Verkle-Accumulator-Based Stateless Transaction Validation (VA-STV) Scheme for the Blockchain-Based IoT Network

Zhaohui Guo, Zhen Gao, *Member, IEEE*, Qiang Liu, *Senior Member, IEEE*, Lei Liu, *Member, IEEE*, Mianxiong Dong, *Senior Member, IEEE*, Ning Zhang, *Senior Member, IEEE*, and Mohammed Atiquzzaman, *Senior Member, IEEE*

***Abstract*—The blockchain-based Internet of Things (IoT) has served widely across various industries for authentication, cooperation, and data sharing but faced the severe challenge of storage scalability. The storage burden gets worse for IoT devices with limited resources. The state data is essential for efficient transaction issuance and validation. This article proposes the Verkle accumulator-based stateless transaction validation (VA-STV) scheme for permissionless blockchains to decrease the storage burden of the state data on each node with the acceptable overhead of computation and communication. In the scheme, the current state is summarized as the commitment maintained in the latest block header, and one witness is generated for each token to guarantee its validity. State transitions are realized by updating the commitment and witnesses so that no state is stored on nodes acting as validators and miners. Only the nodes acting as traders should maintain the tokens controlled by themselves and the witnesses locally. The VA-STV is based on the Verkle accumulator (VA), which is a combination of the Verkle tree (VT) and the KZG polynomial commitment scheme. Simulation results show that the VA-STV provides a smaller witness size (0.6×–0.74×) and faster commitment generation (6×–14×) than the existing stateless schemes in the same settings, which indicates the advantages of VA-STV in succinctness and efficiency. Besides, a tradeoff between the communication and computation requirements can be achieved by adjusting the branching factor, which improves the adaptability of the proposed scheme for different IoT scenarios.**

***Index Terms*—Blockchain, Internet of Things (IoT), KZG polynomial commitment, stateless, Verkle tree (VT).**

Manuscript received 16 May 2023; accepted 7 June 2023. Date of publication 19 June 2023; date of current version 25 December 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFE0205300, and in part by the National Natural Science Foundation of China under Grant 62001357 and Grant 62171313. *(Corresponding authors: Zhen Gao; Lei Liu.)*

Zhaohui Guo and Qiang Liu are with the School of Microelectronics, Tianjin University, Tianjin 300072, China (e-mail: 2017232048@tju.edu.cn; qiangliu@tju.edu.cn).

Zhen Gao is with the School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China (e-mail: zgao@tju.edu.cn).

Lei Liu is with the Guangzhou Institute of Technology, Xidian University, Guangzhou 510555, China (e-mail: tianjiaoliulei@163.com).

Mianxiong Dong is with the Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran 050-0071, Japan (e-mail: mx.dong@csse.muroran-it.ac.jp).

Ning Zhang is with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON N9B 3P4, Canada (e-mail: ning.zhang@uwindsor.ca).

Mohammed Atiquzzaman is with the School of Computer Science, University of Oklahoma, Norman, OK 73019 USA (e-mail: atiq@ou.edu).

Digital Object Identifier 10.1109/JIOT.2023.3287210

I. Introduction

Blockchain is a distributed ledger technology underlying cryptocurrencies, such as Bitcoin and Ethereum. It can provide emerging data management of decentralization, traceability, immutability, and trustlessness with the support of the peer-to-peer network, cryptography, chained data structure, and consensus algorithms [1]. Blockchain promotes the development of data industrialization and has succeeded in many fields, such as distributed finance, smart grids, healthcare, and supply chain management [2]. Besides, blockchain has been widely used for trusted resource management to empower the construction of the digital twins, Industrial Internet of Things (IIoT), space-air-ground integrated network, and even the metaverse by combination with artificial intelligence, 6G, software-defined networking, cloud computing, etc. [3], [4], [5], [6]. In particular, in the blockchain-based Internet of Things (IoT), the devices controlled by operators with different interests interact with each other through sensory data-based transactions, where the blockchain acts as the anchor of multiparty trust to facilitate resource allocation, data sharing, and privacy-preserving [7], [8].

However, the expanding network scale increases the storage pressure on each node in the blockchain system, which is a major bottleneck for implementing blockchain-based IoT networks with resource-limited devices. Most existing blockchains are based on a stateful design, where the storage on each node includes the transaction records (ledger) and the global state [9]. The former records the creations and changes of data as proof for state transitions, which guarantees the traceability. The latter is a compact index of the entire ledger for efficient transaction validation [10]. Two different models are employed to extract the global state: the unspent transaction outputs (UTXOs) model for Bitcoin-like systems (such as Zcash and Xuperchain) and the account-balance model for Ethereum-like systems (such as Fabric and EOSIO) [11]. For both models, the size of the state is substantially smaller than the entire ledger but still in the order of GBs. For example, by October 2022, the sizes of UTXO set in Bitcoin and world state in Ethereum exceed 5 and 400 GB, respectively, [12]. Since the size of the state data would increase linearly with the number of nodes and transactions, *state explosion* becomes inevitable with such a *stateful* model [13]. Recent researches suggest that the account-balance model is inferior to the UTXO model in terms of storage and validation efficiency [14]. The state data organized as the account-balance model could be reformulated as the UTXO model by the method proposed in [14]. In the blockchain-based IoT network, ledger data can be dumped off-chain, e.g., interplanetary file system [15], but the global state is necessary for the transaction process. Thus, the size of state data on each node becomes one of the main bottlenecks for the large-scale applications of blockchain in IoT scenarios with resource-limited devices.

Various solutions have been proposed to address the *state explosion* issue in the blockchain, including schemes of data transfer, distributed storage, and stateless transaction validation. In the Slimchain proposed in [16], the entire ledger and global state are stored on off-chain nodes with ample storage resources, which validate and execute the transactions in parallel and return results with proofs to other nodes. But this scheme weakens the decentralization of the system dramatically. In the SSChain proposed in [17], all nodes are divided into multiple shards, where each node only maintains the state subset of the shard that the node belongs to. However, a complicated architecture is necessary for cross-shard communication. The scheme in [18] was proposed to compress the account balances based on the residue number system, but the storage size still increases linearly with the number of accounts. Besides, the BZIP was proposed in [19] to decrease the storage burden on each node by removing the redundances among database entities. However, the improvement in practice is limited because the underlying database is usually optimized before being applied to the blockchain.

As an emerging solution, the stateless design was proposed to improve storage efficiency by restructuring the transaction process. Based on the cryptographic accumulator, the global state is replaced by the commitment maintained in the latest block header. The transaction validity is determined by checking the attached witness with the latest commitment. In this design, no state data is needed for transaction validation, and only the node-related state data is needed for transaction issuance (TI), which improves both storage efficiency and privacy-preserving relative to the stateful schemes. The performance of stateless design depends on the accumulators used. The accumulator is a cryptographic primitive that provides short membership proof (witness) for any candidates in a certain set, which could be verified against a short commitment to all accumulated elements. The accumulator could be divided into two categories: the dynamic ones support the addition/deletion of elements at a cost independent of the number of accumulated elements, while the static ones do not support the set updates [20]. The accumulators are often used as the communication-efficient authenticated data structure for membership checks. The commonly used dynamic accumulators are those based on the Merkle tree and RSA algorithm. A stateless scheme for the UTXO-like blockchain was proposed in [21] based on the RSA accumulators, where each UTXO is assigned a fixed-length witness so that the communication overhead introduced by the attached witness is constant. However, the high complexity of the witness update makes it difficult to implement in practice. Chepurnoy et al. [22] proposed a stateless scheme named Edrax for the UTXO-like blockchain with the sparse Merkle tree (SMT) as an accumulator, where the root and branch are taken as the commitment and witness. Compared with the RSA-based scheme, the witness update in Edrax is extremely efficient, but the communication overhead increases logarithmically with the size of the UTXO set, which is unacceptable for a large-scale system with massive nodes and transactions. In summary, to make the stateless design more practical, the cryptographic primitive underlying the accumulator should meet the requirements in succinctness and efficiency, which means a smaller proof size and a faster time for proof generation and validation.

This article proposes the Verkle accumulator-based stateless transaction validation (VA-STV) scheme for the permissionless blockchains to decrease the storage burden of state data on each node with the acceptable overhead of communication and computation. The VA-STV is based on the Verkle accumulator (VA), which is a combination of the Verkle tree (VT) and the KZG polynomial commitment scheme (PC). The proposed VA-STV performs better in succinctness, efficiency, and adaptability. The major contributions are summarized below.

1) We construct a new dynamic accumulator VA by combining the VT and the KZG. The VT helps to overcome the limitations of the KZG in the commitment generation and update. The KZG helps to realize the hierarchical membership checks for the leaf nodes in the VT. Thus, the VA provides a smaller proof size than the SMT-based accumulator and more efficient commitment/witness generation and update than the RSA-based accumulator.
2) We describe the design details of the VA-STV for the Bitcoin-like blockchain, including the structure of the block header and transaction validation procedure. The VA-STV realizes a better tradeoff among storage, communication, and computation overhead than the existing stateless schemes. The applicability of the VA-STV in the Ethereum-like blockchains is also discussed.
3) We implement the proposed scheme on a simulation platform and compare its performance with existing stateless schemes in storage, computation, and communication. The results show that, with the same settings, the witness size of the VA-STV is 60%–74% of that of Edrax, and the commitment generation is $6\times$–$14\times$ faster than that in Boneh, which indicates the advantages of VA-STV in succinctness and efficiency. Besides, a tradeoff between the communication and computation requirements can be achieved by adjusting the branching factor, which improves the adaptability of the proposed scheme for different IoT scenarios.

The remainder of this article is organized as follows. Section II briefly introduces the techniques related to the VA-STV scheme. Section III provides details of the VA-STV

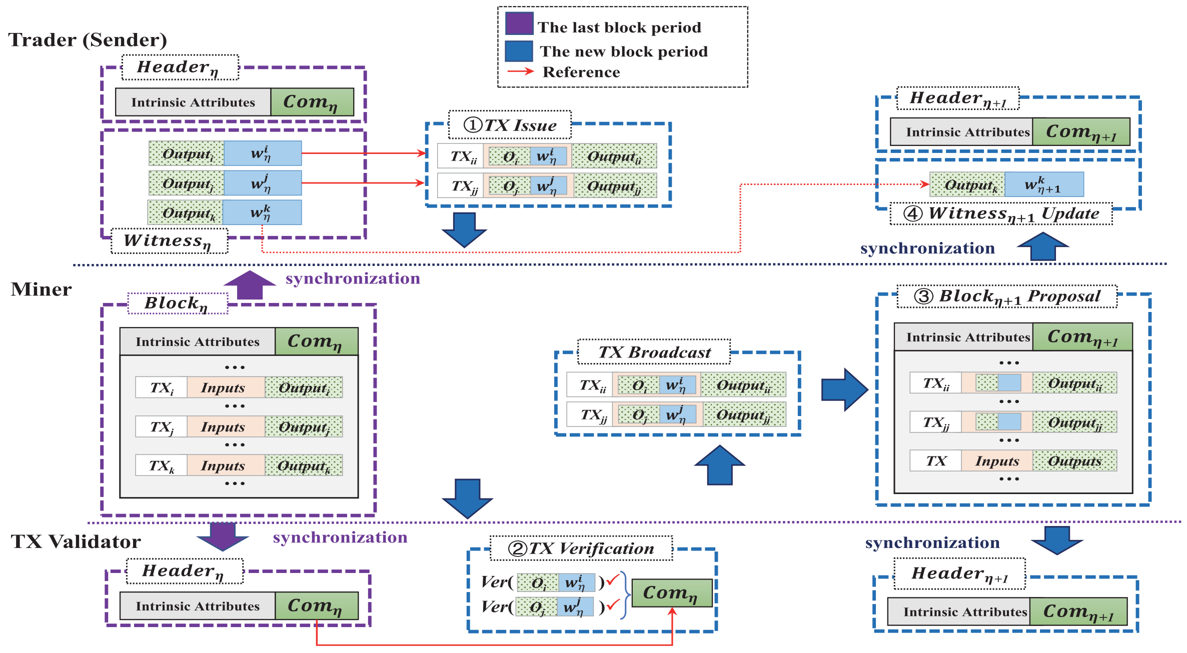


Fig. 1. Typical transaction procedure in the stateless blockchain.

scheme. The performance of the VA-STV scheme is analyzed and evaluated in Section IV and Section V, respectively. Finally, conclusions are given in Section VI.

II. Basics of Related Techniques

This section briefly introduces the transaction procedures in the stateful and the stateless designs in Section II-A. Then the VT is introduced in Section II-B. The cryptographic primitives supporting membership proof are compared to clarify the reason for selecting the KZG for stateless blockchain design in Section II-C. Finally, the functions and security assumptions of the KZG are introduced in Section II-D.

A. Transaction Procedures in the Stateful and Stateless Designs

In the Bitcoin-like blockchain, the global state is maintained as the UTXO set, which is a compact index of each UTXO with its ownership. Typically, a UTXO could be described uniquely by a tuple (*txid*, *pk*, *v*, *index*), where *txid* is the hash of the transaction creating this UTXO, *pk* is the public key or address, *v* is the number of tokens locked to the *pk*, and *index* is its position in the output set [23]. The integrity of the UTXO set is guaranteed by the immutable ledger, and *txid* is used to avoid forging UTXO from nonexistent transaction. A UTXO is taken as valid when it exists and is unspent. A transaction takes some existing UTXOs as inputs and creates new UTXOs locked to the receiver addresses as outputs. The transaction validity depends on the validity of the inputs. A typical transaction usually consists of multiple inputs and outputs. Correspondingly, the UTXO set is updated by removing the spent transaction outputs and adding the new UTXOs. The transaction procedure in the stateful blockchain is as follows [24].

1) The node issues a new transaction with some related UTXOs as inputs and broadcasts it to the network.
2) The nodes receiving the new transaction first validate the inputs against the UTXO set maintained locally. If the total sum of currencies locked to the inputs exceeds the transferred amount, the transaction is determined as valid and cached into the local transaction pool.
3) Each node packs some transactions from the local pool into a new block and includes the parent hash, timestamp, Merkle root, and other items in the header. All block proposals are broadcasted independently to the network.
4) Each node needs to validate the received block proposals and select one valid block as the new block according to the distributed consensus algorithm (such as Proof-of-Work in Bitcoin). Then the UTXO set is updated based on the transactions packed in the new block.

Compared with the stateful design, the state data is not essential for the nodes in the stateless design, where the transaction validity is guaranteed by committing to the current state and representing the state transition as the updates of commitment and the witnesses. The typical transaction procedure in the stateless design is shown in Fig. 1. We assume the ηth block period (represented by the purple dash line) has finished, and nodes acting as various roles are participating in the activities of the (η+1)th period (represented by the blue dash line). The state set corresponding to the Block$_\eta$ is committed to Com$_\eta$ (green parts) and added to the Header$_\eta$. The corresponding witness for the ith UTXO is represented as w_η^i (blue parts), which will be maintained locally by the node controlling the UTXO when synchronizing the Block$_\eta$. It should be noted that each node should maintain a lightweight chain formed by headers to access the recognized commitment. The commitment is involved in the consensus process to guarantee its integrity. There are four differences from the stateful design: 1) to issue a new transaction, a node does not only take the related UTXOs as inputs but also provides the witnesses for their validity; 2) the nodes receiving the transaction validate its inputs by checking whether the attached witnesses match the commitment in the latest header; 3) the nodes update the commitment according to the packed transactions before competing for block proposal; and 4) other nodes update the witnesses related to their UTXOs independently to match the updated commitment. As shown in Fig. 1, the witnesses w_η^i and w_η^j are essential to issue and validate the new transactions TX_{ii} and TX_{jj}. The commitment is updated from Com_η to $\mathrm{Com}_{\eta+1}$ by miners. $\mathrm{Com}_{\eta+1}$ is added to the $\mathrm{Header}_{\eta+1}$ and involved in the consensus process of the $(\eta+1)$th block period. Finally, the witnesses for related UTXOs are updated from $Witness_\eta$ to $Witness_{\eta+1}$ when synchronizing $\mathrm{Block}_{\eta+1}$.

It should be noted that the nodes in the blockchain have different functions, including TI/validation, routing, and generation for new blocks [1]. For clarity, a node in this article is named as a trader, validator, or miner based on their functions. The traders send and receive tokens through transactions. The validators perform transaction validation and routing. The miners are responsible for proposing new blocks. Each node could have multiple roles simultaneously.

B. Verkle Tree

The VT is the α-ary tree, where α is the branching factor ($\alpha>2$). For an ordered sequence of n elements $Y = \{y_1, y_2, \ldots, y_n\}$, the VT with N leaf nodes is full if $n = N$, and is sparse if $n < N$. A subtree in the VT consists of one parent node and α children. All nodes of the tree are organized as D layers ($D = \log_\alpha N$) from the bottom to the top (except the root), and each leaf node L_i has a unique path (Path_i) to the root. The L_i is grouped into the $t_1(i)$th subtree in the first layer, where $t_1(i) = \lceil i/\alpha \rceil$. The lth subtree on the Path_i could be represented as $ST^l_{t_l(i)}$, where l represents the layer where its children are and $t_l(i) = \lceil i/\alpha^l \rceil$ is its position in the subtrees of the lth layer. The lth parent node of L_i along the Path_i is represented as $P^l_{[t_l(i), r_l(i)]}$ ($l > 1$), where $r_l(i)$ is the relative position of the $P^l_{[t_l(i), r_l(i)]}$ in the subtree $ST^l_{t_l(i)}$ and could be obtained as

$$r_l(i) = t_{l-1}(i) + \left(1 - \left\lceil \frac{t_{l-1}(i)}{\alpha} \right\rceil\right) \cdot \alpha \quad (l > 1). \tag{1}$$

In particular, $r_l(i)$ for the first layer could be obtained as

$$r_1(i) = i + \left(1 - \left\lceil \frac{i}{\alpha} \right\rceil\right) \cdot \alpha \quad (l = 1). \tag{2}$$

Except for the root, all nodes in the VT are tuples of (*value*, *proof*). For a leaf node L_i, the tuple is denoted as $(h_i, \pi^1_{[t_1(i), r_1(i)]})$, where h_i is the hash of y_i. For the parent node of L_i, the tuple is denoted as $(\mathrm{com}^l_{[t_l(i), r_l(i)]}, \pi^l_{[t_l(i), r_l(i)]})$, where $\mathrm{com}^l_{[t_l(i), r_l(i)]}$ is the vector commitment of α children [25]. For the root of the VT, the *value* is the final commitment to all nodes, and no *proof* is needed. The membership of a child in a subtree is guaranteed by the unique proof related to the commitment contained in the parent node.

Based on this structure, VT can be used as an accumulator. For L_i, its membership in a VT could be determined by recursively inspecting the memberships of its parent nodes $P^l_{[t_l(i), r_l(i)]}$ in the subtrees $ST^l_{t_l(i)}$ along the Path_i, which could be done by checking whether the *proof* in $P^l_{[t_l(i), r_l(i)]}$ match with the *value* of their parents $P^{l+1}_{[t_{l+1}(i), r_{l+1}(i)]}$. The witness of the L_i is formed by all commitments and proofs in its $\log_{\alpha N}$ parents (except the root) on the Path_i.

C. Comparison of Different Cryptographic Primitives Supporting Membership Proof

Various cryptographic primitives could support the construction of the accumulator by providing commitments and corresponding proofs, such as the PC and zero-knowledge proof system (ZKP). A PC is the commitment to a certain polynomial F, with which the committer (prover) could convince the verifier that the evaluation of F at a certain point x satisfies $F(x) = y$ through a proof without revealing any coefficients of F. The PC has properties of position binding and hiding to avoid proof forgery and information breach [26]. The ZKPs are constructed based on PCs with combinations of other primitives, such as knowledge of exponent (KoE) for consistency checks. The ZKP has properties of completeness, soundness, and zero-knowledgeness [27]. Currently, the commonly used PCs include KZG (also called Kate), inner product argument (IPA), and fast Reed–Solomon interactive oracle proofs of proximity (FRI) schemes [28], [29], [30]. The commonly used ZKPs include the zero-knowledge succinct noninteractive argument of knowledge (zk-SNARK) [31], the scalable transparent argument of knowledge (STARK) [32], Sonic [33], etc., most of which are based on the Rank-1 constraint system (R1CS).

The ZKPs can provide more advanced functions than PCs as follows: 1) the ZKP's target is a complicated relation representing the continuous computation process, while the PC's target is a certain polynomial; 2) the ZKPs achieve the zero-knowledgeness, which has higher security and privacy protection; and 3) some ZKPs (such as Sonic) could generate updatable common reference string (CRS) for different relations [33]. However, in the stateless blockchain, these advanced functions are not essential, but succinctness (proof size as small as possible) and efficiency (commitment/proof generation as fast as possible) are required. From the perspective of succinctness and efficiency, the ZKPs are inferior to PCs due to complicated operations before the proof generation, such as translating the relation into the circuit of bounded size and constructing the R1CS. Thus, compared with the ZKPs, the PCs are more suitable for the stateless blockchain design.

Among various PCs, e.g., FRI and IPA, the KZG provides better succinctness and efficiency due to smaller proof size and less time for proof generation and validation, but has lower security due to the trusted setup. However, the trusted setup could be realized by the secure multiple-party computation (MPC) instead of relying on the third party, which has been proven secure enough in many popular blockchain projects, such as Zcash [34]. For all PCs and ZKPs, the differences are listed in Table I. Based on the comprehensive assessments of

TABLE I
COMPARISON BETWEEN VARIOUS PCs AND ZKPs

	Scheme	UT	TS	Proof size	Proof time	Verify time
PC	FRI[30]	Hash	×	$O(\log^2(n))$	$O(n\log^2(n))$	$O(\log^2(n))$
	KZG[28]	EC+ Pair	✓	$O(1)$	$O(n)$	$O(1)$
	IPA[29]	EC	×	$O(\log(n))$	$O(n)$	$O(\log(n))$
ZKP	SNARK[31]	R1CS +PCs	✓	$O(1)$	$O(\log(d))$	$O(1)$
	Sonic[33]		✓✗		$O(d\log d)$	
	STARK[32]		×	$O(\log(d))$		$O(\log(d))$

UT and TS are short for the underlying techniques and the trusted setup.
EC and Pair are short for the elliptic curve and pairing.
n and d are the degree of a certain polynomial and the depth of a circuit.
The symbol ✓✗ indicates that Sonic needs a TS, but its CRS is updatable.

these schemes, the KZG is the best choice for constructing the accumulator in the stateless blockchain due to its advantages in succinctness and efficiency.

D. Functions and Security Assumptions of the KZG

For the ordered sequence of n values $Y=\{y_1, y_2, \ldots, y_n\}$, the relationship between each value and its position in the sequence could be described as the coordinate point (x_i, y_i), which reformulates the Y as a point set $\hat{Y}=\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. Since a unique polynomial $F(x)$ with the degree of n-1 could be determined based on n specific points, the PC to $F(x)$ could act as the vector commitment to the Y by evaluating whether $F(x = x_i)$ equals to y_i. The KZG is an efficient implementation of the PC based on the bilinear map between elliptic curves.

The bilinear map (also called *Pairing*) is defined as $e : EC_1(F_p) \times EC_2(F_p^k) \to EC_T(F_p^k)$, where EC_1, EC_2, and EC_T are the elliptic curves with the generators of G, I, and T, respectively. F_p is the finite field over the prime p, F_p^k is the extension field of F_p with the embedded degree of k, and $\times$ maps two points from the cyclic groups $EC_1(F_p)$ and $EC_2(F_p^k)$ to the point of the cyclic group $EC_T(F_p^k)$ [35]. For all $P{\in}EC_1$ $(P{=}G^a)$ and $Q{\in}EC_2$ $(Q = I^b)$ with $a, b \in F_p$, there are

$$e(P, Q) = e\left(G^a, I^b\right) = e(G, I)^{ab} = T^{ab} \tag{3}$$

where $P = G^a$ and $Q = I^b$ represent the mapping from the elements of the finite field to the elements of the elliptic cycle group. This process is commonly called *encoding* [28].

Denoting G^x, I^x, and T^x as $[x]_1$, $[x]_2$, and $[x]_T$, respectively, for any $x \in F_p$, (3) can be written as

$$e(P, Q) = e([a]_1, [b]_2) = [ab]_T. \tag{4}$$

Based on the bilinear map, the KZG polynomial commitment has the property of additive and semi-multiplicative homomorphism. For a point set $\hat{Y}$, the polynomial $F(x)$ with the degree of $n{-}1$ passing through all n coordinates could be obtained based on the Lagrange interpolation. Each point (x_i, y_i) could be represented by a polynomial as

$$F_i(x) = \prod_{j\neq i} y_i \cdot \frac{x - x_j}{x_i - x_j} \tag{5}$$

satisfying $F_i(x = x_i) = y_i$ and $F_i(x \neq x_i) = 0$. Finally

$$F(x) = \sum_{i=1}^{n} F_i(x) = \sum_{t=0}^{n-1} f_t \cdot x^t \tag{6}$$

where f_t is the coefficient. In practice, inverse fast Fourier transforms (IFFTs) can be used to get polynomials quickly [36].

Based on the $F(x)$, the basic functions of the KZG commitment could be described as follows:

Function 1: com $\leftarrow$ GenCom$(F(x))$, generating the commitment to the $F(x)$. The com is obtained by evaluating $F(s)$ and mapping the result to the EC_1, where s is the secret position generated by the trusted setup. Based on the additive homomorphism, there is

$$\text{com} = G^{F(s)} = \sum_{t=0}^{n-1} f_t \cdot G^{s^t} = \sum_{t=0}^{n-1} f_t \cdot \left[s^t\right]_1. \tag{7}$$

The computation complexity of commitment generation is $O(n)$ [28].

Function 2: $\pi_i \leftarrow$ GenPrf$(x_i, y_i, F(x))$, generating the proof for the (x_i, y_i). For any (x_i, y_i) on $F(x)$, there exists a polynomial $J(x)$ satisfying $J(x) = F(x){-}y_i = 0$ for $x = x_i$, i.e., $x = x_i$ is the zero point of $J(x)$. Based on the properties of the polynomial, there exists a polynomial $Z(x) = (x - x_i)$ that could divide the $J(x)$ exactly [28]. Thus, as long as (x_i, y_i) is in $F(x)$, there exists a quotient polynomial $Q(x) = \sum_{t=0}^{n-1} q_t \cdot x^t$, satisfying $Q(x) \cdot Z(x) = F(x) - y_i$. Then, the proof could be obtained by evaluating $Q(s)$ and mapping the result to the EC_1

$$\pi_i = G^{Q(s)} = \sum_{t=0}^{n-1} q_t \cdot G^{s^t} = \sum_{t=0}^{n-1} q_t \cdot \left[s^t\right]_1. \tag{8}$$

The computation complexity of proof generation is $O(n)$ [28].

Function 3: $\{0, 1\} \leftarrow$ Ver$(x_i, y_i, \pi_i, \text{com})$, verifying if (x_i, y_i) is in $F(x)$. The membership of (x_i, y_i) could be determined by checking whether $Q(x) \cdot Z(x) = F(x) - y_i$ holds. In KZG commitment, the equality check is achieved based on semi-multiplicative homomorphism introduced by the bilinear mapping [26]. For the matched π_i and com, there is

$$e(\pi_i, [s - x_i]_2) = e\left(\text{com} - \left[y_i\right]_1, I\right). \tag{9}$$

The computation complexity for verification is $O(1)$ [28].

Function 4: com$'$ $\leftarrow$ UpdateCom$(\text{com}, x_i, \delta_i)$, updating the commitment to $\hat{Y}'$ with (x_i, y_i) changed to (x_i, y_i'). Based on the properties of the polynomial, for the updated $\hat{Y}'$, there is a new polynomial $F'(x) = F(x) + \Delta_i(x)$, where

$$\Delta_i(x) = \prod_{j\neq i} \delta_i \cdot \frac{x - x_j}{x_i - x_j} = \sum_{t=0}^{n-1} d_t \cdot x^t \tag{10}$$

satisfying $\Delta_i(x = x_i) = \delta_i$ and $\Delta_i(x \neq x_i) = 0$. Based on the additive homomorphism, com is updated as

$$\text{com}' = [F(s) + \Delta_i(s)]_1 = \text{com} + \sum_{t=0}^{n-1} d_t \cdot \left[s^t\right]_1. \tag{11}$$

The main computation of commitment update is to calculate $\Delta_i(x)$. So the computation complexity of the commitment update is $O(n)$ [28].

Function 5: $\pi'_i \leftarrow \text{UpdatePrf}(x_j, \delta_j, \pi_i)$, updating the proof for (x_i, y_i) to match the updated com$'$ when y_j changes to y'_j. For a point of (x_i, y_i) with $i \neq j$, the proof is updated to $[Q'(s)]_1$, where $Q'(x)$ is the updated quotient polynomial satisfying $Q'(x) \cdot Z(x) = F'(x) - y_i$. Since $x = x_i$ is the zero point of the $\Delta_j(x)$ ($\Delta_j(x_i) = 0$), there exists a quotient polynomial $Q^j_\Delta(x) = [(\Delta_j(x))/(x - x_i)] = \sum_{t=0}^{n-1} q'_t \cdot x^t$. Thus, there is $[(F'(x) - y_i)/(Z(x))] = [(F(x) - y_i)/(x - x_i)] + [(\Delta_j(x))/(x - x_i)] = Q(x) + Q^j_\Delta(x)$. Based on the additive homomorphism, the proof could be updated as

$$\pi'_i = \left[Q(s) + Q^j_\Delta(s)\right]_1 = \pi_i + \sum_{t=0}^{n-1} q'_t \cdot [s^t]_1. \tag{12}$$

When $i = j$, the updated quotient polynomial is $Q'(x) = [(F'(x) - y'_i)/(x - x_i)] = [(F(x) - y_i)/(x - x_i)] + [(\Delta_i(x) - \delta_i)/(x - x_i)]$. Since $x = x_i$ is still the zero point of the $\Delta_i(x) - \delta_i$, there still exists a polynomial $Q^i_\Delta(x)$ satisfying $Q^i_\Delta \cdot Z(x) = \Delta_i(x) - \delta_i$. The proof for (x_i, y'_i) could still be updated as: $\pi'_i = \pi_i + [Q^i_\Delta(s)]_1$. Similar to that for Function 4, the computation complexity for the proof update is also $O(n)$ [28].

From the above process, we can see that the secret position s is essential for constructing the KZG commitment, so s should be inaccessible to anyone to prevent adversaries from falsifying proof [37]. Thus, sets of $([s^1]_1, [s^2]_1, \ldots, [s^n]_1)$ and $([s^1]_2, [s^2]_2, \ldots, [s^n]_2)$ should be precomputed as a part of the public parameters (pp) through the process called trusted setup, which will delete s completely after generating all pp. Typically, the process could be achieved by the secure Multiparty computation and described formally as:

Function 6: $pp \leftarrow \text{Setup}(1^\lambda, n)$, generating the public parameters [also called CRS] by the trusted setup, where λ is the security parameter. The computation complexity for the trusted setup is $O(n)$.

Function 1∼6 have unified security assumptions. The KZG is a noninteractive implementation of the linear probabilistically checkable proofs of proximity (LPCP), where LPCP requires a completely random and unpredictable secret s as the sampling challenge for the prover, and noninteractivity requires the challenge could be reusable by all provers and verifiers [38]. To meet these requirements, the KZG uses the trusted setup to generate the homomorphism encodings for s and its exponents in the pairing-friendly elliptic curves as CRS (pp), which supports arithmetic operations between the group elements with s inaccessible. Thus, the security of the KZG is based on the discrete logarithm (DL), q-strong Diffie–Hellman exponent (q-DHE), and q-strong bilinear Diffie–Hellman (q-SBDH) assumptions [39]. For a probabilistic polynomial time (PPT) adversary *Adv*: 1) the DL guarantees that *Adv* cannot recover s from $[s^i]_1$ or $[s^i]_2$; 2) the q-DHE guarantees that *Adv* only uses the group elements of CRS, instead of computing the new elements based on them; and 3) the q-SBDH guarantees that *Adv* has no efficient method to break down the pairing. Thus, the KZG has enough security for the stateless design.

III. VA-Based Stateless Transaction Validation Scheme

This section first introduces the motivation and basic idea for constructing the VA by combining the VT and the KZG in Section III-A. Then, the VA is constructed based on the KZG and the VT in Section III-B, with a simple example to demonstrate the principles of VA in Section III-C. The VA-STV is introduced in Section III-D. Finally, the applicability of the VA-STV in Ethereum-like blockchains is discussed in Section III-E.

A. Motivation and Basic Idea

As described in Section II-D, the KZG relies on the trusted setup for the CRS (pp), which is then used for the generation and update of the commitment on the elliptic curves. Intuitively, we could represent the entire state set with millions of tokens as a high-degree polynomial and apply the KZG directly. However, the high degree results in two issues: 1) the processes for trusted setup, polynomial interpolation (PI), and commitment generation will be extremely time consuming and 2) the large size of CRS makes the commitment update slow due to the inefficiency of data access.

The VT can effectively address the above issues. Benefiting from the multibranch structure, a VT with millions of leaf nodes could be decomposed into multiple subtrees with α children ($\alpha << N$), and each subtree could be represented by a low-degree polynomial. This divide-and-conquer method brings the following benefits: 1) the trusted setup is only performed once, and the process takes a short time due to small α; 2) CRS has limited size and can be reused for all subtrees; 3) the commitment generation will be more efficient because the subtrees in the same layer could be processed in parallel; 4) when leaf nodes change, only the relevant subtrees need to be modified with the others remain the same, which improves the efficiency for commitment update; and 5) as the sparse tree, the VT allows adding and removing leaf nodes without extra setup, which improves the KZG's scalability.

Intuitively, the VT could be constructed based on the hash functions. In this case, due to the multibranch structure, the membership proof of each child node in a subtree is formed by hash values of other $\alpha - 1$ siblings, which would lead to a much larger witness relative to that for the Merkle tree [25]. However, with the support of the KZG, the membership of each leaf node in the VT could be proved following the unique *Path* without the need for the $\alpha - 1$ siblings in the same layer, which can effectively reduce the size of the witness. Thus, combining the VT and the KZG could achieve a good tradeoff between the computation and communication cost, which can meet the requirements of stateless design for efficiency and succinctness.

B. Construction of the Verkle Accumulator

With the support of the KZG, in any subtree of a α-ary VT, a child could be reformulated as a coordinate of (pos, *value*), where *pos* represents its relative position in the subtree. Then the *value* in the parent node of this subtree is the commitment to the polynomial $F(x)$ with the degree of $\alpha-1$, which is obtained based on the points from its α children. Correspondingly, for each child in the subtree, a proof is generated to prove that (*pos*, *value*) is on $F(x)$. Once the *value* in a child changes, the commitment in the parent and the proofs for all children should be updated synchronously. For the non-leaf node, its *value* should be hashed as an integer to construct the polynomial since the KZG commitment is a point in the elliptic curve. As the final commitment of the VA, the VT root is updated by summarizing the commitments of all subtrees layer-by-layer upward. During this process, the witness for L_i could be updated synchronously.

For an ordered sequence of n elements $Y = \{y_1, y_2, \ldots, y_n\}$, the basic functions of the VA based on a VT with N leaf nodes are described formally as follows:

Function 7: $\hat{Y} \leftarrow \text{Init}(Y)$, initializing the leaf nodes as $\hat{Y} = \{(1, h_1), (2, h_2), \ldots, (n, h_n), \ldots\}$, where h_i is the hash of y_i. The $\hat{Y}$ is a set of N points representing N leaves in the VT. For sparse VT ($n < N$), the hash values of empty leaf nodes $L_{n+1} \sim L_N$ are set to zeros, so we have $\hat{Y} = \{(1, h_1), (2, h_2), \ldots, (n, h_n), (n+1, 0), \ldots, (N, 0)\}$

Function 8: $Com \leftarrow \text{GenVTCom}(\hat{Y})$, generating the final commitment *Com* to Y. In the α-ary VT, a leaf node L_i is grouped into the $t_1(i)$th subtree $ST^1_{t_1(i)}$ in the first layer with the $\alpha-1$ siblings. All children in the $ST^1_{t_1(i)}$ are reformulated as a point set based on their relative position as $\hat{Y}_\tau = \{(1, h_{(\tau-1)\cdot\alpha+1}), (2, h_{(\tau-2)\cdot\alpha+2}), \ldots, (r_1(i), h_i), \ldots, (\alpha, h_{(\tau-1)\cdot\alpha+\alpha})\}$, where $\tau = t_1(i)$, $h_i = h_{(\tau-1)\cdot\alpha+r_1(i)}$, and $r_1(i)$ is obtained based on (2). With $\hat{Y_\tau}$, the commitment to the ST^1_τ can be obtained based on Function 1 as $\text{com}^2_{[t_2(i),r_2(i)]} \leftarrow \text{GenCom}(F^1_\tau(x))$, where $F^1_\tau(x)$ is the unique polynomial obtained by (6), $t_2(i) = \lceil i/\alpha^2 \rceil$, and $r_2(i)$ is obtained by (1). The $\text{com}^2_{[t_2(i),r_2(i)]}$ is the *value* in the $P^2_{[t_2(i),r_2(i)]}$, which is L_i's parent node in the ST^1_τ along the Path_i. For other subtrees in the first layer, their parent nodes could obtain the *values* in the same way, and they are grouped into the subtrees of the Second layer. For L_i, its parent in ST^1_τ could be reformulated as $(r_2(i), H^2_{[t_2(i),r_2(i)]})$, where $r_2(i)$ is the relative position in the upper subtree $ST^2_{t_2(i)}$ and $H^2_{[t_2(i),r_2(i)]}$ is the hash of the $\text{com}^2_{[t_2(i),r_2(i)]}$. Thus, all commitments to subtrees of D layers along the Path_i could be obtained by performing Function 1 upward recursively. The root of VT is the final commitment *Com* to Y. In particular, for the subtree with α empty children, the *value* in the parent node is the zero point in EC_1 and its hash is set to zero.

Function 9: $\text{wit}_i \leftarrow \text{GenWit}(i, h_i, Y)$, generating the witness wit_i for the L_i. For the subtree ST^1_τ of the first layer, during the process of generating $\text{com}^2_{[t_2(i),r_2(i)]}$, the proof for L_i can be obtained by Function 2 as $\pi^1_{[\tau,r_1(i)]} \leftarrow GenPrf(r_1(i), h_i, F^1_\tau(x))$. Similarly, for L_i's parent node $P^2_{[t_2(i),r_2(i)]}$, its proof corresponding to the commitment of the upper subtree $ST^2_{t_2(i)}$ is $\pi^2_{[t_2(i),r_2(i)]} \leftarrow \text{GenPrf}(r_2(i), H^2_{[t_2(i),r_2(i)]}, F^2_{t_2(i)}(x))$. Thus, for all parents $P^l_{[t_l(i),r_l(i)]}$ of L_i along the Path_i, their proofs in the upper subtrees $ST^l_{t_l(i)}$ could be obtained by performing Function 2 upward recursively. Finally, the witness wit_i for the L_i is a collection of all commitments and proofs along the Path_i as

$$\text{wit}_i = \left\{\pi^1_{[t_1,r_1]}\right\} \cup \left\{\left(\text{com}^l_{[t_l,r_l]}, \pi^l_{[t_l,r_l]}\right) | l = 2, 3, \ldots, D\right\} \quad (13)$$

where t_l and r_l are the abbreviations for $t_l(i)$ and $r_l(i)$, respectively, and $(\text{com}^l_{[t_l,r_l]}, \pi^l_{[t_l,r_l]})$ is the tuple of L_i's parent node of the lth layer.

Function 10: $\{0, 1\} \leftarrow \text{VerWit}(i, h_i, \text{wit}_i, Com)$, checking the membership of the L_i in the VT. In the subtree $ST^1_{t_1(i)}$, the membership of L_i could be determined by checking whether L_i's proof $\pi^1_{[t_1,r_1]}$ matches the commitment $\text{com}^2_{[t_2,r_2]}$ in its parent node, which could be done by Function 3 as $\{0, 1\} \leftarrow \text{Ver}(r_1, h_i, \pi^1_{[t_1,r_1]}, \text{com}^2_{[t_2,r_2]})$. For L_i, along the Path_i, the relationships between the parent nodes of the lth and $(l+1)$th $(2 \le l \le D)$ layers could be determined in the same way. If all checking is passed, the membership of (i, y_i) in Y could be confirmed.

Function 11: $\ddot{\text{Com}} \leftarrow \text{UpdateVTCom}(\text{Com}, i, \delta_i, \text{wit}_i)$, updating *Com* to $\ddot{Com}$ when y_i changes to y'_i, where $\delta_i = h'_i - h_i$. The change in the L_i results in the update of the commitment to the subtree $ST^1_{t_1(i)}$ in the first layer. As described in Function 4, the commitment $\text{com}^2_{[t_2,r_2]}$ could be updated as $\ddot{\text{com}}^2_{[t_2,r_2]} \leftarrow \text{UpdateCom}(\text{com}^2_{[t_2,r_2]}, r_1, \delta_i)$. Thus, the *value* of the $P^2_{[t_2,r_2]}$ changes to $\ddot{\text{com}}^2_{[t_2,r_2]}$ from $\text{com}^2_{[t_2,r_2]}$, which results in the update of the commitment in its parent in the subtree $ST^2_{t_2(i)}$ along the Path_i. Finally, the change in L_i will pass to all its parent nodes along the Path_i until the final commitment in the root is updated to $\ddot{Com}$ from *Com*.

Function 12: $\ddot{\text{wit}}_i \leftarrow \text{UpdateWit}(i, j, \delta_j, \text{wit}_i, \text{wit}_j)$, updating the witness of the L_i to $\ddot{\text{wit}}_i$ when y_j changes to y'_j. There are two cases of $t_1(i) = t_1(j)$ and $t_1(i) \neq t_1(j)$. For the case of $t_1(i) = t_1(j)$, L_i and L_j are in the same subtree in the first layer, then the witness of the L_i could be updated by updating *proofs* in its parent nodes along the Path_i based on Function 5, that is

$$\ddot{\text{wit}}_i = \left\{\ddot{\pi}^1_{[t_1,r_1]}\right\} \cup \left\{\left(\ddot{\text{com}}^l_{[t_l,r_l]}, \ddot{\pi}^l_{[t_l,r_l]}\right) | l = 2, 3, \ldots, D\right\}. \quad (14)$$

For the case of $t_1(i) \neq t_1(j)$, L_i and L_j are grouped into different subtrees, but Path_i and Path_j have common nodes in high layers. Assuming that the first common node is the parent node of the subtree of the zth layer, the commitments and proofs in all parent nodes of L_j along Path_j could be updated based on wit_j as described in Function 11. For Path_i, only the proof of the node in the zth layer should be updated. In the zth layer, the change of the node on the Path_j could be obtained by Function 11 as: $\delta^z_{[t_z(j),r_z(j)]} = \ddot{H}^z_{[t_z(j),r_z(j)]} - H^z_{[t_z(j),r_z(j)]}$, then the proof of the node in Path_i could be updated by Function 5 as: $\ddot{\pi}^z_{[t_z(i),r_z(i)]} \leftarrow \text{UpdatePrf}(r_z(j), \delta^z_{[t_z(j),r_z(j)]}, \pi^z_{[t_z(i),r_z(i)]})$.

Finally, the witness of the L_i could be updated as

$$\begin{aligned}\ddot{\text{wit}}_i \leftarrow & \left\{\pi^1_{[t_1,r_1]}\right\} \cup \left\{\left(\text{com}^u_{[t_u,r_u]}, \pi^u_{[t_u,r_u]}\right) | u \in [2, z-1]\right\} \\ & \cup \left\{\left(\text{com}^z_{[t_z,r_z]}, \ddot{\pi}^z_{[t_z,r_z]}\right)\right\} \\ & \cup \left\{\left(\ddot{\text{com}}^l_{[t_l,r_l]}, \ddot{\pi}^l_{[t_l,r_l]}\right) | l \in [z+1, D]\right\}. \end{aligned} \quad (15)$$

Function 13: $\{\ddot{\text{Com}}, \ddot{\text{wit}}_i\} \leftarrow \text{Del}(\text{Com}, i, j, y_j, \text{wit}_i, \text{wit}_j)$, removing elements from the accumulator. Based on the structure of VT, removing leaf nodes from the tree is not supported by VA. But we can define the virtual deletion operation in VA as marking the nodes to be removed as empty (setting the *value* in L_j to zero, and $\delta_j = -h_j$). Then the final commitment to Y and the witness for L_i could be updated by Functions 11 and 12 as

$$\begin{cases} \ddot{\mathrm{Com}} \leftarrow \mathrm{UpdateVTCom}\big(\mathrm{Com}, j, \delta_j, \mathrm{wit}_j\big) \\ \ddot{\mathrm{wit}}_i \leftarrow \mathrm{UpdateWit}\big(i, j, \delta_j, \mathrm{wit}_i, \mathrm{wit}_j\big). \end{cases} \tag{16}$$

Function 14: $\{\ddot{\mathrm{Com}}, \ddot{\mathrm{wit}}_i, \ddot{\mathrm{wit}}_n, \ddot{\mathrm{wit}}_{n+1}\} \leftarrow \mathrm{Add}(Com, i, n+1, y_{n+1}, \mathrm{wit}_i, \mathrm{wit}_n, \mathrm{LP})$, adding a new element y_{n+1} to the accumulator, where wit_n is the witness for the nth leaf corresponding to the last element y_n in Y. LP (latest proofs) is a set formed by all proofs for the empty nodes in the subtrees following the Path_n. The add operation is allowed in VA only when the VT is sparse, which is realized by assigning the y_{n+1} to the L_{n+1} (changing the *value* in L_{n+1} from zero to y_{n+1}, and $\delta_{n+1} = h_{n+1}$). For the case of $t_1(n) = t_1(n+1)$, L_n and L_{n+1} are in the same subtree. The VA could first update the final commitment and wit_n following the Path_n, then generate wit_{n+1} with the support of LP, and finally update wit_i following the Path_i. The processes could be described formally as follows:

$$\begin{cases} \ddot{\mathrm{Com}} \leftarrow \mathrm{UpdateVTCom}(Com, n+1, \delta_{n+1}, \mathrm{wit}_n) \\ \ddot{\mathrm{wit}}_n, \ddot{\mathrm{wit}}_{n+1} \leftarrow \mathrm{UpdateWit}(n, n+1, \delta_{n+1}, \mathrm{wit}_n, \mathrm{wit}_{n+1}) \\ \ddot{\mathrm{wit}}_i \leftarrow \mathrm{UpdateWit}(i, n+1, \delta_{n+1}, \mathrm{wit}_i, \mathrm{wit}_{n+1}) \end{cases} \tag{17}$$

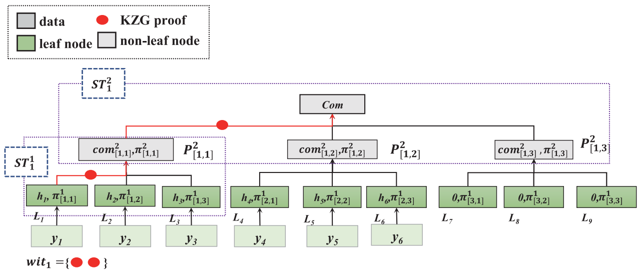
where wit_{n+1} could be obtained by replacing the $\pi^1_{[t_1(n), r_1(n)]}$ in wit_n with $\pi^1_{[t_1(n+1), r_1(n+1)]}$ in LP. Similarly, after wit_n is updated, wit_{n+1} could be updated by replacing $\ddot{\pi}^1_{[t_1(n), r_1(n)]}$ in $\ddot{wit}_n$ with $\ddot{\pi}^1_{[t_1(n+1), r_1(n+1)]}$, which could be obtained based on Function 5 as:$\ddot{\pi}^1_{[t_1(n+1), r_1(n+1)]} \leftarrow \mathrm{UpdatePrf}(r_1(n+1), \delta_{n+1}, \pi^1_{[t_1(n), r_1(n)]})$.

When $t_1(n) \neq t_1(n+1)$, L_n and L_{n+1} are the last and first children in $ST^1_{t_1(n)}$ and $ST^1_{t_1(n+1)}$. The final commitment and witnesses could be updated as (17). The only difference is $\pi^1_{[t_1(n+1), r_1(n+1)]}$ is the zero point in the elliptic curve since $ST^1_{t_1(n+1)}$ is empty before inserting y_{n+1}, which could be updated based on Function 2.

Function 15: $pp \leftarrow \mathrm{Setup}(1^\lambda, \alpha)$, generating the public parameters (CRS) by the trusted setup. Since the VA is built based on the KZG, the secret s is still essential for the security of the accumulator, which should be discarded completely after obtaining the pp. For the plain KZG, all n points in Y are reformulated as a polynomial with a degree of $n-1$. Similarly, α points are used to get a polynomial with the degree of $\alpha-1$ in the VA, so the parameter for setup is α instead of n.

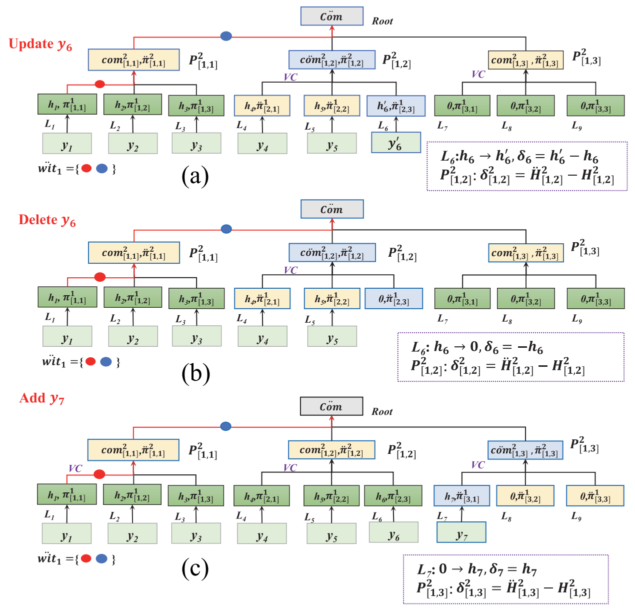
C. Demonstration of the Verkle Accumulator

An example of the 3-ary ($\alpha = 3$) VA committing to the $Y = \{y_1, y_2, \ldots, y_6\}$ ($n = 6$) is shown in Fig. 2, where the leaf and nonleaf nodes are marked by green and gray, respectively. All nodes are organized as two layers except the root ($D = 2$). The VT with nine leaf nodes ($N = 9$) is sparse ($n < N$), and $L_7 \sim L_9$ are empty. The *values* in $L_1 \sim L_6$ are $y_1 \sim y_6$ and their hashes are $h_1 \sim h_6$. $h_7 \sim h_9$ for $L_7 \sim L_9$ are zeros. As shown in Fig. 2, $L_1 \sim L_3$ are grouped



data
KZG proof
leaf node
non-leaf node
ST^2_1
ST^1_1
Com
$com^2_{[1,1]}, \pi^2_{[1,1]}$ $P^2_{[1,1]}$
$com^2_{[1,2]}, \pi^2_{[1,2]}$ $P^2_{[1,2]}$
$com^2_{[1,3]}, \pi^2_{[1,3]}$ $P^2_{[1,3]}$
$h_1, \pi^1_{[1,1]}$ $h_2, \pi^1_{[1,2]}$ $h_3, \pi^1_{[1,3]}$ $h_4, \pi^1_{[2,1]}$ $h_5, \pi^1_{[2,2]}$ $h_6, \pi^1_{[2,3]}$ $0, \pi^1_{[3,1]}$ $0, \pi^1_{[3,2]}$ $0, \pi^1_{[3,3]}$
L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8 L_9
y_1 y_2 y_3 y_4 y_5 y_6
$wit_1 = \{$● ●$\}$

Fig. 2. VT accumulator committing to the $Y = \{y_1, y_2, \ldots, y_6\}$.



Update y_6
Delete y_6
Add y_7
$\ddot{Com}$
Root
VC
$L_6: h_6 \to h'_6, \delta_6 = h'_6 - h_6$
$P^2_{[1,2]}: \delta^2_{[1,2]} = \ddot{H}^2_{[1,2]} - H^2_{[1,2]}$
$L_6: h_6 \to 0, \delta_6 = -h_6$
$P^2_{[1,2]}: \delta^2_{[1,2]} = \ddot{H}^2_{[1,2]} - H^2_{[1,2]}$
$L_7: 0 \to h_7, \delta_7 = h_7$
$P^2_{[1,3]}: \delta^2_{[1,3]} = \ddot{H}^2_{[1,3]} - H^2_{[1,3]}$
y'_6
y_7
$\ddot{wit}_1 = \{$● ●$\}$
(a)
(b)
(c)

Fig. 3. Processes of updating the final commitment and witnesses. (a) Process of updating L6 of VA. (b) Process of removing L6 from VA. (c) Process of adding L7 to VA.

into the first subtree ST^1_1 in the first layer, and their parent $P^2_{[1,1]}$ is grouped into the first subtree ST^2_1 in the second layer with siblings $P^2_{[1,2]}$, $P^2_{[1,3]}$ from the ST^1_2 and ST^1_3, where the path of L_1 (Path_1) is highlighted in red. The witness for L_1 (wit_1) is represented by two red dots, which are formed by the proof $\pi^1_{[1,1]}$ in L_1 and the tuple of $(\mathrm{com}^2_{[1,1]}, \pi^2_{[1,1]})$ in $P^2_{[1,1]}$ ($\mathrm{wit}_1 = \{\pi^1_{[1,1]}, \mathrm{com}^2_{[1,1]}, \pi^2_{[1,1]}\}$). The membership of y_1 in Y is determined with two checks: 1) whether L_1 is the first child of the subtree ST^1_1 and 2) whether $P^2_{[1,1]}$ is the first child of the ST^2_1, where the node's *value* is binding to its relative position in the subtree by the KZG commitment. These two checks could be formally described as 1) whether $e(\pi^1_{[1,1]}, [s-1]_2) = e(\mathrm{com}^2_{[1,1]} - [h_1]_1, I)$ holds and 2) whether $e(\pi^2_{[1,1]}, [s-1]_2) = e(Com - [H^2_{[1,1]}]_1, I)$ holds.

Based on the above example, the processes of updating the final commitment and witnesses when changing, deleting, and adding elements on the VA are shown in Fig. 3. During the processes, the nodes that only change their *proofs* are highlighted in yellow, whereas the nodes changing their tuples of (*value*, *proof*) are highlighted in blue. The nodes with no changes are marked green. In particular, when y_6 changes to y'_6 ($\delta_6 = h'_6 - h_6$), the tuple of (*value*, *proof*) in L_6 and its parents in Path_6 should be updated layer by layer. As the siblings of

L_6 in ST_2^1, the *values* in L_4 and L_5 remain the same, but their *proofs* should be updated to $\ddot{\pi}^1_{[2,1]}$ and $\ddot{\pi}^1_{[2,2]}$ based on (14). For L_1, its tuple of (*value*, *proof*) does not change since L_1 and L_6 belong to different subtrees. However, the *proof* of L_1's parent $P^2_{[1,1]}$ should be updated because its parent (root) in the subtree ST_1^2 is the first node shared by Path_1 and Path_6. Thus, wit_1 should be updated as $\ddot{\text{wit}}_1 = \{\pi^1_{[1,1]}, \ddot{\text{com}}^2_{[1,1]}, \ddot{\pi}^2_{[1,1]}\}$. In Fig. 3, the unchanged and updated parts of $\ddot{\text{wit}}_1$ are represented as red and blue dots, respectively. The L_6 will be removed from the VT following the same process if $h_6' = 0$ and $\delta_6 = -h_6$. Similarly, the new element y_7 could be added to the VA. During the process, the change in L_7 is $\delta_7 = h_7$, and L_1 does not need to update its tuples, but the *proof* in its parent $P^2_{[1,1]}$ should also be updated.

D. VA-STV Scheme

The VA-STV scheme is constructed in three steps.

1) *Parameters Setting:* The branching factor α and the number of the leaf nodes of the VT should be determined first according to the size of the latest UTXO set and its growth rate. Typically, N is set to 2^{32} to guarantee that the VT is sparse, and α is set to the power of 2 (such as 256 and 1024) [21]. Then the public parameters (*pp*) are obtained by the trusted setup as described in Function 15, including three elliptic curves (EC_1, EC_2, and EC_T) satisfying the pairing, and two sets of points $([s^1]_1, [s^2]_1, \ldots, [s^\alpha]_1)$ and $([s^1]_2, [s^2]_2, \ldots, [s^\alpha]_2)$.
2) *Accumulator Initialization:* Assuming that Block_η is the highest block, and the latest state set S_η includes n UTXOs. Each UTXO could be uniquely identified by a tuple of (*txid*, *pk*, *v*, *index*) as described in Section II. Thus, S_η could be reformulated as a point set of $\hat{Y} = \{(i, h_i) | i = 1, 2, \ldots, n)\}$ by arranging all UTXOs in ascending order of their *txid*, where i is the relative position of UTXO_i in the sorted set and $h_i = \text{Hash}(txid + pk + v + index)$ (+ means string concatenation). Then the point set $\hat{Y}$ could be mapped to the leaf nodes of the VT as described in Function 7. In particular, the stateless scheme could also be constructed from scratch for $\eta = 0$, where Block_η represents the genesis block, and all UTXOs in the S_η are from the coinbase transactions.
3) *Header Modifications:* The header of Block_η (Header_η) is modified after accumulator initialization to maintain the commitment and other auxiliary data. In addition to the original items (parent hash, timestamp, etc.), three new fields of Com_η, LW_η, and cnt_η are added to Header_η. The Com_η is the VT commitment to all UTXOs of S_η. The cnt_η counts the total number of the UTXOs from the genesis block to Block_η, no matter whether they are spent or not. So we have $cnt_\eta = n$ after the initialization. The LW_η is the latest witness for the last UTXO ($UTXO_n$) in S_η. A lightweight chain with all block headers will be locally maintained on each node.

The transaction procedure is modified as follows.

1) *TI:* The format of a transaction is modified to guarantee its validity by the attached witnesses. The input is changed to $input = \{(txid, pk, v, index), pos, wit, sig\}$, where *sig* is the sender's digital signature, and *pos* is the position of the UTXO in S_η. The modified transaction could be described formally as
$$TX = \left\{ ver, \underbrace{[\text{in}_1, \text{in}_2, \ldots,]}_{\text{Inputs}}, \underbrace{[\text{out}_1, \text{out}_2, \ldots]}_{\text{Outputs}} \right\} \quad (18)$$
where *Inputs* and *Outputs* are sets of referenced UTXOs and generated UTXOs, respectively. *ver* is the version number. The transaction in this form is broadcasted to the network.
2) *Transaction Validation:* After receiving the transaction, the validator first checks signatures (*sig*) to guarantee that the consumed UTXO belongs to the sender and then checks the membership of each referred UTXO to guarantee that it exists and is unspent. The membership checking can be done based on Function 10 as $\{0, 1\} \leftarrow \text{VerWit}(\text{pos}, h_i, \text{wit}_i, \text{Com}_\eta)$, where Com_η is maintained in the latest header (Header_η), and h_i could be obtained by $h_i = \text{Hash}(txid + pk + v + index)$. The transaction is valid if the sum of tokens locked in the inputs exceeds the values to be transferred. All valid transactions are collected in the local transaction pool.
3) *Block Proposal:* Each miner packs part of the transactions in the local pool into a new block, and the fields of Com_η, LW_η, and cnt_η in the header are updated to $\text{Com}_{\eta+1}$, $LW_{\eta+1}$, and $cnt_{\eta+1}$. The miner first removes all referred UTXOs from the accumulator as described in Function 13, during which LW_η and Com_η update with each deletion operation. Then the miner adds the new UTXOs into the accumulator as described in Function 14. In specific, the first new UTXO in the first transaction could be reformulated as $(cnt_\eta + 1, y_{cnt_\eta+1})$, the second UTXO in the first transaction could be reformulated as $(cnt_\eta + 2, y_{cnt_\eta+2})$, and so on until the last new UTXO in the last transaction. Finally, Com_η, LW_η, and cnt_η are updated to $\text{Com}_{\eta+1}$, $LW_{\eta+1}$, and $cnt_{\eta+1}$, respectively, corresponding to the changed state, which will be filled in the new header $\text{Header}_{\eta+1}$. After filling the original fields in the header, the block proposal $\text{Block}_{\eta+1}$ is broadcasted to the network.
4) *Commitment Check & Witness Update (CC&WU):* After receiving the new block $\text{Block}_{\eta+1}$, the nodes first validate all packed transactions in the same way as the validators and check the original fields as usual. Based on the Header_η maintained locally, the nodes check whether the $\text{Com}_{\eta+1}$, $LW_{\eta+1}$, and $cnt_{\eta+1}$ are updated correctly in the same way as the miners. During the process, the traders update the witnesses to correspond to $\text{Com}_{\eta+1}$ for the UTXOs controlled by themselves based on Functions 12–14, respectively. Finally, all nodes synchronize the latest commitment $\text{Com}_{\eta+1}$ by appending the $\text{Header}_{\eta+1}$ to the top of the lightweight chain maintained locally. The traders can then issue new transactions based on the updated witnesses in the next block period.

TABLE II
MAIN PARAMETERS IN VA-STV

α	VT branching factor	N	number of leaf nodes in VT
Y	set to be accumulated	n	number of items in Y
L_i	i-th leaf node in VT	$Path_i$	path for L_i
wit_i	witness for L_i	π	KZG proof
com	KZG commitment	Com	final commitment of VA
D	VT layer number	N_U	number of local UTXOs
$\overline{S_U}$	average size of UTXOs	N_{in}	number of inputs in a TX
B	bitwidth of Com	T	TX number in a block
B_M	commitment size in SMT accumulator	B_R	commitment size in RSA accumulator
N_I	number of inputs in a block	N_O	number of outputs in a block

TX is short for transaction.

E. Applicability of VA-STV for Ethereum-Like Blockchains

Bitcoin and Ethereum have similar block structures and transaction procedures but different state models. Bitcoin applies the UTXO model, where the new UTXOs are generated with the old UTXOs consumed atomically. In comparison, Ethereum applies the account-balance model, where a transaction causes the increase and decrease of two existing accounts. With this model, the sender can only attach the witness for its state to the transaction but cannot obtain the witness for the receiver's state following the normal cryptocurrency workflow. Therefore, VA-STV cannot be applied to Ethereum-like blockchains directly.

However, the VA-STV could be modified for its application in Ethereum-like blockchains. The first solution is to translate the account-balance model into the UTXO model according to the scheme proposed in [14], then the VA could be updated in the same way as in Bitcoin. The second solution is to make the miner maintain the entire state set for VA commitment updates. These two methods have pros and cons. The first has strong statelessness as the VA-STV applied in Bitcoin does but with the extra cost brought by the translation between two state models. The second is easy to deploy and more efficient for VA updates. However, its statelessness is weak because the miners still need to maintain the entire state set [40], and the system storage efficiency and decentralization are also weakened.

IV. Theoretical Analysis of the VA-STV Scheme

This section analyzes the performances of the VA-STV in terms of storage, communication, and computation overhead in Section IV-A–IV-C, respectively. The symbols and their meanings are listed in Table II for clarity.

A. Storage Overhead

In the VA-STV scheme, the local storage on each node includes two parts: 1) the lightweight chain formed by the headers as the on-chain data (for validators and miners) and 2) the controlled UTXOs and the corresponding witnesses as the off-chain data (for traders). Compared with the original header, the modified header in the VA-STV scheme has a larger size due to the new fields of *Com*, *LW*, and *cnt*. *Com* has the fixed length of B bits as a point in the EC_1. *LW* contains D commitments and D proofs, each of which has the fixed length of B bits as the point in the EC_1. The bit width of the *cnt* could be set as $\log N$ because its maximum value is N for the VT with N leaf nodes. Therefore, the extra storage overhead of the modified header can be expressed as

$$S_{\Delta}^{\text{Head}} = \log N + (2D + 1) \cdot B \tag{19}$$

where $D = \log_{\alpha} N$ is the depth of the α-ary VT.

For the off-chain data, if denoting the number of UTXOs controlled by a trader node and their average size as N_U and $\overline{S_U}$, respectively, the storage overhead could be expressed as

$$S_{\text{off}} = N_U \cdot \left(2B \cdot D + \overline{S_U}\right). \tag{20}$$

In the proposed scheme, we have $N = 2^{32}$, $B = 762$ and $\overline{S_U} \approx 80$ bytes. In this case, the size of the UTXO could be ignored compared with that of the witnesses, so the storage for the off-chain data on the trader is dominated by the witnesses. Compared with the total number of the UTXOs in the stateful design (for example, the state set of Bitcoin contains more than 84 million UTXOs [12]), the number of the UTXOs controlled by a node (N_U) is negligible (hundreds of UTXOs accounting for less than 0.001% of the total number). Therefore, the storage of the off-chain data in the VA-STV is negligible compared with the state data in the stateful design. Table III compares the sizes of the off-chain data in different stateless schemes (Boneh, Edrax, and VA-STV), where we assume that N leaf nodes are included in the Edrax and VA-STV. B_R and B_M are the sizes of the commitments in RSA and SMT accumulators, and their typical values are 3,072 and 256, respectively. As we can see, since the witness size does not increase with the number of UTXOs, the Boneh achieves the best storage efficiency for extremely large N, and the VT-STV is superior to the Edrax because the value of α is usually much larger than 10 (typically, α are set to 256 or 1024).

TABLE III
SIZES OF THE OFF-CHAIN DATA FOR DIFFERENT STATELESS SCHEMES

Scheme	Accumulator	Off-chain data
Boneh	RSA	$N_U \cdot B_R$
Edrax	SMT	$N_U \cdot B_M \cdot \log N$
VA-STV	VA	$N_U \cdot 2B \cdot \log_{\alpha} N$

B. Communication Overhead

In the VA-STV scheme, extra communication overhead is introduced in two phases: 1) TI and 2) CC&WU. Since the witness is attached for each referred UTXO in a transaction, extra communication overhead is the total size of all witnesses in the transaction, which could be calculated as

$$O_{\text{comm}}^{TI} = N_{\text{in}} \cdot 2B \cdot D \tag{21}$$

where N_{in} is the number of input UTXOs in the transaction.

In the VA-STV scheme, each block proposal needs to be broadcasted to other nodes. During the process, the extra communication overhead is introduced by the three new fields (*Com*, *LW*, and *cnt*) into the header and all witnesses attached in the transactions packed in the block. Since the number of referred input UTXOs can be as large as multiple thousands [41], the overhead introduced by the modified header could be ignored. Thus, for a new block with T transactions, the communication overhead could be calculated as

$$O_{\text{comm}}^{CC\&WU} = S_W \cdot \sum_{i=1}^{T} N_{\text{in}}^{i} \tag{22}$$

where $S_W = 2B \cdot D$ is the witness size.

It can be seen that the communication overhead in the two phases is proportional to the witness sizes. Since the witness sizes in the three stateless schemes keep the relationships of $O(1)$, $O(\log N)$, and $O(\log_{\alpha N})$ with N, respectively, although slightly inferior to the Boneh, the VA-STV performs far better than the Edrax in terms of communication overhead. For example, when $N = 2^{32}$ and $\alpha = 256$, the extra communication overheads for each UTXO in the Boneh, Edrax, and VA-STV are 3072, 8192, and 6096 bits, respectively. And the performance gap between the Edrax and VA-STV gets bigger for a larger N.

C. Computation Overhead

In the VA-STV, the computation overhead is introduced by accumulator initialization, transaction validation, new block proposal, and CC&WU, which are undertaken separately by the nodes with different roles.

1) Accumulator Initialization: The overhead for this process is undertaken by all nodes. First, n UTXOs are mapped as the leaf nodes in VT, and the final commitment is generated. The computation in this process includes the PI, the scalar multiplication (SM) in the elliptic curve, and the hash operations, where the last can be ignored due to its low complexity. In the α-ary VT, one PI is used to obtain the polynomial for a subtree, α SM operations are used to get the KZG commitment, so the computation overhead in this phase could be calculated as

$$O_{\text{comp}}^{AI-1} = N_P \text{ PI} + N_S \cdot \alpha \text{ SM} \tag{23}$$

where N_P and N_S are the numbers of PI and SM operations to get all commitments in D layers. For the α-ary VT with N leaf nodes, there are $\sum_{i=1}^{D} N/\alpha^i$ subtrees and $\sum_{i=1}^{D-1} N/\alpha^i$ nonleaf nodes, so we have $N_P = \sum_{i=1}^{D} N/\alpha^i$ and $N_S = \sum_{i=1}^{D-1} N/\alpha^i$. Considering that the complexity for committing to the empty subtree is very low in sparse VT ($n << N$), the computation overhead could be approximated as

$$O_{\text{comp}}^{AI-1} \approx n/(\alpha - 1) \text{ PI} + n \text{ SM}. \tag{24}$$

Correspondingly, the time complexity is $O(n)$ (or $O(N)$).

Second, the trader generates the witnesses for UTXOs controlled by itself, which is done mainly with the SM operations. Each witness includes D proofs, and α SM operations are needed to generate one proof after obtaining the commitment. So the computation overhead to obtain a witness could be calculated as

$$O_{\text{comp}}^{AI-2} \approx N_U \cdot \alpha \cdot \log_\alpha N \text{ SM}. \tag{25}$$

2) Transaction Validation: This is performed by the validators. For each referred input UTXO, the witness is formed by $D-1$ commitments and D proofs. With the support of the final commitment maintained in the header, the validator validates the transaction by checking the correlations between the D pairs of commitments and proofs. Since two Pairing operations are involved for each checking, the computation overhead can be estimated as

$$O_{\text{comp}}^{TV} = 2D \text{ Pairing}. \tag{26}$$

Since each Pairing operation can be taken as a multiplication between two elliptic curves, the complexity is $O(1)$ even though the it is time consuming. Finally, the computation complexity for transaction validation is $O(\log_{\alpha N})$.

3) Block Proposal: The generation of block proposal is performed by the miners, which involves the update of *Com* and *LW* by deleting the consumed UTXOs from and adding the new UTXOs to the VA. When deleting UTXOs, one PI and α SM operations are needed to update a commitment. The same computation overhead is needed for *LW* update, so the computation overhead for each deletion is

$$O_{\text{comp}}^{BP-\text{Del}} = 2D \text{ PI} + 2\alpha \cdot D \text{ SM}. \tag{27}$$

The computation overhead for adding a new UTXO to the VA is the same as that for deleting a UTXO, so the total computation for a block proposal is

$$O_{\text{comp}}^{BP} = (N_I + N_O) \cdot (2D \text{ PI} + 2\alpha \cdot D \text{ SM}) \tag{28}$$

where N_I and N_O are the total numbers of the consumed and generated UTXOs, respectively. Correspondingly, the complexity is $O(\log_{\alpha N}) + O(\alpha \log_{\alpha N})$.

4) Commitment Check: This is performed by all nodes. The validity of the updated final commitment *Com* could be determined by checking whether the updated *LW* matches the updated *Com*. The computation overhead is the same as that for transaction validation

$$O_{\text{comp}}^{CC} = 2D \text{ Pairing}. \tag{29}$$

5) Witness Update: This is only performed by traders. The operations for updating the witness for a UTXO are the same as that for updating *LW*, which involves D PI operations and $\alpha \cdot D$ SM operations. Therefore, the computation overhead for witness updates for N_U UTXOs can be expressed as

$$O_{\text{comp}}^{WU} = N_U \cdot (D \text{ PI} + \alpha \cdot D \text{ SM}). \tag{30}$$

The complexity is $O(\log_{\alpha N}) + O(\alpha \log_{\alpha N})$.

The computation complexities of different processes for the three stateless schemes are listed in Table IV. Since the complexity of hash operation is very low, the Edrax has the lowest computation overhead in practical implementation. For Boneh, the complexities of all processes are higher than Edrax and VA-STV, except for the transaction validation. In the practical applications, thousands of UTXOs would be consumed and generated in a block (e.g., 5000–14 000 UTXOs in a Bitcoin block), and the modular exponentiation operation is relatively expensive, so the computation overhead of the Boneh scheme would be much higher than that of the VA-STV scheme.

TABLE IV
COMPUTATION COMPLEXITIES DURING THE TRANSACTION PROCESS

Schemes	Edrax	VA-STV	Boneh
Accum	SMT	VA	RSA
Ops	Hash	PI,SM,Pairing	ModularExp
ComGen	$O(N)$	$O(N)$	$O(N)$
WitGen	$N_U \cdot O(\log N)$	$N_U \cdot O(\alpha \cdot \log_\alpha N)$	$N_U \cdot O(N)$
TxVer	$N_{in} \cdot O(\log N)$	$N_{in} \cdot O(\log_\alpha N)$	$N_{in} \cdot O(1)$
BlocPros	$(N_I + N_O) \cdot O(\log N)$	$N_I \cdot O(\log_\alpha N) + N_O \cdot O(\alpha \cdot \log_\alpha N)$	$O(N_I^2) + O(N_O)$
ComVer	$O(\log N)$	$O(\log_\alpha N)$	$O(N_I^2) + O(N_O)$
WitSync	$N_U \cdot (N_I + N_O) \cdot O(\log N)$	$N_U \cdot (N_I \cdot O(\log_\alpha N) + N_O \cdot O(\alpha \cdot \log_\alpha N))$	$N_U \cdot (O(N_I^2) + O(N_O))$

TABLE V
COMPARISON OF STORAGE AND COMMUNICATION OVERHEAD (BYTES)

Overhead		Edrax	Boneh	VA-STV
$Comm_i$	wit_i	1,024	384	762
$Comm_\eta$	wit_η	1,024,000	384,000	762,000
	Com_η	32	384	96
	Ad_η	1,028	0	766
$Stor_i$		1,104	464	842

$Comm_i$ and $Stor_i$ are the overheads for each UTXO.
$Comm_\eta$ is the overhead for a block.
Ad is the short for the auxiliary data.

V. SIMULATION EVALUATION OF THE VA-STV SCHEME

This section first compares the performances of the Edrax, Boneh, and VA-STV in storage and communication overhead in Section V-A. Section V-B compares the computation overhead of these stateless schemes based on the execution time in the same settings. The effects of the branching factor α on the performances of the VA-STV are estimated in Section V-C, where the tradeoff between the communication and computation overhead and the corresponding applicable scenarios are discussed.

A. Comparison of Storage and Communication Overhead

The extra storage and communication overhead for three stateless schemes is determined by the witness size. Without the loss of the generality, we suppose that all issued transactions are single-input–single-output, and 1000 transactions are packed in a block on average. For the Boneh, the bit width of the prime number is set as 3072 bits ($B_R = 3072$). For the Edrax, the number of the leaf nodes is 2^{32} ($N = 2^{32}$), and the hash algorithm is determined as SHA-256 ($B_M = 256$). In the VA-STV, the pairing is realized based on the elliptic curve BLS12_381 ($B = 762$) [42], with N and α set as 2^{32} and 256, respectively. For the Edrax and VA-STV, the fields added to the header (LW, cnt) are recorded as the auxiliary data. The storage and communication overhead in the three schemes is compared in Table V.

Assuming each node controls 1000 UTXOs locally, the off-chain storage volumes in three schemes are 1079, 454, and 823 kB. As we can see from the table, the Boneh has the highest storage and communication efficiency due to its complexity of $O(1)$ for witness size. The VA-STV is superior to the Edrax due to the smaller depth of the VT than the SMT for the same number of leaf nodes. This advantage of VT further enhances for a larger N. It should be noted that the advantages of the Boneh in storage and communication are at the cost of extremely expensive computation, which will be discussed in Section V-B.

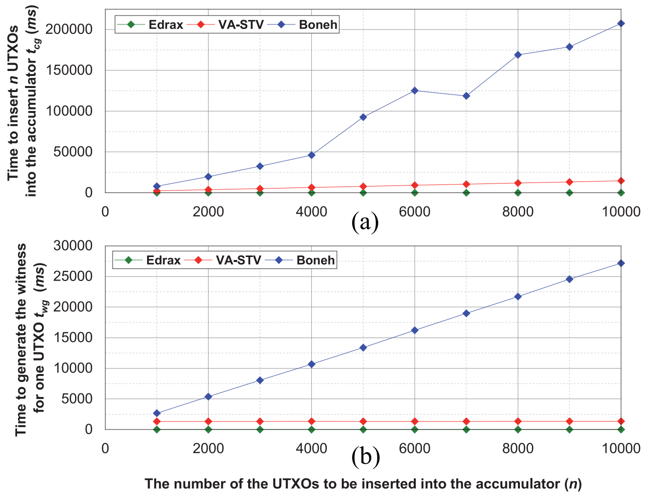


Fig. 4. Time for accumulator initialization and witness generation. (a) Time for initializing the VA with n UTXOs. (b) Time for generating the witness for one UTXO.

B. Computation Complexity Evaluation

A simulation platform is established using Python 3.7, which runs on a PC with Windows 10 OS, AMD Ryzen 7-5800H CPU@ 3.2 GHz, and 8-Gb memory. The UTXO is the basic unit to be processed, which is realized as the object with three fields of *meta*, *token*, and wit to represent the meta data, the number of tokens locked, and the witness, respectively. For a UTXO, the *meta* indicates its position in the sorted UTXO set, and the *token* acts as the *value* when inserting the UTXO into the VT as a leaf node. During the simulation, the time of the commitment generation (t_{cg}), witness generation (t_{wg}), transaction validation (t_{tv}), block proposal (t_{bp}), and witness update (t_{wu}) is recorded with different measures. For example, t_{wg}, t_{tv}, and t_{wu} is the average time for each UTXO, whereas t_{cg} and t_{bp} is the total time for the entire UTXO set (n UTXOs) and a block (T transactions), respectively. Thus, the parameters for each process are set separately, and the results are averaged over 1000 runs. The time for the commitment check is not involved in the estimation since it can be done in the same way as the transaction validation.

The time (t_{cg}) to insert n (1000–10 000) UTXOs into the accumulators and the time (t_{wg}) to generate one witness after the initialization are shown in subfigures (a) and (b) of Fig. 4, respectively, where the performances of the Edrax, VA-STV ($\alpha = 256$), and the Boneh are represented by the green, red, and blue lines, respectively. The number of leaf nodes in VT and SMT are 2^{32} ($N = 2^{32}$). For convenience, all intermediate variables during the accumulator initialization (such as converted prime numbers in Boneh and polynomial coefficients in the VA-STV) are cached to facilitate the witness generation. In Fig. 4, the green lines representing the Edrax are almost invisible due to the extremely cheap hash operations

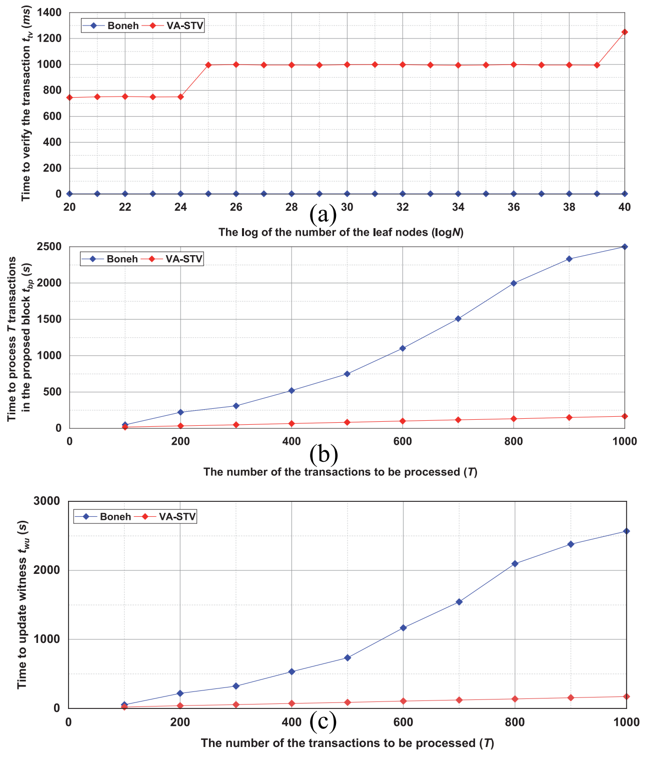


Fig. 5. Performances for other processes in Boneh and VA-STV. (a) Time to verify one transaction. (b) Time to propose a new block with T transactions. (c) Time to update the witness for one UTXO controlled by the trader.

of the SMT accumulator. In Fig. 4(a), the time of the VA-STV increases linearly with n due to the fixed time cost to obtain the polynomial (~2 ms by IFFT) and its commitment (~330 ms) for $\alpha = 256$. About four new polynomials should be calculated and committed when n is increased by 1000. For Boneh, before inserting the UTXO into the RSA accumulator, the time-consumed and unpredictable per-process is essential to convert the hash into the prime number of 3072 bits, which causes the volatility points, even the time also increases linearly with n approximately. Besides, the time of the Boneh is far larger than that of the VA-STV, and the difference increases for larger n. For example, the Boneh is 6× slower than the VA-STV when $n = 2000$, whereas 14× slower when $n = 10\,000$. In Fig. 4(b), since the witness generation time depends only on the VT depth, the VA-STV line keeps flat for different n ($D = 4$ for n varying in 1000–10 000). Since all converted prime numbers are cached, the time for Boneh increases linearly with n without any volatility points and is much larger than that of the VA-STV.

The time for other processes in the Boneh (blue line) and VA-STV (red line) is shown in Fig. 5, where the time of the Edrax is too small to be shown. Fig. 5(a) shows the relationship between the time to check transaction validity and the number of leaf nodes (N) in the VT. The time for Boneh keeps constant since the checking time only depends on B_R. The time for VA-STV is determined by the depth ($D = \log_\alpha N$, $\alpha = 256$) of the VT, which is raised by one as N increases α times. Thus, the time for VA-STV is slowly growing as N increases. In particular, the time remains the same when N increases from 2^{30} to 2^{39} ($D = \lceil \log_\alpha 2^{30} \rceil = \lceil \log_\alpha 2^{39} \rceil$), until N reaches 2^{40}. Due to the time-consumed pairing operations, the VA-STV takes more time than the Boneh for transaction validation (~250 ms versus ~3 ms). However, based on benchmarks in [43], the performance of the KZG implementation is highly correlated with the programming languages for different API calling and memory access rules. The Rust implementation is 15× faster than the Go implementation based on the same C++ libraries. The Python implementation here is not fully optimized due to the lack of Python libraries, which enlarges the gap between the two schemes. After enough optimizations in the code level, the time for transaction validation in VA-STV could be reduced to 12–16 ms, which is close to that in Boneh.

Fig. 5(b) shows the time to process T (0–1000) transactions when proposing the new block. As we can see, the time for VA-STV increases linearly with T, which is much shorter than that for Boneh. This is because each consumed or new UTXO has an independent path in the VT, which takes almost constant time to update the commitment following the path upward. The Boneh is quite inefficient, especially for the deletion operations, because the proofs for all the referred UTXOs need to be updated before each UTXO is removed from the accumulator. Fig. 5(c) shows the time to update one local witness based on T transactions packed in the received block. For convenience, there is no intersection between the referred UTXOs and those maintained locally. Since the trader needs to perform all the operations on a miner when updating the local witness, the curves for the two schemes demonstrate the same trends as in Fig. 5(b), and VA-STV is also more efficient than Boneh.

Based on the above results, we can get two conclusions. From the perspective of storage and communication overhead, Boneh is the best, and VA-STV is much better than Edrax. From the perspective of the computation overhead, Edrax is the best, and VA-STV outperforms Boneh. Therefore, a tradeoff between the advantages of Boneh and Edrax could be achieved by VA-STV, which can dramatically reduce the storage burden on each node without abusing the bandwidth and computing resources.

C. *Effect of* α

As analyzed in Section IV, when the number of the leaf nodes in the VT (N) is determined, the storage, communication, and computation overhead varies with the depth of VT and are determined by the branching factor α. For a larger α, the storage and communication overhead decreases due to the fewer layers, but the computation overhead increases due to more operations of PI and SM. Conversely, the computation overhead is reduced for smaller α at the cost of more storage and communication overhead. The impact of α on the communication overhead is shown in Fig. 6 ($N = 2^{40}$), where the x-axis is the logarithm of α, and that for Boneh and Edrax are provided as blue and green lines for comparison. As we can see, the witness sizes for the Boneh and Edrax are independent of α. The Boneh has the smallest witness size (384 bytes), which is determined by the bit width of the selected prime

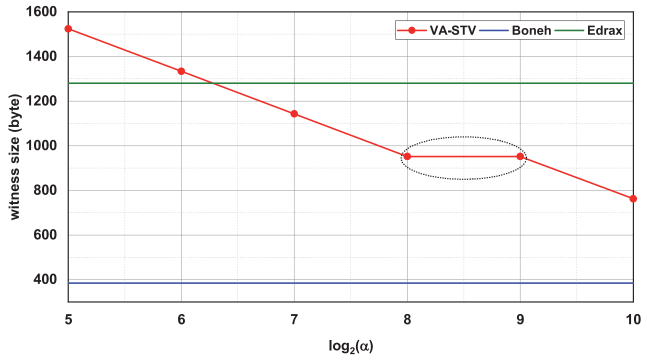


Fig. 6. Witness sizes for different α.

TABLE VI
TIME TO PROCESS A SUBTREE (MS) FOR DIFFERENT α

α	2^5	2^6	2^7	2^8	2^9	2^{10}
t_I	0.31	0.53	1.02	2.12	3.99	9.24
t_C	49.67	88.06	179.46	330.17	668.31	1378.26
t_P	51.26	87.98	178.25	332.50	665.76	1376.97

number ($B_R = 3072$) as analyzed in Section IV-B. The "step" in the line of VA-STV is caused by the ceiling operations ($D = \log_{2^8} N = \lceil \log_{2^9} N \rceil$). Besides, the Edrax has smaller witness sizes than VA-STV when $\alpha < 2^7$. This is because the proof for each layer in VA-STV has a larger bit width ($B = 762$) than that in Edrax ($B_M = 256$). But for $\alpha = 2^{10}$, the witness size in VA-STV is 0.6× of that in Edrax.

The impact of α on the computation overhead could be measured by the time to process a subtree with α children (obtaining the polynomial by IFFT, calculating the commitment and proof) since the subtree is the basic computation unit. The results for α of 2^5–2^{10} are listed in Table VI, where t_I, t_C, and t_P are the time for IFFT, commitment generation, and proof generation, respectively. As we can see, t_I, t_C, and t_P increase for a larger α in similar trends. Though IFFT is the precondition for commitment and proof generation, t_I is almost negligible compared with t_C and t_P, which means the vast majority of the computation overhead is introduced by SM over the elliptic curve. Thus, t_C is close to t_P for different α since the commitment and proof are generated by α SM operations. This conclusion is consistent with the complexity of Function 1 and 2 described in Section II-C. Therefore, in the scenario with limited bandwidth, the nodes should select larger α to compress the witness size further. Conversely, in a network with relatively rich bandwidth resources, the nodes can select smaller α to save the local computing power.

VI. CONCLUSION

This article proposed a VA-STV scheme to reduce the storage burden for the global state on each node in the blockchain-based IoT network. With the support of the KZG polynomial commitment, all UTXOs are summarized as the fixed-length commitment maintained in the block header. A witness is generated for each UTXO to guarantee its validity. The state transition is realized by updating the accumulator. Thus, no state is needed for the validators and miners, where the transactions are validated by checking the attached witnesses against the latest commitment. For the traders, only the UTXOs controlled by themselves and the corresponding witnesses need to be maintained locally for TI. The performance of the VA-STV scheme in terms of storage, communication, and computation overhead is obtained through theoretical analysis and simulations. It is compared with that of the Edrax and Boneh schemes. The evaluation results show that VA-STV achieves a tradeoff between the advantages of the Boneh and Edrax. It can dramatically reduce each node's storage burden without abusing the bandwidth and computing resources. Besides, VA-STV can easily adapt to different computing power and bandwidth requirements by adjusting the branching factor, which improves the adaptability and scalability of the scheme in scenarios with heterogeneous resource configurations. Furthermore, the security of the VA-STV is based on the secret unavailable to all participants, which guarantees that the witness cannot be forged. It should be noted that the VA-STV scheme is proposed for the world state, so it could be combined with the block body compression schemes (such as proposed in [44]) to decrease the overall blockchain storage burden dramatically.

REFERENCES

[1] J. Al-Jaroodi and N. Mohamed, "Blockchain in industries: A survey," *IEEE Access*, vol. 7, pp. 36500–36515, 2019.
[2] J. A. Jaoude and R. G. Saade, "Blockchain applications—Usage in different domains," *IEEE Access*, vol. 7, pp. 45360–45381, 2019.
[3] K. Gai, Y. Zhang, M. Qiu, and B. Thuraisingham, "Blockchain-enabled service optimizations in supply chain digital twin," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1673–1685, May/Jun. 2023.
[4] F. Khan, M. A. Jan, A. ur Rehman, S. Mastorakis, M. Alazab, and P. Watters, "A secured and intelligent communication scheme for IIoT-enabled pervasive edge computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 5128–5137, Jul. 2021.
[5] Q. Pan, J. Wu, A. K. Bashir, J. Li, S. Vashisht, and R. Nawaz, "Blockchain and AI enabled configurable reflection resource allocation for IRS-aided coexisting drone-terrestrial networks," *IEEE Wireless Commun.*, vol. 29, no. 6, pp. 46–54, Dec. 2022.
[6] A. Tiwari and R. Garg, "Adaptive ontology-based IoT resource provisioning in computing systems," *Int. J. Semantic Web Inf. Syst.*, vol. 18, no. 1, pp. 1–18, 2022.
[7] A. Dahiya, B. B. Gupta, W. Alhalabi, and K. Ulrichd, "A comprehensive analysis of blockchain and its applications in intelligent systems based on IoT, cloud and social media," *Int. J. Intell. Syst.*, vol. 37, no. 12, pp. 11037–11077, 2022.
[8] Z. Gao, D. Zhang, J. Zhang, L. Liu, D. Niyato, and V. C. M. Leung, "World state attack to blockchain based IoV and efficient protection with hybrid RSUs architecture," *IEEE Trans. Intell. Transp. Syst.*, early access, Apr. 26, 2023, doi: 10.1109/TITS.2023.3268222.
[9] W. Liang, Y. Fan, K.-C. Li, D. Zhang, and J.-L. Gaudiot, "Secure data storage and recovery in industrial blockchain network environments," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6543–6552, Oct. 2020.
[10] Y. Ren et al., "Multiple cloud storage mechanism based on blockchain in smart homes," *Future Gener. Comput. Syst.*, vol. 115, pp. 304–313, Feb. 2021.
[11] N. Kannengießer, S. Lins, C. Sander, K. Winter, H. Frey, and A. Sunyaev, "Challenges and common solutions in smart contract development," *IEEE Trans. Softw. Eng.*, vol. 48, no. 11, pp. 4291–4318, Nov. 2022.
[12] "Blockchain monitoring." 2022. [Online]. Available: https://blockchain.info/
[13] R. Zhang, F. R. Yu, J. Liu, R. Xie, and T. Huang, "Blockchain-incentivized D2D and mobile edge caching: A deep reinforcement learning approach," *IEEE Netw.*, vol. 34, no. 4, pp. 150–157, Jul./Aug. 2020.
[14] J. Zahnentferner, "Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies," IACR Cryptol. ePrint Arch., Bellevue, WA, USA, Rep. 2018/262, 2018.

[15] Q. Zheng, Y. Li, P. Chen, and X. Dong, “An innovative IPFS-based storage model for blockchain,” in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, 2018, pp. 704–708.

[16] C. Xu, C. Zhang, J. Xu, and J. Pei, “SlimChain: Scaling blockchain transactions through off-chain storage and parallel processing,” *Proc. VLDB Endowment*, vol. 14, no. 11, pp. 2314–2326, 2021.

[17] H. Chen and Y. Wang, “SSChain: A full sharding protocol for public blockchain without data migration overhead,” *Pervasive Mobile Comput.*, vol. 59, Oct. 2019, Art. no. 101055.

[18] Z. Guo, Z. Gao, H. Mei, M. Zhao, and J. Yang, “Design and optimization for storage mechanism of the public blockchain based on redundant residual number system,” *IEEE Access*, vol. 7, pp. 98546–98554, 2019.

[19] S. Jiang et al., “BZIP: A compact data memory system for UTXO-based blockchains,” *J. Syst. Archit.*, vol. 109, 2020, Art. no. 101809.

[20] L. Wang, Y. Tian, and D. Zhang, “Toward cross-domain dynamic accumulator authentication based on blockchain in Internet of Things,” *IEEE Trans. Ind. Informat.*, vol. 18, no. 4, pp. 2858–2867, Apr. 2022.

[21] D. Boneh, B. Bünz, and B. Fisch, “Batching techniques for accumulators with applications to IOPs and stateless blockchains,” in *Proc. 39th Annu. Int. Cryptol. Conf.*, Santa Barbara, CA, USA, Aug. 2019, pp. 561–586.

[22] A. Chepurnoy, C. Papamanthou, S. Srinivasan, and Y. Zhang, “Edrax: A cryptocurrency with stateless transaction validation,” IACR Cryptol. ePrint Arch., Bellevue, WA, USA, Rep. 2018/968, 2018.

[23] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the Internet of Things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[24] S. Nakamoto. “A peer-to-peer electronic cash system.” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[25] J. Kuszmaul, “Verkle trees (2018),” unpublished. [Online]. Available: https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf

[26] A. Kate, G. M. Zaverucha, and I. Goldberg, “Constant-size commitments to polynomials and their applications,” in *Proc. 16th Int. Conf. Theory Appl. Cryptol. Inf. Security*, Singapore, Dec. 2010, pp. 177–194.

[27] D. Čapko, S. Vukmirović, and N. Nedić, “State of the art of zero-knowledge proofs in blockchain,” in *Proc. 30th Telecommun. Forum (TELFOR)*, 2022, pp. 1–4.

[28] D. Boneh, J. Drake, B. Fisch, and A. Gabizon, “Efficient polynomial commitment schemes for multiple points and polynomials,” IACR Cryptol. ePrint Arch., Bellevue, WA, USA, Rep. 2020/81, 2020.

[29] V. Daza, C. Ràfols, and A. Zacharakis, “Updateable inner product argument with logarithmic verifier and applications,” in *Proc. 23rd IACR Int. Conf. Pract. Theory Public-Key Cryptogr.*, Edinburgh, U.K., May 2020, pp. 527–557.

[30] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, “Fast reed-solomon interactive oracle proofs of proximity,” in *Proc. 45th Int. Colloq. Automata Lang. Program. (ICALP)*, 2018, pp. 1–17.

[31] S. Bowe, A. Gabizon, and M. D. Green, “A multi-party protocol for constructing the public parameters of the pinocchio zk-SNARK,” in *Proc. Int. Conf. Finan. Cryptogr. Data Security*, 2019, pp. 64–77.

[32] Y. Gong, Y. Jin, Y. Li, Z. Liu, and Z. Zhu, “Analysis and comparison of the main zero-knowledge proof scheme,” in *Proc. Int. Conf. Big Data Inf. Comput. Netw. (BDICN)*, 2022, pp. 366–372.

[33] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, “Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2019, pp. 2111–2128.

[34] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, “An empirical analysis of anonymity in Zcash,” in *Proc. 27th USENIX Conf. Security Symp.*, 2018, pp. 463–477.

[35] B. Libert, “New secure applications of bilinear maps in cryptography,” Ph.D. dissertation, Dept. Appl. Sci. Microelectron. Lab., Universite Catholique de Louvain, Ottignies-Louvain-la-Neuve, Belgium, 2006.

[36] L. Paccagnella and G. Pierobon, “FFT calculation of a determinantal polynomial,” *IEEE Trans. Autom. Control*, vol. AC-21, no. 3, pp. 401–402, Jun. 1976.

[37] D. A. Luong and J. H. Park, “Privacy-preserving blockchain-based healthcare system for IoT devices using zk-SNARK,” *IEEE Access*, vol. 10, pp. 55739–55752, 2022.

[38] J. Tang, Z. Cao, J. Shen, and X. Dong, “LPCP: An efficient privacy-preserving protocol for polynomial calculation based on CRT,” *Appl. Sci.*, vol. 12, no. 6, p. 3117, 2022.

[39] X. Feng, K. Cui, and L. Wang, “PBAG: A privacy-preserving blockchain-based authentication protocol with global-updated commitment in IoV,” 2022, *arXiv:2208.14616*.

[40] F. Bonnet, Q. Bramas, and X. Défago, “Stateless distributed ledgers,” in *Proc. 8th Int. Conf. NETYS*, Marrakech, Morocco, 2021, pp. 349–354.

[41] “Statoshi info.” 2022. [Online]. Available: https://statoshi.info/

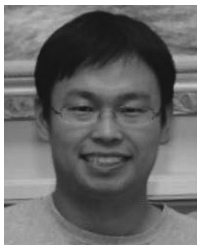
[42] R. S. Wahby and D. Boneh, “Fast and simple constant-time hashing to the BLS12-381 elliptic curve,” IACR Cryptol. ePrint Arch., Bellevue, WA, USA, Rep. 2019/403, 2019.

[43] B. Work and V. Rakutis, *KZG10 Polynomial Commitments in Blockchain Technologies*, Vilnius Univ., Vilnius, Lithuania, 2021.

[44] Z. Guo et al., “RNS-based adaptive compression scheme for the block data in the blockchain for IIoT,” *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 9239–9249, Dec. 2022.

Zhaohui Guo was born in Shanxi, China, in 1996. He is currently pursuing the Ph.D. degree with the School of microelectronics, Tianjin University, Tianjin, China.

His research interest includes microwave and blockchain technology.



Zhen Gao (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical and information engineering from Tianjin University, Tianjin, China, in 2005, 2007, and 2011, respectively.

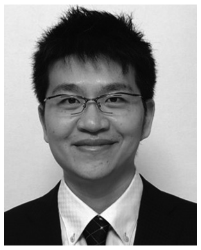
From October 2008 to November 2010, he was a Visiting Scholar with GeorgiaTech, Atlanta, GA, USA. From July 2011 to December 2014, he was a Postdoctoral Researcher with Wireless and Mobile Communication Research Center, Tsinghua University, Beijing, China. Since December 2014, he has been an Associate Professor with Tianjin University. His focus now is fault-tolerant signal processing, software-defined radio, and blockchain.

Qiang Liu (Senior Member, IEEE) received the Ph.D. degree from the Department of Electrical and Electronic Engineering, Imperial College London, London, U.K., in 2008.

From 2009 to 2011, he was a Research Associate with the Department of Computing, Imperial College London. He is currently a Professor with the School of Microelectronics, Tianjin University, Tianjin, China, with research interests in digital integrated circuit design, high-speed and low-power circuit system design, VLSI design optimization, and reconfigurable computing.

Lei Liu (Member, IEEE) received the B.Eng. degree in electronic information engineering from Zhengzhou University, Zhengzhou, China, in 2010, and the M.Sc. and Ph.D. degrees in communication and information systems from Xidian University, Xi’an, China, in 2013 and 2019, respectively.

From 2013 to 2015, he was employed by a Subsidiary of China Electronics Corporation, Shenzhen, China. From 2018 to 2019, he was supported by China Scholarship Council to be a visiting Ph.D. student with the University of Oslo, Oslo, Norway. From 2020 to 2022, he was a Lecturer with the School of Telecommunications Engineering, Xidian University, where he is currently an Associate Professor with the Guangzhou Institute of Technology. His research interests include vehicular ad hoc networks, intelligent transportation, edge intelligence, and distributed computing.



Mianxiong Dong (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and engineering from The University of Aizu, Aizuwakamatsu, Japan, in 2006, 2008, and 2013, respectively.

He is the Vice President and Professor of Muroran Institute of Technology, Muroran, Japan. He was a JSPS Research Fellow with the School of Computer Science and Engineering, The University of Aizu and was a Visiting Scholar with BBCR Group, University of Waterloo, Waterloo, ON, Canada, from April 2010 to August 2011. He was selected as a Foreigner Research Fellow (a total of three recipients all over Japan) by NEC C&C Foundation in 2011.

Dr. Dong is the recipient of the 12th IEEE ComSoc Asia–Pacific Young Researcher Award 2017, the Funai Research Award 2018, the NISTEP Researcher 2018 (one of only 11 people in Japan) in recognition of significant contributions in science and technology, the Young Scientists' Award from MEXT in 2021, the SUEMATSU-Yasuharu Award from IEICE in 2021, and the IEEE TCSC Middle Career Award in 2021. He is a Clarivate Analytics 2019, a 2021 Highly Cited Researcher (Web of Science), and a Foreign Fellow of EAJ.

Ning Zhang (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2015.

He is an Associate Professor with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, Canada. After that, he was a Postdoctoral Research Fellow with the University of Waterloo and the University of Toronto, respectively. His research interests include connected vehicles, mobile edge computing, wireless networking, and machine learning.

Dr. Zhang received eight Best Paper Awards from conferences and journals, such as IEEE Globecom and IEEE ICC. He also received the IEEE TCSVC Rising Star Award for outstanding contributions to research and practice of mobile edge computing and Internet of Things service. He is a Highly Cited Researcher. He serves as an Associate Editor of IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING, and IEEE SYSTEMS JOURNAL.

Mohammed Atiquzzaman (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering and electronics from The University of Manchester, Manchester, U.K., in 1984 and 1987, respectively.

He is the Edith Kinney Gaylord Presidential Professor with the School of Computer Science, University of Oklahoma, Norman, OK, USA. He teaches courses in Data Networks and Computer Architecture. Many of his current research activities are supported by NSF, NASA, Honeywell, Cisco, Oklahoma Department of Transportation, and the U.S. Air Force.

Dr. Atiquzzaman serves as the Editor-in-Chief of *Journal of Network and Computer Applications*, the Founding Editor-in-Chief of *Vehicular Communications*, and the Editor of many journals, including IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, and *Journal of Real-Time Image Processing*. He was the Past Co-Editor-in-Chief of the *Computer Communications Journal*.