational power of the network expands, making the blockchain infinitely scalable and capable of being used in a myriad of business and industrial applications. In terms of business support, our distributed computation platform provides a revolutionary breakthrough in the blockchain industry with our novel BISC instruction set, our MVM virtual machine and our PoIE consensus, and this allows our platform to be able to supply an ever-increasable level of resources to distributed applications.

(A Final Sidenote: The name MultiVAC is derived from the name of the supercomputer in Isaac Asimov's *The Last Question*. MultiVAC evolved from our present-day transistor based computers into an entity existing in hyperspace beyond gravity or time, having merged with all the human souls in the universe. In the last days of the universe, MultiVAC finally discovers the answer to the question, "How can the net amount of entropy of the universe be massively decreased?," and thus makes the pronouncement, "Let there be light".)

References

[1]. Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, October 31, 2008

[2]. M Castro, B Liskov. Practical Byzantine fault tolerance. Symposium on Operating Systems Design & Implementation, 1999 , 20 (4) :173-186

[3]. S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), New York, NY, Oct. 1999.

[4]. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies, MIT CSAIL, Arxiv: 1607.01341

[5]. D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT '01,pages 514–532, London, UK, 2001. Springer-Verlag.

[6]. B David, P Gazi, A Kiayias,and A Russell. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. International Conference on the Theory & Applications of Cryptographic Techniques, 2018: 66-98

[7]. Global Bitcoin Nodes Distribution, website: https://bitnodes.earn.com/

[8]. Ether Nodes Network Number, website: https://www.ethernodes.org/network/1

[9]. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 626–645. Springer, 2013.

[10]. Nir Bitansky, Alessandro Chiesa, and Yuval Ishai.

Succinct non-interactive arguments via linear interactive proofs. Springer, 2013.
[11]. Vitalik Buterin, A Next-Generation Smart Contract and Decentralized Application Platform, 2013.
[12]. Dr Gavin Wood, Ethereum: A Secure Decentrailised Generalised Trasactioin Ledger.
[13]. The Block.One. EOS.IO Technical White Paper. March 2018.
[14]. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In Advances in Cryptology–CRYPTO 2013, pages 90–108. Springer, 2013.
[15]. L Luu, V Narayanan, C Zheng, K Baweja and S Gilbert.A Secure Sharding Protocol For Open Blockchains.Acm Sigsac Conference on Computer & Communications Security. 2016 :17-30
[16]. The Zilliqa Team, The ZILLIQA Technical Whitepaper, 2017.
[17]. Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, Bryan Ford, OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding, 2017
[18]. The Seele Team. Seele Tech Whitepaper: Innovate New Era of Value Internet. 2018.
[19]. Timo Hanke, Mahnush Movahedi and Dominic Williams. DFINITY Technology Overview Series: Consensus System. Jan. 2018.
[20]. Vitalik Buterin. Ethereum 2.0 Mauve Paper. 2016.
[21]. Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. 2014 USENIX Annual Technical Conference. June 2014: 305-219.
[22]. Andrew Waterman, Krste Asanovie, The RISC-V Instruction Set Manual, May 7, 2017.
[23]. Juan Benet. IPFS - Content Addressed, Versioned, P2P File System. 2014.
[24]. The Filecoin Team. Filecoin: A Cryptocurrency Operated File Storage Network. July 2014.
[25]. Shawn Wilkinson, Tome Boshevski, Josh Brandoff, James Prestwich, Gordon Hall, Patrick Gerbes, Philip Hutchins and Chris Pollard. Storj: A Peer-to-Peer Cloud Storage Network. Dec 2016.
[26]. David Vorick and Luke Champine. Sia: Simple Decentralized Storage. Nov 2014.
[27]. The Maidsafe Team. A Safe Network Premier: An Introductory Guide's to the World's First Fully Autonomous Data Network. Feb 2018.
[28]. Joseph Poon and Vitalik Buterin. Plasma: Scalable Autonomous Smart Contracts. August 2017.
[29]. Isaac Asimov. The Last Question. Science Fiction Quarterly. Nov 1956