Roll-DPoS\textsuperscript{\ding{113}}: A Randomized Delegated Proof of Stake Scheme for Scalable Blockchain-Based Internet of Things Systems

IoTeX Team

Abstract—Delegated Proof-of-Stake (DPoS) is an efficient, decentralized, and flexible consensus framework available in the blockchain industry. However, applying DPoS to the decentralized Internet of Things (IoT) applications is quite challenging due to the nature of IoT systems such as large-scale deployments and huge amount of data. To address the unique challenge for IoT based blockchain applications, we present Roll-DPoS, a randomized delegated proof of stake algorithm. Roll-DPoS inherits all the advantages of the original DPoS consensus framework and further enhances its capability in terms of decentralization as well as extensibility to complex blockchain architectures. A number of modern cryptographic techniques have been utilized to optimize the consensus process with respect to the computational and communication overhead.

Index Terms—Delegated Proof-of-Stake, Randomization, Distributed Key Generation, Threshold Signature, Bilinear Pairing

1 INTRODUCTION

According to the recent data from Juniper Research \cite{32}, the total number of connected Internet of Things (IoT) sensors and devices is set to exceed 50 billion by 2022 and the massive growth in connected devices over the next four years is mainly driven by edge computing services. With rapid increase of connected devices and services, IoT device manufactures and application developers are looking for a secure method for automating processes and exchanging information in real time. Blockchain appears to be a promising solution for enabling large-scale IoT applications in a decentralized and autonomous manner. As an open, transparent, immutable and distributed ledger, a blockchain is able to record transactions among IoT devices in a verifiable and permanent way. The self-executing smart contracts further offer a standardized method to accelerating data exchange and realizing workflow automation for smart devices without any interference and coordination of third parties.

The consensus protocol, which allows secure updating of a distributed shared state, is the core component of a blockchain. Achieving consensus ensures that all nodes in the network agrees upon a consistent global view of the blockchain state. The two key properties of a consensus protocol are: i) Safety/Consistency: All honest nodes produce the same output and the outputs produced by the honest nodes are valid; and ii) Liveness: All honest nodes in consensus eventually produce a value. A secure and robust consensus protocol needs to be tolerant a wide variety of Byzantine behaviors, including, but not limited to, failures of network nodes, partition of the network, message delay, out-of-order and corruption. While various consensus mechanisms have been extensively studied in the distributed systems community for closed systems for decades, the public, permissionless blockchains have revitalized the field and resulted in a multitude of new designs during the past few years. The interested reader is referred to \cite{3}, \cite{14} for good literature surveys and recent progress regarding to this topic.

It is well-known that designing a secure and robust consensus protocol for permissionless blockchains is quite challenging. The introduction of resource-constrained and heterogeneous IoT devices has added additional layer of design complexity. Among existing blockchain consensus protocols in the literature, the Delegated Proof-of-Stake (DPoS) framework proposed by Larimer \cite{33} sheds some light on blockchain based IoT applications in practice. In D PoS, stakeholders elect a certain number of network nodes (e.g., 21) as block producers, which are responsible and rewarded for generating and adding blocks to the blockchain. Each block producer takes turn proposing a block and distributes (partial) rewards back to the stakeholders. The election of block producers is a continuous process so that they have a strong incentive to provide high-quality service for maintaining their roles as block producers.

By electing a small group of powerful nodes to execute the expensive consensus process on behalf of the entire network, DPoS is able to accommodate the resource limitations and heterogeneity of IoT devices. However, the D PoS framework has some shortcomings when used in practice. Firstly, the block producer group is relatively static so that sufficient decentralization may never be achieved. Secondly, it is quite difficult, if not impossible, for the D PoS framework handling complex blockchain architectures where multiple, large-scale blockchain applications run simultaneously. To address these issues, we propose Roll-DPoS, a randomized and scalable variant of the D PoS framework. Roll-DPoS runs in epochs and combines multiple modern cryptographic techniques such as distributed key generation, BLS threshold signature, random beacon, etc., together with the Practical Byzantine Fault Tolerance (PBFT) consensus protocol to realize random selection of block producers from a dynamically updated candidate pool in each epoch and ensure instant finality for the proposed blocks simultaneously. In addition, Roll-DPoS
supports complex blockchain architectures by auto-scaling the candidate pool and scheduling multiple group of block producers with respect to the number of provisioned sub-chains in the system. When compared to the original DPoS framework, Roll-DPoS achieves a good trade-off in terms of decentralization, safety and elasticity.

The rest of the yellow paper is organized as follows. Section 2 give an overview of the cryptographic building blocks in the Roll-DPoS design as well as the DPoS framework, followed by the detailed description of the four Roll-DPoS core components and workflow in Section 3. Section 4 describes the implementation parameters and optimization techniques used in the short-lived BLS threshold signature scheme. Finally, we conclude this yellow paper in Section 5.

2 PRELIMINARIES AND CRYPTOGRAPHIC BUILDING BLOCKS

2.1 Threshold Secret Sharing

The notion of threshold secret sharing was introduced independently by Shamir [45] and Blakley [9] in 1979. For distributing shares of a secret \( x \) among \( n \) entities, a trusted dealer TD randomly chooses a polynomial \( f(z) \) of degree \( t \) over \( \mathbb{Z}_q \):

\[
f(z) = \sum_{i=0}^{t} a_i z^i,
\]

where the secret \( x \) is stored at the constant term \( a_0 \), i.e., \( f(0) = a_0 = x \). TD then computes the secret share \( x_i = f(id_i) \mod q \) for each entity, where \( id_i \) is an identifier of entity \( P_i \), and transfers \( x_i \) to \( P_i \) through a secure channel. Upon receiving the secret shares, any group of \( t+1 \) entities is able to recover the secret using the Lagrange interpolation formula:

\[
f(z) = \sum_{i=1}^{t+1} x_i \lambda_i(z),
\]

where \( \lambda_i(z) = \prod_{j=1,j\neq i}^{t+1} \frac{z-id_i}{i-j} \mod q \). Note that the secret \( x \) can also be written as \( x = f(0) = \sum_{i=1}^{t+1} x_i \lambda_i(0) \mod q \). Therefore, the secret \( x \) can be recovered only if at least \( t+1 \) shares are combined and the coalition of less than \( t+1 \) entities cannot derive any information about \( x \).

2.2 Feldman’s Verifiable Secret Sharing

A basic threshold secret sharing schemes is defined to solely resist passive attacks in which all entities involved run the protocol as prescribed by the scheme. In practice, however, a secret sharing scheme might need to withstand active attacks. To this end, Chor et al. [18] introduced the concept of verifiable secret sharing (VSS), which aims to thwart the following two types of active attacks:

- A malicious dealer might send incorrect shares to some or all of the entities during the share distribution phase;
- Malicious entities might submit incorrect shares during the secret reconstruction phase.

1. Without loss of generality and for easy exposition, we consider a group consisting of the first \( t+1 \) entities.

A well-known VSS scheme under the discrete logarithm setting is due to Feldman [24], which is an extension of Shamir’s scheme and requires the dealer to send additional values to all the entities so that each share can be checked for validity. More specifically, let \( p \) and \( q \) be two large primes and \( g \in \mathbb{Z}_p \) be an element of order \( q \) (i.e., \( q \mid p-1 \)). Feldman’s VSS scheme achieves the share verifiability as follows:

- **Witness Generation**: TD randomly chooses a polynomial \( f(z) \) and distributes the secret shares \( x_i’s \) as described in Section 2.1. Moreover, TD computes witnesses \( W_i = g^{f_i} \mod p \) for \( i \in [0, t] \) and publishes those \( W_i’s \) in some public domain.
- **Share Verification**: Upon receipt of the share \( x_i \), entity \( P_i \) verifies its validity by checking the equation

\[
g^{x_i} = \prod_{j=0}^{t} W_j^{x_{ij}} \mod p.
\]

Feldman’s VSS scheme is secure under the discrete logarithm assumption and any set of \( t \) entities is not able to find the secret from their shares, given that they can see all the witnesses \( W_i, i \in [0, t] \).

2.3 Pedersen’s Distributed Key Generation

In decentralized applications where a TD does not exist, a distributed key generation (DKG) protocol is an essential component for generating cryptographic keys and initializing the cryptosystem. Pedersen [44] proposed a simple DKG scheme in which all the entities run multiple parallel instances of the Feldman’s VSS scheme and collaboratively generate shares that are corresponding to the Shamir’s secret sharing of a random value. For a distributed system with \( n \) entities \( P_1, \ldots, P_n \), Pedersen’s DKG protocol works as follows:

- Each \( P_i \) chooses at random a polynomial \( f_i(z) \) of degree \( t \) over \( \mathbb{Z}_q \):

\[
f_i(z) = a_{i0} + a_{i1} z + \cdots + a_{it} z^t,
\]

where \( f_i(0) = a_{i0} = x_i \) is a random secret that \( P_i \) selects. Each \( P_i \) computes the shares \( \bar{x}_{ij} = f_i(id_j) \mod q \) for \( j \in [1, n], j \neq i \) and sends \( \bar{x}_{ij} \) to \( P_j \) through a secure channel. Moreover, \( P_i \) computes the witnesses \( W_{ik} = g^{x_{ik}} \mod p \) for \( k \in [0, t] \) and broadcasts them into the system. Denote \( W_{i0} \) by \( y_i \).

- Each \( P_j \) verifies the shares received from other entities by checking the equation for \( i \in [1, n] \):

\[
g^{\bar{x}_{ij}} = \prod_{k=0}^{t} W_{ik}^{x_{ik}} \mod p.
\]

If the verification fails for an index \( i \), \( P_j \) broadcasts a complaint against \( P_i \). Each \( P_i \) who receives a complaint from \( P_j \) broadcasts the value \( x_{ij} \).

- Each entity marks as disqualified a peer that either receives at least \( t+1 \) complaints or answers a complaint with a false value, and then creates a set QUAL which contains all the qualified entities.

- Each \( P_i \) computes its secret share as \( x_i = \sum_{j \in \text{QUAL}} x_{ij} \) and the system private key \( x \), which is not known to any entity, is equal to \( \sum_{i \in \text{QUAL}} x_i \mod q \).
2.4 Bilinear Pairing

Let $N$ be a positive integer and $G_1$ and $G_2$ be additively-written groups of order $N$ with identity $O$, and let $G_T$ be a multiplicatively-written group of order $N$ with identity 1. A bilinear pairing – or pairing for short – is a computable, non-degenerate function:

$$e : G_1 \times G_2 \rightarrow G_T,$$

satisfying the additional properties. The most important property for cryptographic applications is so-called bilinearity, namely:

$$e(aP, bQ) = e(P, bQ)^a = e(aP, Q)^b = e(P, Q)^{ab},$$

for all $P \in G_1, Q \in G_2$ and $a, b \in \mathbb{Z}_N$. In practice, the groups $G_1$ and $G_2$ are subgroups or quotient groups of an elliptic curve defined over a finite field $\mathbb{F}_q$, or one of its extensions and $G_T$ is a subgroup or quotient group $\mathbb{F}_{q^k}$, where $k$ is the embedded degree. The security of pairing-based cryptography requires that the discrete logarithm problem on $G_1, G_2$ and $G_T$ is sufficiently difficult. For more details about the bilinear pairing and its cryptographic applications, the interested reader is referred to [26], [43].

2.5 The BLS Signature Scheme and Its Extensions

2.5.1 The BLS Signature Scheme

In [13], Boneh et al. described a simple, deterministic short signature scheme, namely the BLS short signature. It works in any Gap Diffie-Hellman (GDH) group and requires a hash function from the message space onto the group. Let $g_1, g_2$ and $g_T$ be an arbitrary generator of $G_1, G_2$ and $G_T$, respectively, and $H_1 : \{0, 1\}^* \rightarrow G_1$ be a hash function with values in $G_1$. The BLS signature scheme consists of the following three algorithms:

- **KeyGen()**: Choose a random $\alpha \leftarrow \mathbb{R}_{\mathbb{Z}_N}$ and set $h \leftarrow g_2^\alpha \in G_2$. The system public key is then computed as $y = \prod \{\text{GQUA}_{i, y_i} \; (\text{mod} \; p)\}$. Note that $y = g^t$.

Due to the simple mathematical structure, the BLS signature scheme supports a variety of extensions [10], [12], including threshold signatures, multisignatures, aggregate signatures, and blind signatures.

2.5.2 The BLS Threshold Signature Scheme

In [12], Boneh et al. showed that one can build a non-interactive threshold signature scheme based on the plain BLS signature scheme and the Shamir’s secret sharing. For a system with $n$ entities $P_1, \ldots, P_n$, $t$ of which may be corrupted, and a trusted dealer $TD$, the BLS threshold signature consists of the following five algorithms:

- **KeyGen()**: $TD$ chooses a random $\alpha \leftarrow \mathbb{R}_{\mathbb{Z}_N}$ and set $h \leftarrow g_2^\alpha \in G_2$. The system private/public key pair is $(sk, pk) = (\alpha, h)$. $TD$ also generates a random polynomial $f(z) \in \mathbb{Z}_q$ of degree $t$, such that $f(0) = sk$. $TD$ then computes $n$ private key shares $sk_i = f(Id_i)$ and public key shares $pk_i = g_2^sk_i$ for $i \in [1, n]$. The private key share $sk_i$ as well as the public key shares $pk_j, j \in [1, n], j \neq i$ are sent to $P_i$ through a secure channel.

- **SignShareGen(sk_i, m)**: Given a private key share $sk_i$ and a message $m \in \{0, 1\}^*$, output the signature share $\sigma_i = H_1(m)sk_i \in G_1$.

- **SignShareVerify(pk_i, m, \sigma_i)**: Given a public key share $pk_i$, a message $m$, and a signature share $\sigma_i$, check whether $e(\sigma_i, g_2) = e(H_1(m), pk_i)$. If the verification is successful, $\sigma_i$ is a valid signature share received from $P_i$.

- **SignShareCombine(\sigma_{i_1}, \ldots, \sigma_{i_t+1})**: Given $t + 1$ valid signature shares $\sigma_{i_1}, \ldots, \sigma_{i_{t+1}}, \{i_1, \ldots, i_{t+1}\} \subseteq \{1, \ldots, n\}$, output the signature $\sigma = \prod_{j=1}^{t+1} \sigma_{i_j}$, where $\lambda_{i_k} = \prod_{j=1, j \neq k}^{t+1} \frac{0-Id_{i_k}}{Id_{i_j}-Id_{i_k}}, k = 1, \ldots, t+1$ are Lagrange coefficients.

- **Verify(pk, m, \sigma)**: Given a public key $pk$, a message $m$, and a combined signature $\sigma$, check whether $e(\sigma, g_2) = e(H_1(m), pk)$ and output “accept” or “reject” accordingly.

2.6 Delegated Proof of Stake

Delegated Proof of Stake (DPoS) is a robust and flexible blockchain consensus mechanism invented by Larimer [33] in 2014, which leverages the voting power of the stakeholder to resolve consensus issue in a fair and democratic manner. DPoS was first applied by Larimer to power the blockchain project BitShares [8] and further refined in his subsequent projects Steem [48] and EOS [23]. Other blockchain projects, such as Ark [1], Lisk [35], Tezos [49], etc., also adopted the similar DPoS framework with certain feature changes.

Unlike the traditional Proof of Stake (PoS) system that involves the participation of the entire network to validate a transaction, DPoS concentrates block production in the hands of a limited number of semi-trusted delegates. The DPoS consensus framework consists of the following four major steps as shown in Fig. 1:

1) **Block Producer Election**. The cryptocurrency token holders cast votes to elect an odd number of users...
3 Roll-DPoS: A Randomized Delegated Proof of Stake Consensus Algorithm

In this section, we present the design of Roll-DPoS, a randomized delegated proof of stake consensus algorithm, in great detail.

3.1 Design Rationale

Like other consensus algorithms, the DPoS consensus framework also makes certain trade-offs among safety, scalability, and decentralization of block production to address the scalability trilemma. By limiting the number of block producers, DPoS is able to reduce the computational and communication overhead significantly during the consensus process, thereby leading to fast block production, high throughput, and low latency. While there are still heated discussions regarding to the design principle of DPoS (see [19], [34] for examples), we believe that the DPoS consensus framework fits well for blockchain based IoT applications due to the following main reasons:

- It is quite difficult, if not impossible, for IoT devices performing complex consensus protocols, due to their constrained computational and storage resources.
- It is not energy and throughput efficient for IoT devices running a consensus protocol across the entire network, due to their constrained power and bandwidth.

However, considering the salient characteristics of IoT systems such as large-scale deployments and huge amount of data, the DPoS consensus framework needs to be further enhanced to accommodate more complex blockchain architectures and decentralized IoT applications, which leads to the design of Roll-DPoS, a randomized delegated proof of stake consensus algorithm.

A high-level description of Roll-DPoS is illustrated in Fig. 2. In a nutshell, Roll-DPoS initiates a candidate pool through a community voting process with the aid of the Ethereum blockchain and nodes that receive the most number of votes from the community become the potential block producers. The Roll-DPoS consensus algorithm runs in epochs and each epoch is further composed of multiple sub-epochs. At the beginning of the epoch, a fixed number of block producers are randomly chosen from the candidate pool using a cryptographic hash function as well as a random beacon that is generated from the previous epoch. The selected block producers then perform the Pedersen’s DKG protocol to generate short-lived, epoch-specific private key shares that will be used to sign the messages in the Practical Byzantine Fault Tolerance (PBFT) process [15] with the short-lived BLS threshold signature scheme. The four core components (CCs) of the Roll-DPoS design are detailed in the following subsections.

3.2 CC-I: Ethereum Assisted Bootstrapping

The Roll-DPoS protocol is bootstrapped with the aid of the Ethereum blockchain and composed of the three steps as detailed below. The bootstrapping phase results in the selection of $n$ block producers for the first epoch through a community-based voting mechanism.

3.2.1 Step 1: Block Producer Self-Nomination

In Step 1, any node can self-nominate in the community and register to become a potential block producer. During this phase, a node usually sets up a campaign website (see [16] for an example) for attracting community members to vote for him/her, which specifies the software and hardware resources available for hosting block producer servers as well as the terms for block reward distribution, among many other things. The self-nomination process must be completed before all the community members are able to start the voting process.
3.2.2 Step 2: Block Producer Candidate Pool Formation

In Step 2, the community members vote for an initial block producer candidate pool of $N$ nodes by sending special Ethereum voting transactions that contain ‘0’ in the VALUE field and ‘vote’ in the DATA field. Each transaction indicates that a community member (i.e., an ERC-20 token wallet holder) has pledged backing towards a candidate and all the ERC-20 tokens in the wallet will be tallied at the end of a pre-defined cut-off time. At that moment, the Ethereum token contract freezes all the accounts and takes a snapshot of the beginning balances for all the ERC-20 token holders. Moreover, a vote counting process is executed and the highest-backed set of $N$ candidates, denoted by $\{C^{(1)}_1, \ldots, C^{(1)}_N\}$, are selected to form the initial block producer candidate pool.

3.2.3 Step 3: Block Producer Selection

In Step 3, $n$ block producers, denoted by $\{BP^{(1)}_1, \ldots, BP^{(1)}_n\}$, are chosen from the candidate pool as the bootstrapping nodes for the first epoch using a deterministic random bit generator (DRBG) [4]. DRBG can be efficiently realized with a block cipher or cryptographic hash function (see [2] for an example) and a nothing-up-my-sleeve number, e.g., the hash of the string “IoTeX”, is used as the initial seed $s_0$ for the DRBG. The $n$ bootstrapping nodes are then selected by sorting the output of DRBG. More specifically, each candidate computes

$$o^{(1)}_i = \text{DRBG}(s_0, pk^{(1)}_i, 1), \quad i = 1, \ldots, N,$$

where $pk^{(1)}_i$ is the public key of the candidate $C^{(1)}_i$ and ‘1’ is the epoch number of the first epoch, followed by the sorting of all $o^{(1)}_i$’s and selection of the top $n$ candidates as the initial block producers.

3.3 CC-II: Short-Lived BLS Threshold Signature Based PBFT Consensus

The Roll-DPoS protocol utilizes a short-lived BLS threshold signature based PBFT variant to reach consensus and instant finality for block proposals among block producers in each epoch, which takes advantage of a short-lived BLS threshold signature scheme at the 70-bit² security level coupling with a distributed key share generation process at the beginning of each epoch.

3.3.1 Distributed Key Share Generation

At the beginning of each epoch, $n$ block producers of the current epoch jointly perform the Pedersen’s DKG scheme (see Section 2.3 for details). As a result, each block producer obtains a 158-bit key share of an unknown group private key. These private key shares will be used to sign the PREPARE and COMMIT messages with the BLS threshold signature scheme by block producers during the PBFT consensus process. To further amortize the cost of DKG, an epoch has been further divided into $L$ sub-epochs and the DKG scheme is only being executed once per epoch.

2. The implementation of the short-lived BLS threshold signature utilizes the bilinear pairing over a Miyaji-Nakabayashi-Takano (MNT) curve [42] with embedding degree 6.
3.3.2 PBFT Consensus with BLS Threshold Signature

The non-interactive BLS threshold signature scheme (see Section 2.5.2 for details) has been integrated into the PBFT consensus process of the Roll-DPoS protocol for improving performance. In Roll-DPoS, block producers take turns proposing blocks in each sub-epoch. There is only one legitimate block producer at any given time and other block producers act as the block validators. Upon receiving and validating a block proposal, a validator utilizes the \texttt{SignShareGen}() algorithm to sign a PREPARE message with its own private key share $s_{ki}$, where the PREPARE message can be either ‘YES’ or ‘NO’, depending on the validation result of the block proposal. The validator then broadcasts the signed PREPARE message to the network. Once the number of ‘YES’ in collected PREPARE messages exceeds a pre-defined threshold, the validator aggregates all the signature shares and verifies the aggregate signature using the \texttt{SignShareCombine}() and \texttt{Verify}() algorithms, respectively. If the aggregate signature passes the verification, the validator broadcasts a signed ‘YES’ as the COMMIT message to the network, which will be processed similarly as the PREPARE message by other validators. A validator appends the proposed block to the blockchain only when the block proposal passes the PBFT consensus process.

3.4 CC-III: Random Beacon Enabled Block Producer Rotation

In Roll-DPoS, block producers will be randomly selected from a candidate pool using the DRBG at the beginning of each epoch. For the $i^{th}$ epoch, each candidate computes

$$o_{i}^{(j)} = \text{DRBG}(s_{j-1}, pk_{i}^{(j)}, j), \ i = 1, \ldots, N,$$

where $s_{j-1}$ is the random beacon generated by the block producers during the $(j-1)^{th}$ epoch using the BLS threshold signature scheme. $pk_{i}^{(j)}$ is the public key of the candidate $C_{i}^{(j)}$ and $j$ is the current epoch number. All the $o_{i}^{(j)}$'s are then sorted and the top $n$ candidates are selected as the block producers for the current epoch. The generation process of random beacons is similar to that in DFINITY [29]. Besides proposing blocks in the $j^{th}$ epoch, a block producer $i$ also uses the \texttt{SignShareGen}() algorithm to generate a random beacon share $s_{j}^{(i)}$ as follows:

$$s_{j}^{(i)} = \text{SignShareGen}(sk_{i}^{(j)}, s_{j-1} || j),$$

where $sk_{i}^{(j)}$ is the private key share of the block producer $i$ in the $j^{th}$ epoch and $s_{j-1}$ is the random beacon of the previous epoch. $sk_{i}^{(j)}$ will also be included as of the block by the block producer. Once the number of random beacon shares exceeds the predefined threshold, every node in the candidate pool is able to compute and verify $s_{j}$ using the \texttt{SignShareCombine}() and \texttt{Verify}() algorithms, respectively, and then obtain consistent global view regarding to the block producers for the next epoch. In the case that the number of random beacon shares contained in the blockchain is less than the threshold at the end of epoch, the random beacon will be updated pseudorandomly, i.e.,

$$s_{j} = H(s_{j-1} || j),$$

where $H(\cdot)$ is a cryptographic hash function.

3.5 CC-IV: Auto-Scaling Candidate Pool for Complex Blockchain Architectures

The IoTeX blockchain is composed of a root chain and unlimited number of on-demand provisioned sidechains. To support this complex blockchain architecture as well as large-scale IoT DApps, we design an auto-scaling candidate pool to power consensus process for sidechains in Roll-DPoS. The basic idea is to dynamically adjust the size of the candidate pool based on the number of sidechains running simultaneously in the system. In our design, the blockchain nodes in the candidate pool has an option to become the block producers for subchains if they are not being selected as the block producers for the root chain at the current epoch. As a result, the candidate pool is divided into multiple subgroups, one of which acts as block producers for the root chain and others for different subchains. Whenever the number of idle nodes (i.e., those nodes are not block producers for either root chain or sidechains) is larger than (resp. less than) a predefined threshold, a certain number of blockchain nodes will be evicted from (resp. filled into) the candidate pool. The auto-scaling candidate pool is able to power large-scale blockchain-based IoT systems by enabling a significant number of nodes participating in the consensus process for different applications. Our novel design not only solves the scalability issues of sidechains, but also enables more blockchain nodes to become block producers and receive rewards.

4 Implementation Notes

We implement the short-lived BLS threshold signature scheme using the Tate pairing over a MNT curve with embedding degree 6 [42]. The elliptic curve is defined by the equation

$$E/\mathbb{F}_p : y^2 = x^3 - 3x + b$$

with the following parameters represented in hexadecimal:

$$p = \begin{array}{l} 7\text{ddc6a}13 \ a2e3\text{ddb}1 \ 749d01\text{95} \\ \text{bb}9f14\text{cf} \ 44626303 \ (159\text{-bit}) \end{array}$$

$$b = \begin{array}{l} 21c3f3\text{ac} \ 7864d1f1 \ f99273d0 \\ f828d365 \ 7d8cf4d4 \ (158\text{-bit}) \end{array}$$

$$n = \begin{array}{l} 3\text{eee}5309 \ d171eed8 \ ba4e12de \\ f44414fd \ 17d369b7 \ (158\text{-bit}) \end{array}$$

Note that the order of the curve $E$ has a cofactor $h = 2$ (i.e., \#$E(\mathbb{F}_p) = 2 \cdot n$) and these parameters can be represented in terms of an 80-bit integer $z = \text{d}bd7d316e\text{ad}514\text{bb}8f95$ (in hexadecimal) such that $n = (z^2 - z + 1)/3$ and $p = 2n + z$. Since the MNT curve in question has embedding degree $k = 6$, pairings are computed over points in $E(\mathbb{F}_p)$. For efficiency reasons, we can restrict the first input to be a point in $E(\mathbb{F}_p)[n]$ and compress the second input in $E(\mathbb{F}_p)$ to a point in a quadratic twist $E'(\mathbb{F}_p^2)$. Let $D \in \mathbb{F}_p^*$ be a quadratic non-residue over $\mathbb{F}_p$. A quadratic twist of $E$, denoted by $E'$, over $\mathbb{F}_p$, for which $n \mid \#E'(\mathbb{F}_p^2)$ is defined by the following equation [30]

$$E'/\mathbb{F}_p^2 : Dy^2 = x^3 - 3x + b.$$
The injective group homomorphism \( \phi : E'(\mathbb{F}_{p^g}) \to E(\mathbb{F}_{p^g}) \) is given by \( (x, y) \mapsto (x, \sqrt[p^g]{y}) \), which allows us to map points in the quadratic twist \( E'(\mathbb{F}_{p^g}) \) to points in \( E(\mathbb{F}_{p^g}) \).

We propose a multibase variant of Miller’s algorithm to compute the Tate pairing efficiently using the Jacobian coordinates and all the improved algorithms and optimization techniques are detailed in Appendix A.

5 Conclusion

In this yellow paper, we describe the design rationale as well as the four core components of the Roll-DPoS consensus scheme that aims to achieve good trade-off among decentralization, safety and elasticity when running consensus in large-scale blockchain-based IoT systems. Roll-DPoS is designed using the modern cryptographic techniques and keeps in mind various restrictions of IoT devices and applications. By enabling the random selection of block producers as well as auto-scaling the candidate pool for handling complex blockchain architectures, Roll-DPoS addresses a number of limitations of the previous DPoS framework. IoTeX is currently working on the implementation optimization as well as large-scale simulations of the Roll-DPoS consensus scheme towards the next release of our testnet.

Appendix A

A.1 Tate Pairing on Elliptic Curves

Let \( \mathbb{F}_q \) be a finite field with \( q = p^r \) elements, where \( p > 3 \) is a prime and \( r \) is a positive integer. An elliptic curve \( E(\mathbb{F}_q) \) is the set of solutions \((x, y)\) over \( \mathbb{F}_q \) satisfying an equation of the form \( E : y^2 = x^3 + ax + b \), where \( a, b \in \mathbb{F}_q \) and \( 4a^3 + 27b^2 \not\equiv 0 \pmod{q} \), together with an additional point at infinity, denoted by \( \mathcal{O} \). Note that the same equation also defines curves over \( \mathbb{F}_{p^k} \) for \( k > 0 \). Let \( #E(\mathbb{F}_{p^k}) \) denote the number of points on an elliptic curve \( E(\mathbb{F}_{p^k}) \), which is called the order of the curve over the field \( \mathbb{F}_{p^k} \). The points on an elliptic curve form an (additive) Abelian group, where \( \mathcal{O} \) is the identity element and the group operation is given by the well-known chord-and-tangent rule [46].

The order of a point \( P \in E \) is defined as the smallest non-negative integer \( n \) such that \( nP = \mathcal{O} \). For a given integer \( n \), the set \( E[n] \) of all points \( P \in E \) such that \( nP = \mathcal{O} \) is called an \( n \)-torsion group. We say that \( E[n] \) has embedding degree \( k \) if \( n \) divides \( q^k - 1 \), but does not divide \( q^i - 1 \) for any \( 0 < i < k \). In this paper we assume \( k > 1 \) and only consider curves where \( k \) is even for efficient implementation.

Tate pairing is defined in terms of divisors of a rational function. For our purpose, a divisor is a formal sum \( D = \sum_{P \in E} a_P(P) \) of points on the curve \( E(\mathbb{F}_{p^g}) \). The degree of a divisor \( D \) is the sum \( \deg(D) = \sum_{P \in E} a_P \). The set of divisors forms an Abelian group by the addition of corresponding coefficients in their formal sums. Let \( f : E(\mathbb{F}_{p^g}) \to \mathbb{F}_{p^g} \) be a rational function on the elliptic curve, then the divisor of \( f \) is \( \text{div}(f) = \sum_{P \in E} \text{ord}_P(f)(P) \), where \( \text{ord}_P(f) \) is the order of the zero or pole of \( f \) at \( P \). Let \( D \) be a divisor of degree zero, the evaluation of the rational function \( f \) at \( D \) is defined as \( f(D) = \prod_{P \in E} f(P)^{a_P} \). A divisor \( D \) is called a principal divisor if \( D = \text{div}(f) \) for some rational function \( f \). A divisor \( D \) is \( \frac{1}{2} \)-principal if and only if \( \deg(D) = 0 \) and \( \sum_{P \in E} a_P P = \mathcal{O} \). Two divisors \( D_1 \) and \( D_2 \) are equivalent (i.e., \( D_1 \sim D_2 \)) if their difference \( D_1 - D_2 \) is a principal divisor. Let \( P \in E(\mathbb{F}_q)[n] \) where \( n \) is coprime to \( q \) and \( Q \in E(\mathbb{F}_{p^g}) \). Let \( D_P \) be a divisor equivalent to \( \langle P \rangle - \langle \mathcal{O} \rangle \) and therefore there is a rational function \( f_{n,P} \in \mathbb{F}_q(E)^* \) such that \( \text{div}(f_{n,P}) = nD_P = n\langle P \rangle - n\langle \mathcal{O} \rangle \). Let \( D_Q \) be a divisor equivalent to \( \langle Q \rangle - \langle \mathcal{O} \rangle \) with its support disjoint from \( \text{div}(f_{n,P}) \). The Tate pairing [25] is a well-defined, non-degenerate, bilinear map \( e_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_{p^g})/nE(\mathbb{F}_{p^g}) \to \mathbb{F}_{q^k}/(\mathbb{F}_{q^k})^n \) given by \( e_n(P, Q) = f_{n,P}(D_Q) \). The computation of \( f_{n,P}(D_Q) \) is achieved by an application of Miller’s algorithm [41], whose output is only defined up to \( n \)-th powers in \( \mathbb{F}_{q^k} \), which is usually undesirable in practice since many pairing-based protocols require a unique pairing value. According to Theorem 1 in [6], one can define reduced Tate pairing as \( e(P, Q) = f_{n,P}(Q)^{(q^k-1)/n} \). The extra powering required to compute the reduced pairing is referred to as the final exponentiation.

A.2 Multibase Number Representation

Various multibase number representations (MSRs) have recently been proposed to accelerate elliptic curve scalar multiplication, see [20], [21], [36], [39] for example. The basic idea behind MSR is to record a scalar in a very compact and sparse form and therefore significantly reduce the number of point additions during the computation of scalar multiplication. In [39], Longa and Miri introduced the following generic multibase representation for a scalar \( n \):

\[
n = \sum_{i=1}^{m} n_i \prod_{j=1}^{J} a_j^{c_i(j)},
\]

where (1) bases \( a_1 \neq a_2 \neq \cdots \neq a_J \) are positive primes (\( a_1 \) is called the main base) and \( m \) is the length of the expansion; (2) \( n_i \) are signed digits from a given set \( \mathbb{D} \); (3) \( c_1(j) \geq c_2(j) \geq \cdots \geq c_m(j) \geq 0 \) for each \( j \) from \( 2 \) to \( J \); and (4) \( c_1(1) > c_2(1) > \cdots > c_m(1) > 0 \). Based on the above multibase representation, Longa et al. [36], [39] proposed Multibase Non-Adjacent Form (mbNAF) and its window-based variants including Window-w Multibase Non-Adjacent Form (wmbNAF) and Fractional Window-w Multibase Non-Adjacent Form (Frac-wmbNAF). Combined with optimized composite operations and precomputation schemes [38], the multibase methods have set new speed records for computing the elliptic curve scalar multiplication [36].

A.3 A Multibase Variant of Miller’s Algorithm

We propose a multibase variant of Miller’s algorithm and show how to efficiently extract the required rational functions in this case. Considering that one inversion is required for each group operation in the process of computing Tate pairings, and the calculation of the inversion of an element in large characteristic is usually quite expensive, we use Jacobian coordinates to represent points on an elliptic curve instead of affine coordinates. Moreover, we also employ the Frac-wmbNAF method [36] to record the scalar, which has been shown to achieve the highest performance among window-based methods for standard elliptic curves in Weierstrass form.
Algorithm 1. A Multibase Variant of Miller’s Algorithm for Pairing Computation

Input: $n = (n_1(q), \ldots, n_t(q), n_0(q))$, where $n_i(q) \in \mathcal{D}$ and $n_i \in \mathcal{A}$.

$P = (x_P, y_P) \in E(F_q)[n]$ and $Q = (x_Q, y_Q) \in E(F_{q^k})$.

Output: $e(P, Q) = f_{s, x}(Q)^{(q^k-1)/n}$.

[Precomputation]:

1. Compute $P_t = iP$ for $i \in \{1, 3, \ldots, t\}$, where $t \geq 3$ is an odd integer.

2. Compute $f_{t+1} = f_{\pm 1}(P, Q)$ for $i \in \{1, 3, \ldots, t\}$, where $\text{div}(f_i) = i(P) - (i(P) - (i - 1)(Q))$.

[Main Loop]:

3. $j' \leftarrow f_{\pm 1}(n_1)$; $T \leftarrow P_{n_1}$.

4. for $i \leftarrow l - 1$ downto 0 do

5. if $n_i(b_i) = 2$, then $j' \leftarrow j' - 1$; $T \leftarrow 2T$.

6. else $j' \leftarrow j' - 1$; $T \leftarrow b_iT$.

7. if $n_i(b_i) \neq 0$ then

8. if $n_i(b_i) > 0$, then $P' \leftarrow P_{n_i(b_i)}$; else $P' \leftarrow -P_{n_i(b_i)}$.

9. $f' \leftarrow f' - f_{\pm 1}(n_i(b_i)) \cdot \frac{1}{\nu_{i+1}(P)} \cdot \frac{1}{\nu_{i+1}(Q)}$; $T \leftarrow T + P'$.

10. return $f'(q^k-1)/n$.

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_j\}$ be a set of bases, where $a_1 = 2$ and $a_2 \neq a_3 \neq \cdots \neq a_j$ are positive odd primes. Let $\mathcal{D} = \{0, \pm 1, \pm 3, \ldots, \pm t\}$ be a digit set, where $t \geq 3$ is an odd integer. Let $P \in E(F_{q^k})[n]$ and $Q \in E(F_{q^k})$, where $n$ is a prime. Assume that $n$ is represented by the Frac-wnbNAF method as $(n_1(b_1), \ldots, n_t(b_t), n_0(b_0))$, where $n_i(b_i) \in \mathcal{D}$ is the $i$-th digit and the superscript $b_i \in \mathcal{A}$ denotes the base associated to the corresponding digit for $0 \leq i \leq l$. With the above notations, the multibase variant of Miller’s algorithm is described in Algorithm 1. The algorithm includes two phases: the precomputation phase that calculates the required points and evaluates the corresponding rational functions at the image point $Q$, and the main loop that uses the results of the precomputation phase to compute the Tate pairing efficiently.

In the following subsections, we will explain how to perform the precomputation and the main loop efficiently. Let $I, M$ and $S$ denote an inversion, a multiplication and a squaring in $F_{q^k}$, and let $M_k$ and $S_k$ denote a multiplication and a squaring in the large field $F_{q^k}$. We also assume that an elliptic curve $E$ over $F_q$ admits a twist $E'$ of degree $w$ with $w \mid k$. Letting $d = k/w$, then we can take $Q$ to be a point on the twist curve $E'(F_q)$, which allows us to apply the efficient denominator elimination technique due to Barreto, Lynn and Scott [7]. Using twist curves, we can work within the groups $E(F_q)[n]$ and $E'(F_{q^k})$ at all times except when a pairing is being evaluated, where we use the twist map and operate in $F_{q^k}$.

A.3.1 Precomputation Method

The precomputed points $P_i$ and function values $f_i$ are extensively used to accelerate the computation of Tate pairing in the above multibase variant of Miller’s algorithm. Longa and Miri [38] proposed a highly efficient precomputation scheme for elliptic curve cryptosystems over prime fields, which is based on the combination of the traditional chain $P \rightarrow 2P \rightarrow 3P \rightarrow 5P \rightarrow \cdots \rightarrow tP$ and the special point addition formula with the same $z$-coordinate introduced by Meloni [40]. In this subsection, we show how to efficiently obtain the function values $f_{t+1} = f_{t+1}(P, Q)$ for $i \in \{1, 3, \ldots, t\}$ and $Q \in E'(F_{q^k})$ based on the precomputation scheme in [38]. Considering that the precomputed table $\{P, 3P, \ldots, tP\}$ are stored in affine coordinates, it is not hard to find that $f_i$ can be computed more efficiently in affine coordinates instead of projective coordinates. Assuming that $jP = (x_jP, y_jP)$ for any $j \in Z^*$ and $Q = (x_Q, y_Q)$, we first have $f_1 = f_1(P, Q) = 1$ and $f_{-1} = f_{-1}(P, Q) = \frac{1}{x_Q-x_P}$. Using the parabola method proposed by Eisenträger et al. [22], we can calculate $f_{\pm 3}$ simultaneously as follows:

$$
\begin{align*}
f_{3} &= f_{s, x}(Q) = \frac{1}{\nu_{3}(P)} \cdot \frac{1}{\nu_{3}(Q)} \\
&= \frac{(x_Q-x_P)^2 + (\lambda_1 + \lambda_2)(x_Q-x_P) - (y_Q-x_P)}{x_Q-x_P} \\
&= \frac{(x_Q-x_P)^2 + \lambda_1(x_Q-x_P) - (\lambda_1 + \lambda_2)x_Q - \lambda_2y_Q}{x_Q-x_P} \\
&= \frac{(x_Q-x_P)(x_Q-x_P)^2 + (3x_P^2 + a)(x_Q-x_P) + 2y_P^2 - 2y_PY_Q}{(x_Q-x_P)(x_Q-x_P)}.
\end{align*}
$$

where $\lambda_1$ and $\lambda_2$ are the slopes of the lines $l_P, P$ and $l_2P, P$, respectively. Using the fact that $-jP = (x_jP, -y_jP)$ for any $j \in Z^*$, we can obtain $f_{-3}$ virtually for free by reusing the intermediate results during the computation of $f_3$:

$$
\begin{align*}
f_{-3} &= \frac{(x_Q-x_P)(x_Q-x_P)^2 + (3x_P^2 + a)(x_Q-x_P) + 2y_P^2 + 2y_PY_Q}{(x_Q-x_P)(x_Q-x_P)}.
\end{align*}
$$

Note that $(3x_P^2 + a)$ and $2y_P^2$ can be obtained from the computation of $2P$ and the denominator $(x_Q-x_P)(x_Q-x_3P)$ can be eliminated by the final exponentiation. Therefore, we only need to compute the numerators of $f_{\pm 3}$ and the computation cost is $dM$, with a precomputation of $1S_d + (d+k)M$.

Similarly, for an odd integer $s$ satisfying $5 \leq s \leq t$, $f_{\pm s}$ can be computed simultaneously as follows:

$$
\begin{align*}
f_{s} &= f_{s, x}(Q) = f_{s-2} \cdot \frac{1}{\nu_{s}(P)} \\
&= f_{s-2} \cdot \frac{(y_Q-y_2P) - \lambda_s(x_Q-x_2P)}{y_2P^2} \\
&= f_{s-2} \cdot \frac{(x_{s-2}P-x_2P)(y_Q-y_2P) - (y_{s-2}P-y_2P)(x_Q-x_2P)}{(x_{s-2}P-x_2P)(x_Q-x_2P)}.
\end{align*}
$$

and

$$
\begin{align*}
f_{-s} &= f_{-s, x}(Q) = f_{-(s-2)} \cdot \frac{(x_{s-2}P-x_2P)(y_Q+y_2P) + (y_{s-2}P-y_2P)(x_Q-x_2P)}{(x_{s-2}P-x_2P)(x_Q-x_2P)}.
\end{align*}
$$
where \( \lambda \) is the slope of the line \( l_{2P(s-2)p} \). Hence, given 
\( f_{x,(s-2)p} \), we need \( 2Mk + (k + d + 1)M \) to calculate the numerators of \( f_{x,s} \) simultaneously.

We summarize the precomputation scheme in Algorithm 2. Again, we only consider the computations of the numerators of \( f_{x,t} \) due to the efficient denominator elimination technique [7]. Note that \( F_{t,i}, i \in \{1, 3, \ldots, t\} \) can be calculated with \( 11 + 3(t-1)M + (t+5)S \) using the precomputation method (Scheme 2) in [38]. Moreover, we also need to cost another \( 4M \) to recover the affine coordinates of \( 2P \) that are used in the computation of \( f_{x,t} \). Therefore, the total cost of the precomputation scheme in Algorithm 2 is \((t-3)M_k + 1M_d + 11 + (\frac{(5t-2) + (k+d)(t-1)}{2})M + (t+5)S \).

### A.3.2 Encapsulated Composite Operations and Line Computations

In [17], Chatterjee et al. introduced the idea of encapsulating elliptic curve group operations (i.e., point addition and point doubling) and line computations, and applied this idea to improve the computation of Tate pairing for two families of pairing-friendly curves with embedding degree 2. It is also straightforward to adapt this idea to other pairing-friendly curves which admit twists of degree \( w \) with \( w \mid k \). Moreover, fast doubling and mixed-addition formulae proposed in [37] can be used to further improve Chatterjee et al.’s algorithms. In this subsection, we show how to efficiently perform the main loop in the multibase variant of Miller’s algorithm by encapsulating fast composite operations and line computations.

**Encapsulated Point Mixed-Addition and Line Computation:**
Let \( T = (X_1, Y_1, Z_1) \) and \( P' = (x_2, y_2) \) be two points in Jacobian and affine coordinates, respectively, on the elliptic curve \( E(F_p) \). The mixed addition \( T + P' = (X_3, Y_3, Z_3) \) can be computed efficiently as follows [37]:

\[
\begin{align*}
X_3 &= \alpha^2 - 4\beta^3 - 8X_1\beta^2, \\
Y_3 &= \alpha(4X_1\beta^2 - X_3 - 8Y_1\beta^3), \\
Z_3 &= (Z_1 + \beta)^2 - Z_1^2 - \beta^2, \\
\end{align*}
\]

where \( \alpha = 2(Z_1^2)y_2 - Y_1 \) and \( \beta = Z_1^2x_2 - X_1 \). In this case, the evaluation of the line function \( l_{T,T'} \) at the image point \( Q \) can be calculated as follows:

\[
l_{T, P}(Q) = (y_Q - y_2) - 2(Z_1^2y_2 - Y_1) \cdot (x_Q - x_2) \\
Z_3y_Q - ax_Q + (ax_2 - Z_3y_2) \\
Z_3 \\
\]

where \( Z_3 \) and \( \alpha \) are available from the point mixed-addition. The encapsulated point mixed-addition and line computation are given in Algorithm 3, which requires \((9 + d + k)M + 4S \). Note that in the case of \( d = 1 \) (i.e., \( x_Q \in \mathbb{F}_q \)) we can save one more field multiplication by combining the terms involving \( \alpha \) in the above equation (1).

**Encapsulated Point Doubling and Line Computation:**
Let \( T = (X_1, Y_1, Z_1) \) be a point in Jacobian coordinates on the elliptic curve \( E(F_p) \). The point doubling \( 2T = (X_3, Y_3, Z_3) \) can be computed efficiently as follows [37]:

\[
\begin{align*}
X_3 &= \alpha^2 - 2\beta, \\
Y_3 &= \alpha(\beta - X_3) - 8Y_1^4, \\
Z_3 &= (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2, \\
\end{align*}
\]

where \( \alpha = 3X_1^2 + aZ_1^4 \) and \( \beta = 2[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4] \) for a general \( a \), and \( \alpha = 3X_1 + Z_1^2(X_1 - Z_1^2) \) and \( \beta = 4X_1Y_1^2 \) if \( a = -3 \). In this case, the evaluation of the line function \( l_{T,T} \) at the image point \( Q \) can be calculated as follows:

\[
l_{T,T}(Q) = \frac{(y_Q - Y_1) - 3X_1^2 + aZ_1^4}{Z_3} \cdot (x_Q - X_1) \\
= \frac{Z_3Z_1^2y_Q - (4Y_1^2x_1 - ax_1) + (aZ_1^2)\alpha}{Z_3^2Z_1^2}.
\]

where \( Z_3, Z_1^2, Y_1^2 \) and \( \alpha \) are available from the point doubling. The encapsulated point doubling and line computation are given in Algorithm 4, where some intermediate results of the point doubling, as shown in the boxes, can be reused for the line computation. Algorithm 4 requires \((5 + d + k)M + 8S \) to compute the point doubling and evaluate the line function. In the case \( a \) is small, this cost is \((4 + d + k)M + 8S \) and for the case \( a = -3 \), this cost is \((6 + d + k)M + 5S \). Note that in the case of \( d = 1 \) (i.e., \( x_Q \in \mathbb{F}_q \)) we can save two more field multiplications by combining the terms involving \( \alpha \) in the above equation (2).

**Encapsulated Point Tripling and Line Computation:**
Let \( T = (X_1, Y_1, Z_1) \) be a point in Jacobian coordinates on the elliptic curve \( E(F_p) \). The point tripling \( 3T = (X_3, Y_3, Z_3) \) can be computed efficiently as follows [37]:

\[
\begin{align*}
X_3 &= 16Y_1^2(2\beta - 2a) + 4X_1\omega^2, \\
Y_3 &= 8Y_1[(2a - 2\beta)(4\beta - 2a) - \omega^3], \\
Z_3 &= (Z_1 + \omega)^2 - Z_1^2 - \omega^2, \\
\end{align*}
\]

where \( 2a = (\theta + \omega)^2 - \theta^2 - \omega^2, 2\beta = 16Y_1^2, \theta = 3X_1^2 + aZ_1^4 \) and \( \omega = 6[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4] - \theta^2 \) for a general \( a \). For the case when \( a = -3, \theta \) and \( \omega \) can be computed more efficiently as \( \theta = 3(X_1 + Z_1^2)(X_1 - Z_1^2) \) and \( \omega = 12X_1Y_1^2 - \theta^2 \). Similar to the precomputation procedure, we can apply the parabola method [22] again to simplify the required line function \( f_{3,T} \). However, in the main loop of Algorithm 2 the points \( T \) and \( Q \) are represented in Jacobian and affine coordinates, respectively. Letting \( \lambda_1 \) and \( \lambda_2 \) be the slopes of the lines \( l_{T,T} \) and \( l_{2T,T} \), we can obtain \( \lambda_1' = \frac{x_1}{y_1}, Z_1' + \lambda_2' = \frac{y_1^2}{Z_1} \). In this case, the evaluation of the line function \( f_{3,T} \) at the image point \( Q \) can be calculated as follows:

\[
l_{3, T}(Q) = \frac{f_{T,T}(Q)}{f_{2T,T}(Q)} \cdot \frac{l_{2T,T}(Q)}{l_{T,T}(Q)} \\
= \frac{\left(\frac{y_Q - x_1}{z_1^2} + \lambda_1' + \lambda_2' \right) \left(\frac{y_Q - x_1}{z_1^2} - \frac{\lambda_1^2 \left(\frac{y_Q - x_1}{z_1^2}\right) - \lambda_2'^2}{\lambda_2'}\right)\lambda_2'}{\lambda_1^2 \left(\frac{y_Q - x_1}{z_1^2}\right) - \lambda_2'^2} \\
= \frac{\left(\frac{y_Q - x_1}{z_1^2} - \frac{\lambda_1^2 \left(\frac{y_Q - x_1}{z_1^2}\right) - \lambda_2'^2}{\lambda_2'}\right)\lambda_2'}{\lambda_1^2 \left(\frac{y_Q - x_1}{z_1^2}\right) - \lambda_2'^2} \\
= \frac{\left(\frac{y_Q - x_1}{z_1^2} - \frac{\lambda_1^2 \left(\frac{y_Q - x_1}{z_1^2}\right) - \lambda_2'^2}{\lambda_2'}\right)\lambda_2'}{\lambda_1^2 \left(\frac{y_Q - x_1}{z_1^2}\right) - \lambda_2'^2}.
\]
Algorithm 2. Precomputation Scheme for the Multibase Variant of Miller’s Algorithm

Input: \( P = (x_P, y_P) \in E(F_q)[n] \) and \( Q = (x_Q, y_Q) \in E(F_{q^k}) \).

Output: \( P_i = iP \) and the numerators \( f_{i,j} \) of \( f_{i,j} = iP + jQ \) for \( i \in \{1, 3, \ldots, t\} \),

where \( t \geq 3 \) is an odd integer and \( \text{div}(f_j) = (iP) - ((i-1)Q) \).

1. Compute \( P_i = iP = (x_{iP}, y_{iP}) \) for \( i \in \{1, 3, \ldots, t\} \) and \( 2P = (x_{2P}, y_{2P}) \) using the precomputation method proposed in [38] (see pp. 238-240 of [38]).

2. \( f_1 \leftarrow 1, f_{t,1} \leftarrow 1 \)
3. \( T_1 \leftarrow -2(x_P^2 + ay_P + a)(x_Q - x_P), T_2 \leftarrow 2y_Q Q \)
4. \( f'_2 \leftarrow T_1 + T_2, f'_{t-2} \leftarrow T_1 - T_2 \)
5. \( T_1 \leftarrow x_Q - x_P, T_3 \leftarrow T_1 y_Q T_3 \leftarrow T_1 y_Q + (y_2 - y_P)(x_Q - x_P) \)
6. \( f'_i \leftarrow f'_i - f'_{i-2} (T_2 - T_3), f'_3 \leftarrow f'_{t-2} - f'_{(t-3) (T_2 - T_3)} \)
7. \( s \leftarrow s + 2 \)
8. return \( P_i \) and \( f'_{i,1} \) for \( i \in \{1, 3, \ldots, t\} \)

Algorithm 3. Encapsulated Point Mixed-Addition and Line Computation

Input: \( T = (X_1, Y_1, Z_1) \) in Jacobian coordinates, \( P_T = (x_2, y_2) \) in affine coordinates on \( E(F_p) \) and \( Q = (x_Q, y_Q) \in E(F_{q^k}) \).

Output: \( T + P_T = (X_3, Y_3, Z_3) \) in Jacobian and the numerator \( l_{ADD} \) of \( l_{T, P_T} \).

Point Mixed-Addition

1. \( T_1 \leftarrow Z_1^2 \)
2. \( T_2 \leftarrow x_2 \cdot T_1 \)
3. \( T_3 \leftarrow x_2 \cdot T_1 - X_1 \)
4. \( T_4 \leftarrow T_2^2 \)
5. \( T_5 \leftarrow Z_1 + T_2 \)
6. \( Z_3 \leftarrow Z_1 + T_2 \)
7. \( X_3 \leftarrow T_1^2 \)
8. \( T_3 \leftarrow 8 \cdot T_1 \)
9. \( T_4 \leftarrow T_2 \cdot T_3 \)
10. \( T_5 \leftarrow T_1 - Y_1 \)
11. \( T_6 \leftarrow T_2 - T_4 \)
12. \( T_7 \leftarrow T_1 - T_4 \)
13. \( T_8 \leftarrow T_2 - T_4 \)
14. \( T_9 \leftarrow T_1 - T_4 \)

Line Computation

1. \( T_2 \leftarrow x_2 \cdot T_1 \)
2. \( T_6 \leftarrow y_Q \cdot Z_1 \)
3. \( T_5 \leftarrow Z_1 + T_2 \)
4. \( T_6 \leftarrow y_Q \cdot T_1 \)
5. \( T_7 \leftarrow T_2 - T_3 \)
6. \( T_7 \leftarrow T_2 - T_3 \)
7. \( l_{ADD} \leftarrow T_7 + T_2 \)
8. \( l_{ADD} \leftarrow T_7 + T_2 \)

is small, this cost is \( (12 + 2d + k)M + 11S \) and for a general \( a \), this cost is \( (13 + 2d + k)M + 11S \). Note that in the case of \( d = 1 \) or 2 (i.e., \( x_Q \in F_p \) or \( F_{q^2} \)) it is more efficient to evaluate the line functions using equation (3) instead of (4), which can save \( 3M + 1S \) and \( 1M + 1S \), respectively.

REFERENCES

Algorithm 4. Encapsulated Point Doubling and Line Computation

Input: \( T = (X_1, Y_1, Z_1) \) in Jacobian coordinates on \( E(F_p) \) and \( Q = (x_Q, y_Q) \in E(F_{p^k}) \).

Output: \( 2T = (X_3, Y_3, Z_3) \) in Jacobian coordinates and the numerator \( l_{DBL} \) of \( l_T, T(Q) \).

<table>
<thead>
<tr>
<th>( a = -3 )</th>
<th>( a ) General ( a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( T_1 \leftarrow Z_1^2 )</td>
<td>1. ( T_1 \leftarrow Z_1^2 )</td>
</tr>
<tr>
<td>2. ( T_2 \leftarrow X_1 + T_1 )</td>
<td>12. ( T_4 \leftarrow 4 \cdot T_4 )</td>
</tr>
<tr>
<td>3. ( T_3 \leftarrow X_1 - T_1 )</td>
<td>13. ( X_3 \leftarrow T_3^2 )</td>
</tr>
<tr>
<td>4. ( T_4 \leftarrow T_2 + T_3 )</td>
<td>14. ( X_3 \leftarrow T_4 - T_3 )</td>
</tr>
<tr>
<td>5. ( T_5 \leftarrow 3 \cdot T_2 )</td>
<td>15. ( X_3 \leftarrow X_3 - T_4 )</td>
</tr>
<tr>
<td>6. ( T_6 \leftarrow 3 \cdot T_3 )</td>
<td>16. ( T_6 \leftarrow T_5 - 3 \cdot T_6 )</td>
</tr>
<tr>
<td>7. ( T_7 \leftarrow Y_1 + T_1 )</td>
<td>17. ( T_6 \leftarrow T_6 + T_7 )</td>
</tr>
<tr>
<td>8. ( T_8 \leftarrow Z_3 )</td>
<td>18. ( T_7 \leftarrow 4 \cdot T_7 )</td>
</tr>
<tr>
<td>9. ( T_9 \leftarrow Z_3 )</td>
<td>19. ( T_8 \leftarrow 3 \cdot T_8 )</td>
</tr>
<tr>
<td>10. ( T_{10} \leftarrow Z_3 - T_3 )</td>
<td>20. ( T_9 \leftarrow 8 \cdot T_9 )</td>
</tr>
<tr>
<td>( )</td>
<td>21. ( T_10 \leftarrow T_1 )</td>
</tr>
<tr>
<td>( )</td>
<td>22. ( T_1 \leftarrow X_1 - T_3 )</td>
</tr>
<tr>
<td>( )</td>
<td>23. ( T_2 \leftarrow X_1 + T_3 )</td>
</tr>
<tr>
<td>( )</td>
<td>24. ( T_3 \leftarrow T_3 + T_3 )</td>
</tr>
</tbody>
</table>

\( l_{DBL} \leftarrow T_k - T_3 \)

\[ \text{Return } (X_3, Y_3, Z_3) \text{ and } l_{DBL} \]

Algorithm 5. Encapsulated Point Tripling and Line Computation (\( a = -3 \))

Input: \( T = (X_1, Y_1, Z_1) \) in Jacobian coordinates on \( E(F_p) \) and \( Q = (x_Q, y_Q) \) in \( E(F_{p^k}) \).

Output: \( 3T = (X_3, Y_3, Z_3) \) in Jacobian coordinates and the numerator \( l_{TRPL} \) of \( l_T, 3T(Q) \).

<table>
<thead>
<tr>
<th>( T_1 \leftarrow Z_1^2 )</th>
<th>( T_2 \leftarrow X_1 + T_1 )</th>
<th>( T_3 \leftarrow X_1 - T_1 )</th>
<th>( T_4 \leftarrow 3 \cdot T_4 )</th>
<th>( T_5 \leftarrow z_3 - T_1 )</th>
<th>( T_6 \leftarrow 3 \cdot T_6 )</th>
<th>( T_7 \leftarrow T_7 )</th>
<th>( T_8 \leftarrow T_8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_9 \leftarrow Y_1 )</td>
<td>( T_10 \leftarrow T_3 - T_2 )</td>
<td>( T_11 \leftarrow T_1 - T_2 )</td>
<td>( T_12 \leftarrow T_2 + T_3 )</td>
<td>( T_13 \leftarrow T_3 - T_2 )</td>
<td>( T_14 \leftarrow 2 \cdot T_2 )</td>
<td>( T_15 \leftarrow 3 \cdot T_3 )</td>
<td>( T_16 \leftarrow T_2 )</td>
</tr>
<tr>
<td>( T_17 \leftarrow Z_3 - T_1 )</td>
<td>( T_18 \leftarrow T_7 - T_7 )</td>
<td>( T_19 \leftarrow T_9 )</td>
<td>( T_20 \leftarrow T_3 - T_2 )</td>
<td>( T_21 \leftarrow 2 \cdot T_1 )</td>
<td>( T_22 \leftarrow T_5 - T_5 )</td>
<td>( T_23 \leftarrow T_7 - T_3 )</td>
<td>( T_24 \leftarrow T_3 )</td>
</tr>
<tr>
<td>( T_25 \leftarrow T_8 )</td>
<td>( T_26 \leftarrow T_9 )</td>
<td>( T_27 \leftarrow T_6 )</td>
<td>( T_28 \leftarrow T_8 )</td>
<td>( T_29 \leftarrow T_9 )</td>
<td>( T_30 \leftarrow T_8 )</td>
<td>( T_31 \leftarrow T_9 )</td>
<td>( T_32 \leftarrow T_9 )</td>
</tr>
</tbody>
</table>

\( l_{TRPL} \leftarrow T_d - T_d \)

\[ \text{Return } (X_3, Y_3, Z_3) \text{ and } l_{TRPL} \]


consumer-industrial-public-services-(1).


