Roll-DPoS: 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 [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 tolerate 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 [3], [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 [33] sheds some light on blockchain based IoT applications in practice. In DPoS, 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 DPoS 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 DPoS 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 DPoS 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 $x_i = f(\mathsf{id}_i) \pmod q$ 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[1] is able to recover the secret using the Lagrange interpolation formula:

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

where $\lambda_i(z) = \prod_{j=1, j\neq i}^{t+1} \frac{z-\mathsf{id}_j}{\mathsf{id}_i - \mathsf{id}_j} \pmod q$. Note that the secret x can also be written as $x = f(0) = \sum_{i=1}^{t+1} x_i \lambda_i(0) \pmod q$. 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., $q | p-1$). 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 $W_i = g^{a_i} \pmod p$ 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 $g^{x_i} = \prod_{j=0}^{t} W_j^{\mathsf{id}_i^j} \pmod p$.

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 $f_i(0) = a_{i0} = \bar{x}_i$ is a random secret that P_i selects. Each P_i computes the shares $\bar{x}_{ij} = f_i(\mathsf{Id}_j) \pmod q$ 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 $W_{ik} = g^{a_{ik}} \pmod p$ for $k \in [0, t]$ and broadcasts them into the system. Denote W_{i0} by $\bar{y}_i$.
- Each P_j verifies the shares received from other entities by checking the equation for $i \in [1, n]$:

  $$g^{x_{ij}} = \prod_{k=0}^{t} W_{ik}^{\mathsf{id}_j^k} \pmod p.$$

  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 compliant with a false value, and then creates a set QUAL which contains all the qualified entities.
- Each P_i computes its secret share as $x_i' = \sum_{j \in \mathsf{QUAL}} x_{ij}$ and the system private key x, which is not known to any entity, is equal to $\sum_{i \in \mathsf{QUAL}} \bar{x}_i \pmod q$.

- For each disqualified entity P_i, any set of $t+1$ entities can recover its corresponding secret a_{i0} and compute $\bar{y}_i = g^{a_{i0}} \pmod p$. For other entities, we have $\bar{y}_i = W_{i0}$. The system public key is then computed as $y = \prod_{i \in \mathsf{QUAL}} \bar{y}_i \pmod p$. Note that $y = g^x$.

In Pedersen's DKG scheme, the private key x is uniquely defined at the end of the protocol and no coalition of up to $t < n/2$ adversaries can prevent its recovery. Although Gennaro *et al.* [31] pointed out that an adversary can control to some extent the distribution of public keys generated by the Pedersen's DKG scheme, the bias generally does not weaken the hardness of solving the discrete logarithm problem for the generated public key [27], [28].

2.4 Bilinear Pairing

Let N be a positive integer and $\mathbb{G}_1$ and $\mathbb{G}_2$ be additively-written groups of order N with identity $\mathcal{O}$, and let $\mathbb{G}_T$ be a multiplicatively-written group of order N with identity 1. A *bilinear pairing* – or *pairing* for short – is a computable, non-degenerate function:

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T,$$

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 $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_N$. In practice, the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups or quotient groups of an elliptic curve defined over a finite field $\mathbb{F}_q$ or one of its extensions and $\mathbb{G}_T$ is a subgroup or quotient group $\mathbb{F}_{q^k}$, where k is called the *embedded degree*. The security of pairing-based cryptography requires that the discrete logarithm problem on $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ 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 $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, respectively, and $H_1 : \{0,1\}^* \mapsto \mathbb{G}_1$ be a hash function with values in $\mathbb{G}_1$. The BLS signature scheme consists of the following three algorithms:

- **KeyGen**(): Choose a random $\alpha \overset{R}{\longleftarrow} \mathbb{Z}_N$ and set $h \leftarrow g_2^{\alpha} \in \mathbb{G}_2$ and output a private/public key pair $(sk, pk) = (\alpha, h)$.
- **Sign**(sk, m): Given a private key sk and a message $m \in \{0,1\}^*$, output $\sigma \leftarrow H_1(m)^{sk} \in \mathbb{G}_1$. Note that the signature σ is a single group element.
- **Verify**(pk, m, σ): Given a public key pk, a message m, and a signature σ, check whether $e(\sigma, g_2) = e(H_1(m), pk)$ and output "accept" or "reject" accordingly.

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 $\alpha \overset{R}{\longleftarrow} \mathbb{Z}_N$ and set $h \leftarrow g_2^{\alpha} \in \mathbb{G}_2$. 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 $sk_i = f(\mathsf{Id}_i)$ 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 $\sigma_i \leftarrow H_1(m)^{sk} \in \mathbb{G}_1$.
- **SignShareVerify**(pk_i, m, σ_i): Given a public key share pk_i, a message m, and a signature share σ_i, check whether $e(\sigma_i, g_2) = e(H_1(m), pk_i)$. If the verification is successful, σ_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 $\sigma_{i_1}, \ldots, \sigma_{i_{t+1}}, \{i_1, \ldots, i_{t+1}\} \subset \{1, \ldots, n\}$, output the signature $\sigma = \prod_{j=1}^{t+1} \sigma_{i_j}^{\lambda_{i_j}}$, where $\lambda_{i_k} = \prod_{j=1, j\neq k}^{t+1} \frac{0 - \mathbf{Id}_{i_k}}{\mathbf{Id}_{i_j} - \mathbf{Id}_{i_k}}, k = 1, \ldots, t+1$ are Lagrange coefficients.
- **Verify**(pk, m, σ): Given a public key pk, a message m, and a combined signature σ, 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

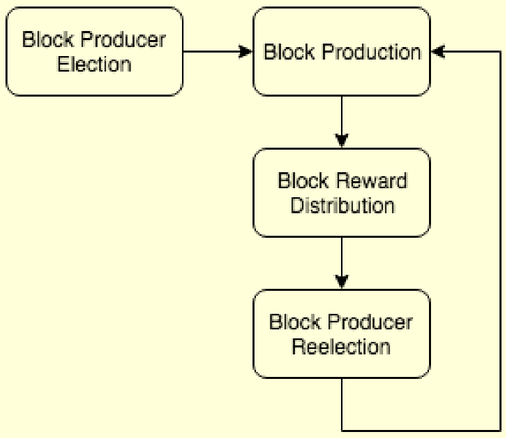

Fig. 1. The Delegated Proof of Stake (DPoS) Framework

to produce blocks. The number of votes are proportional to the voter's stake and a fixed number of top candidates that receive the most votes become the block producers.

2) **Block Production**. The block production happens in rounds and is conducted in a round-robin manner among block producers. At the beginning of each round, the block producers are shuffled and assigned a time slot in which they should produce a block. In the case that a block producer fails to create a block during its time slot, the block is skipped and the transactions within are included in the next one.
3) **Block Reward Distribution**. The block producer receives a block reward for successfully producing a block and spreads the (partial) block reward to its voters as dividends.
4) **Block Producer Reelection**. A reelection process is performed at the end of each round and block producers can be voted in or out each round by the cryptocurrency token holders.

It is not difficult to find out that the DPoS consensus framework is essentially a liquid, representative democracy with token holder suffrage [47], which enables it to work well naturally in a consortium-like setting where a certain degree of trust has been placed in a small group of entities. For DPoS based public blockchains, the viability of a system depends on whether byzantine producers can be promptly removed through a decentralized voting process.

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 throughout, 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.

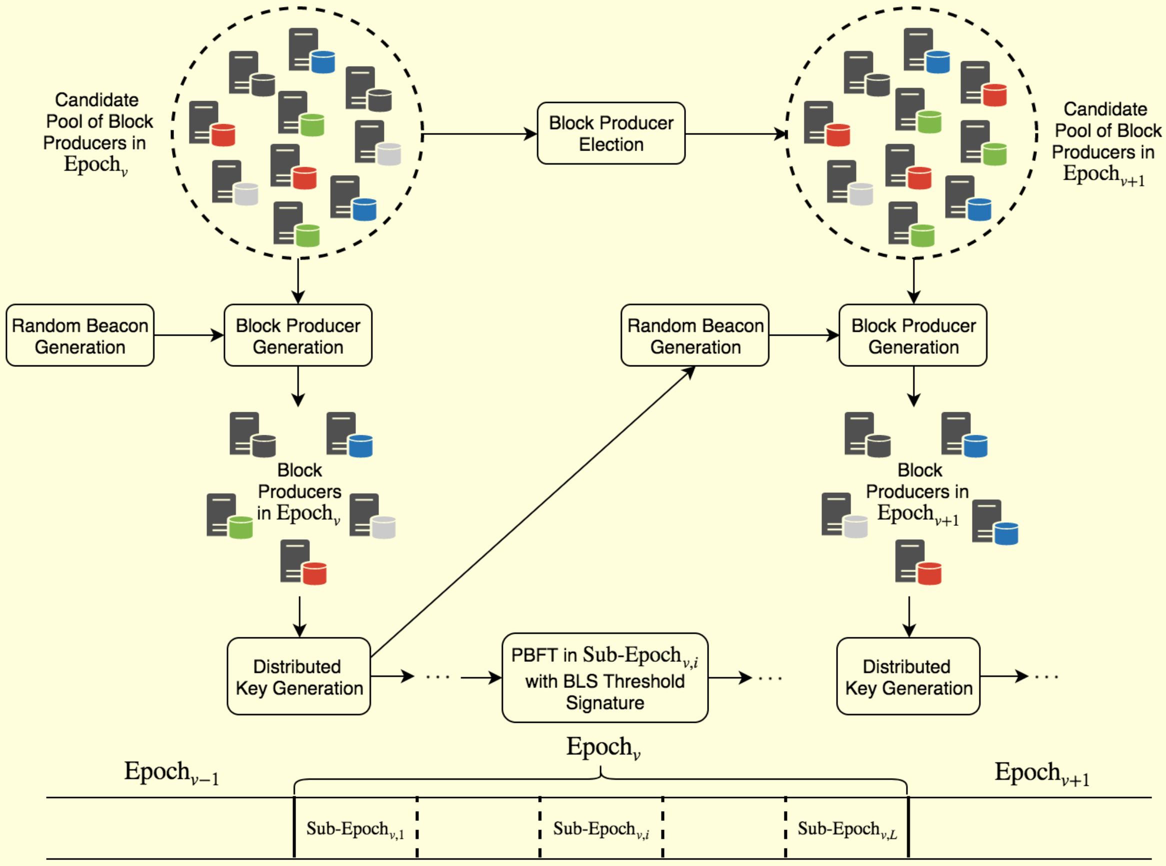
Candidate Pool of Block Producers in Epoch$_v$
Block Producer Election
Candidate Pool of Block Producers in Epoch$_{v+1}$
Random Beacon Generation
Block Producer Generation
Random Beacon Generation
Block Producer Generation
Block Producers in Epoch$_v$
Block Producers in Epoch$_{v+1}$
Distributed Key Generation
PBFT in Sub-Epoch$_{v,i}$ with BLS Threshold Signature
Distributed Key Generation
Epoch$_v$
Epoch$_{v-1}$
Epoch$_{v+1}$
Sub-Epoch$_{v,1}$
Sub-Epoch$_{v,i}$
Sub-Epoch$_{v,L}$

Fig. 2. A High-Level Overview of the Roll-DPoS Consensus Algorithm

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 $\{\mathsf{C}_1^{(1)}, \ldots, \mathsf{C}_N^{(1)}\}$, 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 $\{\mathsf{BP}_1^{(1)}, \ldots, \mathsf{BP}_n^{(1)}\}$, 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

$$o_i^{(1)} = \mathbf{DRBG}(s_0, pk_i^{(1)}, 1), \quad i = 1, \ldots, N,$$

where $pk_i^{(1)}$ is the public key of the candidate $C_i^{(1)}$ and '1' is the epoch number of the first epoch, followed by the sorting of all $o_i^{(1)}$'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-bit[2] 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 **SignShareGen**$(\cdot)$ algorithm to sign a PREPARE message with its own private key share sk_i, 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 **SignShareCombine**$(\cdot)$ and **Verify**$(\cdot)$ 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 j^{th} epoch, each candidate computes

$$o_i^{(j)} = \mathbf{DRBG}(s_{j-1}, pk_i^{(j)}, j), \quad i = 1, \ldots, N,$$

where s_{j-1} is the random beacon generated by the block producers during the $(j-1)^{\text{th}}$ 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 **DIFINITY** [29]. Besides proposing blocks in the j^{th} epoch, a block producer i also uses the **SignShareGen**$(\cdot)$ algorithm to generate a random beacon share $s_j^{(i)}$ as follows:

$$s_j^{(i)} = \mathbf{SignShareGen}(sk_i^{(j)}, s_{j-1} \| j),$$

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 **SignShareCombine**$(\cdot)$ and **Verify**$(\cdot)$ 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:

$$\begin{aligned} p &= \texttt{7ddca613 a2e3ddb1 749d0195} \\ &\quad\ \texttt{bb9f14cf 44626303 (159-bit)} \\ b &= \texttt{21c3f3ac 7864d1f1 f99273d0} \\ &\quad\ \texttt{f828d365 7d8cfd4e (158-bit)} \\ n &= \texttt{3eee5309 d171eed8 ba4e12de} \\ &\quad\ \texttt{f44414fd 17d369b7 (158-bit)} \end{aligned}$$

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 $z = \texttt{dbd7d316ead514bb8f95}$ (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 $E(\mathbb{F}_{p^6})$. 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 $E(\mathbb{F}_{p^6})$ to a point in a quadratic twist $E'(\mathbb{F}_{p^3})$. Let $D \in \mathbb{F}_{p^3}$ be a quadratic non-residue over $\mathbb{F}_{p^3}$. A quadratic twist of E, denoted by E', over $\mathbb{F}_{p^3}$ for which $n \mid \#E'(\mathbb{F}_{p^3})$ is defined by the following equation [30]

$$E'/\mathbb{F}_{p^3} : Dy^2 = x^3 - 3x + b.$$

The injective group homomorphism $\phi : E'(\mathbb{F}_{p^3}) \to E(\mathbb{F}_{p^6})$ is given by $(x, y) \mapsto (x, \sqrt{D}y)$, which allows us to map points in the quadratic twist $E'(\mathbb{F}_{p^3})$ to points in $E(\mathbb{F}_{p^6})$.

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 optimizations 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 $q = p^\nu$ elements, where $p > 3$ is a prime and ν 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 $4a^3 + 27b^2 \in \mathbb{F}_q^*$, together with an additional *point at infinity*, denoted by $\mathcal{O}$. Note that the same equation also defines curves over $\mathbb{F}_{q^k}$ for $k > 0$. Let $\#E(\mathbb{F}_{q^k})$ denote the number of points on an elliptic curve $E(\mathbb{F}_{q^k})$, which is called the *order* of the curve over the field $\mathbb{F}_{q^k}$. 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 $D = \sum_{P \in E} a_P \langle P \rangle$ of points on the curve $E(\mathbb{F}_{q^k})$. 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 $f : E(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}$ be a rational function on the elliptic curve, then the divisor of f is $\operatorname{div}(f) \equiv \sum_{P \in E} \operatorname{ord}_P(f)\langle P \rangle$, where $\operatorname{ord}_P(f)$ 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 $f(D) \equiv \prod_{P \in E} f(P)^{a_P}$. A divisor D is called a *principal* divisor if $D = \operatorname{div}(f)$ for some rational function f. A divisor $D = \sum_{P \in E} a_P \langle P \rangle$ is 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 $Q \in E(\mathbb{F}_{q^k})$. 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 $\operatorname{div}(f_{n,P}) = nD_P = n\langle P \rangle - n\langle \mathcal{O} \rangle$. Let D_Q be a divisor equivalent to $\langle Q \rangle - \langle \mathcal{O} \rangle$ with its support disjoint from $\operatorname{div}(f_{n,P})$. The *Tate pairing* [25] is a well defined, non-degenerate, bilinear map $e_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_{q^k})/nE(\mathbb{F}_{q^k}) \to \mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$ 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 $\mathcal{D}$; (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: $n = (n_l^{(b_l)}, \cdots, n_2^{(b_2)}, n_0^{(b_0)})$, where $n_i^{(b_i)} \in \mathcal{D}$ and $b_i \in \mathcal{A}$.
       $P = (x_P, y_P) \in E(\mathbb{F}_q)[n]$ and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{q^k})$.
Output: $e(P,Q) = f_{n,P}(Q)^{(q^k-1)/n}$.
[Precomputation]:
1. Compute $P_i = iP$ for $i \in \{1, 3, \cdots, t\}$, where $t \geq 3$ is an odd integer
2. Compute $f_{\pm i} = f_{\pm i,P}(Q)$ for $i \in \{1, 3, \cdots, t\}$, where $\mathrm{div}(f_i) = i\langle P\rangle - \langle iP\rangle - (i-1)\langle\mathcal{O}\rangle$
[Main Loop]:
3. $f' \leftarrow f_{n_l^{(b_l)}}, T \leftarrow P_{n_l^{(b_l)}}$
4. for $i \leftarrow l-1$ downto 0 do
5.     if $b_i = 2$, then $f' \leftarrow f'^2 \cdot \frac{l_{T,T}(Q)}{v_{2T}(Q)}, T \leftarrow 2T$
6.     else $f' \leftarrow f'^{b_i} \cdot \frac{l_{T,T}(Q)}{v_{2T}(Q)} \cdot \frac{l_{2T,T}(Q)}{v_{3T}(Q)} \cdot \frac{l_{2T,3T}(Q)}{v_{5T}(Q)} \cdots \frac{l_{2T,(b_i-2)T}(Q)}{v_{b_iT}(Q)}, T \leftarrow b_iT$
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.     $f' \leftarrow f' \cdot f_{n_i^{(b_i)}} \cdot \frac{l_{T,P'}(Q)}{v_{T+P'}(Q)}, T \leftarrow T + P'$
10. return $f'^{(q^k-1)/n}$
```

Let $\mathcal{A} = \{a_1, a_2, \cdots, a_J\}$ 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 $\mathcal{D} = \{0, \pm 1, \pm 3, \cdots, \pm t\}$ be a digit set, where $t \geq 3$ is an odd integer. Let $P \in E(\mathbb{F}_q)[n]$ and $Q \in E(\mathbb{F}_{q^k})$, where n is a prime. Assume that n is represented by the Frac-wmbNAF method as $(n_l^{(b_l)}, \cdots, n_2^{(b_2)}, n_0^{(b_0)})$, 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 $\mathbb{F}_q$, and let M_k and S_k denote a multiplication and a squaring in the large field $\mathbb{F}_{q^k}$. We also assume that an elliptic curve E over $\mathbb{F}_q$ 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 $E'(\mathbb{F}_{q^d})$ in this case, 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 $E(\mathbb{F}_q)[n]$ and $E'(\mathbb{F}_{q^d})$ at all times except when a pairing is being evaluated, where we use the twist map and operate in $\mathbb{F}_{q^k}$.

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 $P \to 2P \to 3P \to 5P \to \cdots \to tP$ 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 $f_{\pm i} = f_{\pm i,P}(Q)$ for $i \in \{1, 3, \cdots, t\}$ and $Q \in E'(\mathbb{F}_{q^d})$ based on the precomputation scheme in [38]. Considering that the precomputed table $\{P, 3P, \cdots, tP\}$ 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 $j \in \mathbb{Z}^*$ 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:

$$\begin{aligned} f_3 = f_{3,P}(Q) &= \frac{l_{P,P}(Q)}{v_{2P}(Q)} \cdot \frac{l_{2P,P}(Q)}{v_{3P}(Q)} \\ &= \frac{(x_Q - x_P)^2 + (\lambda_1 + \lambda_2)\left[\lambda_1(x_Q - x_P) - (y_Q - y_P)\right]}{x_Q - x_{3P}} \\ &= \frac{(x_Q - x_P)^2 + \lambda_1(\lambda_1 + \lambda_2)(x_Q - x_P) - (\lambda_1 + \lambda_2)(y_Q - y_P)}{x_Q - x_{3P}} \\ &= \frac{(x_P - x_{2P})(x_Q - x_P)^2 + (3x_P^2 + a)(x_Q - x_P) + 2y_P^2 - 2y_P y_Q}{(x_P - x_{2P})(x_Q - x_{3P})}, \end{aligned}$$

where λ_1 and λ_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 $j \in \mathbb{Z}^*$, we can obtain f_{-3} virtually for free by reusing the intermediate results during the computation of f_3:

$$f_{-3} = \frac{(x_P - x_{2P})(x_Q - x_P)^2 + (3x_P^2 + a)(x_Q - x_P) + 2y_P^2 + 2y_P y_Q}{(x_P - x_{2P})(x_Q - x_{3P})}.$$

Note that $(3x_P^2 + a)$ and $2y_P^2$ can be obtained from the computation of $2P$ and the denominator $(x_P - x_{2P})(x_Q - x_{3P})$ 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:

$$\begin{aligned} f_s = f_{s,P}(Q) &= f_2 \cdot f_{s-2} \cdot \frac{l_{2P,(s-2)P}(Q)}{v_{sP}(Q)} \\ &= f_2 \cdot f_{s-2} \cdot \frac{(y_Q - y_{2P}) - \lambda_s(x_Q - x_{2P})}{v_{sP}(Q)} \\ &= f_2 \cdot f_{s-2} \cdot \frac{(x_{(s-2)P} - x_{2P})(y_Q - y_{2P}) - (y_{(s-2)P} - y_{2P})(x_Q - x_{2P})}{(x_{(s-2)P} - x_{2P})(x_Q - x_{sP})}, \end{aligned}$$

and

$$f_{-s} = f_{-2} \cdot f_{-(s-2)} \cdot \frac{(x_{(s-2)P} - x_{2P})(y_Q + y_{2P}) + (y_{(s-2)P} - y_{2P})(x_Q - x_{2P})}{(x_{(s-2)P} - x_{2P})(x_Q - x_{sP})},$$

where λ_s is the slope of the line $l_{2P,(s-2)P}$. Hence, given $f_{\pm(s-2)}$, we need $2M_k + (k+d+1)M$ to calculate the numerators of $f_{\pm s}$ simultaneously.

We summarize the precomputation scheme in Algorithm 2. Again, we only consider the computations of the numerators of $f_{\pm i}$ due to the efficient denominator elimination technique [7]. Note that $P_i, i \in \{1, 3, \cdots, t\}$ can be calculated with $1I + \frac{9(t-1)}{2}M + (t+5)S$ 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 $f_{\pm i}$. Therefore, the total cost of the precomputation scheme in Algorithm 2 is $(t-3)M_k + 1M_d + 1I + \left((5t-2) + \frac{(k+d)(t-1)}{2}\right)M + (t+5)S$.

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 $E(\mathbb{F}_q)$. The mixed addition $T + P' = (X_3, Y_3, Z_3)$ can be computed efficiently as follows [37]:

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

where $\alpha = 2(Z_1^3 y_2 - Y_1)$ and $\beta = Z_1^2 x_2 - X_1$. In this case, the evaluation of the line function $l_{T,T}$ at the image point Q can be calculated as follows:

$$\begin{aligned} l_{T,P'}(Q) &= (y_Q - y_2) - \frac{2(Z_1^3 y_2 - Y_1)}{Z_3} \cdot (x_Q - x_2) \\ &= \frac{Z_3 y_Q - \alpha x_Q + (\alpha x_2 - Z_3 y_2)}{Z_3}, \end{aligned} \quad (1)$$

where Z_3 and α 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 α 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 $E(\mathbb{F}_q)$. The point doubling $2T = (X_3, Y_3, Z_3)$ can be computed efficiently as follows [37]:

$$\begin{aligned} X_3 &= \alpha^2 - 2\beta, \\ Y_3 &= \alpha(\beta - X_3) - 8Y_1^4, \\ Z_3 &= (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2, \end{aligned}$$

where $\alpha = 3X_1^2 + aZ_1^4$ and $\beta = 2\left[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\right]$ for a general a, and $\alpha = 3(X_1 + Z_1^2)(X_1 - Z_1^2)$ 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:

$$\begin{aligned} l_{T,T}(Q) &= \left(y_Q - \frac{Y_1}{Z_1^3}\right) - \frac{3X_1^2 + aZ_1^4}{Z_3} \cdot \left(x_Q - \frac{X_1}{Z_1^2}\right) \\ &= \frac{(Z_3Z_1^2)y_Q - ((2Y_1^2 - \alpha X_1) + (\alpha Z_1^2)x_Q)}{Z_3Z_1^2}, \end{aligned} \quad (2)$$

where Z_3, Z_1^2, Y_1^2 and α 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 α 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 $E(\mathbb{F}_q)$. The point tripling $3T = (X_3, Y_3, Z_3)$ can be computed efficiently as follows [37]:

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

where $2\alpha = (\theta + \omega)^2 - \theta^2 - \omega^2, 2\beta = 16Y_1^4, \theta = 3X_1^2 + aZ_1^4$ and $\omega = 6\left[(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4\right] - \theta^2$ for a general a. For the case when $a = -3$, θ and ω 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 λ_1' and λ_2' be the slopes of the lines $l_{T,T}$ and $l_{2T,T}$, we can obtain $\lambda_1' = \frac{\theta}{2Y_1Z_1}$ and $\lambda_1' + \lambda_2' = \frac{8Y_1^3}{Z_1\omega}$. In this case, the evaluation of the line function $f_{3,T}$ at the image point Q can be computed as follows:

$$\begin{aligned} f_{3,T}(Q) &= \frac{l_{T,T}(Q)}{v_{2T}(Q)} \cdot \frac{l_{2T,T}(Q)}{v_{3T}(Q)} \\ &= \frac{\left(x_Q - \frac{X_1}{Z_1^2}\right)^2 + (\lambda_1' + \lambda_2')\left[\lambda_1'\left(x_Q - \frac{X_1}{Z_1^2}\right) - \left(y_Q - \frac{Y_1}{Z_1^3}\right)\right]}{x_Q - x_{3T}} \\ &= \frac{(x_QZ_1^2 - X_1)\left[\omega(x_QZ_1^2 - X_1) + 4Y_1^2\theta\right] + 8Y_1^4 - (2Y_1Z_1)^3 y_Q}{\omega Z_1^4(x_Q - x_{3T})} \quad (3) \\ &= \frac{(\omega Z_1^4)x_Q^2 + Z_1^2(4Y_1^2\theta - 2\omega X_1)x_Q + X_1(\omega X_1 - 4Y_1^2\theta) + 8Y_1^4 - (2Y_1Z_1)^3 y_Q}{\omega Z_1^4(x_Q - x_{3T})}, \quad (4) \end{aligned}$$

where $\theta, \omega, 4Y_1^2, Z_1^2, 8Y_1^4$ and Z_1^4 are available from the point tripling. Due to space limitations, we only describe the encapsulated point tripling and line computation for the case $a = -3$ in the following Algorithm 5, where some intermediate results of the point tripling, as shown in the boxes, can be reused for the line computation. However, it is also straightforward to derive the explicit formula for a general a. Assuming that x_Q^2 is precomputed with $1S_d$, Algorithm 3 requires $(14 + 2d + k)M + 9S$ to calculate the point tripling and evaluate the line functions. In the case a

Algorithm 2. Precomputation Scheme for the Multibase Variant of Miller's Algorithm

```
Input: $P = (x_P, y_P) \in E(\mathbb{F}_q)[n]$ and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{q^k})$.
Output: $P_i = iP$ and the numerators $f'_{\pm i}$ of $f_{\pm i} = f_{\pm i,P}(Q)$ for $i \in \{1, 3, \cdots, t\}$,
        where $t \geq 3$ is an odd integer and $\mathrm{div}(f_i) = i\langle P\rangle - \langle iP\rangle - (i-1)\langle\mathcal{O}\rangle$.
1. Compute $P_i = iP = (x_{iP}, y_{iP})$ for $i \in \{1, 3, \cdots, t\}$ and $2P = (x_{2P}, y_{2P})$ using the
   precomputation method proposed in [38] (see pp. 238-240 of [38])
2. $f'_1 \leftarrow 1, f'_{-1} \leftarrow 1$
3. $T_1 \leftarrow -(2y_P^2 + (3x_P^2 + a)(x_Q - x_P)), T_2 \leftarrow 2y_P y_Q$
   $f'_2 \leftarrow T_1 + T_2, f'_{-2} \leftarrow T_1 - T_2$
4. $T_1 \leftarrow x_Q - x_P, T_2 \leftarrow T_1\big((x_P - x_{2P})T_1 + (3x_P^2 + a)\big) + 2y_P^2, T_3 \leftarrow 2y_P y_Q$
   $f'_3 \leftarrow T_2 - T_3, f'_{-3} \leftarrow T_2 + T_3$
5. for $s$ from 5 to $t$ do
6.     $T_1 \leftarrow x_{(s-2)P} - x_{2P}, T_2 \leftarrow T_1 y_Q, T_3 \leftarrow T_1 y_{2P} + (y_{(s-2)P} - y_{2P})(x_Q - x_{2P})$
7.     $f'_s \leftarrow f'_2 \cdot f'_{s-2} \cdot (T_2 - T_3), f'_{-s} \leftarrow f'_{-2} \cdot f'_{-(s-2)} \cdot (T_2 + T_3), s \leftarrow s + 2$
8. return $P_i$ and $f'_{\pm i}$ for $i \in \{1, 3, \cdots, t\}$
```

Algorithm 3. Encapsulated Point Mixed-Addition and Line Computation

Input: $T = (X_1, Y_1, Z_1)$ in Jacobian coordinates, $P' = (x_2, y_2)$ in affine coordinates on $E(\mathbb{F}_q)$ and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{q^k})$.

Output: $T + P' = (X_3, Y_3, Z_3)$ in Jacobian and the numerator l_{ADD} of $l_{T,P'}(Q)$.

Point Mixed-Addition

1.	$T_1 \leftarrow Z_1^2$	**7.**	$Z_3 \leftarrow Z_3 - T_1$	**13.**	$T_3 \leftarrow 8 \cdot T_1$	**19.**	$X_3 \leftarrow T_1^2$
2.	$T_2 \leftarrow x_2 \cdot T_1$	**8.**	$\boxed{Z_3 \leftarrow Z_3 - T_3}$	**14.**	$T_2 \leftarrow T_2 \cdot T_3$	**20.**	$X_3 \leftarrow X_3 - T_2$
3.	$T_2 \leftarrow T_2 - X_1$	**9.**	$T_1 \leftarrow Z_1 \cdot T_1$	**15.**	$T_4 \leftarrow X_1 \cdot T_3$	**21.**	$T_4 \leftarrow T_4/2$
4.	$T_3 \leftarrow T_2^2$	**10.**	$T_1 \leftarrow y_2 \cdot T_1$	**16.**	$Y_3 \leftarrow Y_1 \cdot T_2$	**22.**	$T_4 \leftarrow T_4 - X_3$
5.	$Z_3 \leftarrow Z_1 + T_2$	**11.**	$T_1 \leftarrow T_1 - Y_1$	**17.**	$T_2 \leftarrow T_2/2$	**23.**	$T_4 \leftarrow T_1 \cdot T_4$
6.	$Z_3 \leftarrow Z_3^2$	**12.**	$\boxed{T_1 \leftarrow T_1 + T_1}$	**18.**	$T_2 \leftarrow T_2 + T_4$	**24.**	$Y_3 \leftarrow T_4 - Y_3$

Line Computation

1.	$T_2 \leftarrow x_2 \cdot T_1$	**3.**	$T_2 \leftarrow T_2 - T_3$	**5.**	$T_k \leftarrow y_Q \cdot Z_3$	**7.**	$l_{ADD} \leftarrow T_k + T_2$
2.	$T_3 \leftarrow y_2 \cdot Z_3$	**4.**	$T_d \leftarrow x_Q \cdot T_1$	**6.**	$T_k \leftarrow T_k - T_d$		

Return (X_3, Y_3, Z_3) and l_{ADD}

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., $x_Q \in \mathbb{F}_q$ or $\mathbb{F}_{q^2}$) 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

[1] ARK | All-in-One Blockchain Solutions. https://ark.io/.

[2] J-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2X", https://blake2.net/blake2x.pdf, 2016.

[3] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "SoK: Consensus in the Age of Blockchains", arXiv:1711.03936, 2017.

[4] E. Barker and J. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", *NIST Special Publication 800-90A, Revision 1*, National Institute of Standards and Technology, 2015.

[5] P.L.S.M. Barreto, S. Galbraith, C. Ó hÉigeartaigh, and M. Scott, "Efficient Pairing Computation on Supersingular Abelian Varieties", *Design, Codes and Cryptography*, 42:239-271, 2007.

[6] P.L.S.M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, "Efficient Algorithm for Pairing-Based Cryptosystems", *Advance in Cryptology - CRYPTO'2002*, ser. LNCS 2442, M. Yung (ed.), Berlin, Germany: Springer-Verlag, pp. 354-368, 2002.

[7] P.L.S.M. Barreto, B. Lynn, and M. Scott, "On the Selection of Pairing-Friendly Groups", *Selected Areas in Cryptography - SAC'2002*, ser. LNCS 3006, M. Matsui and R. Zuccherato (eds.), Berlin, Germany: Springer-Verlag, pp. 17-25, 2003.

[8] BitShares. https://bitshares.org/.

[9] G. R. Blakley, "Safeguarding Cryptographic Keys", *International Workshop on Managing Requirements Knowledge*, IEEE Computer Society, pp. 313-317, 1979.

[10] A. Boldyreva, "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme", *Public Key Cryptography – PKC 2003*, ser. LNCS 2567, Y.G. Desmedt (eds.), Berlin, Germany: Springer-Verlag, pp. 31-46, 2003.

[11] D. Boneh, "BLS Multi-Signatures With Public-Key Aggregation", https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html, 2018.

[12] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps", *Advances in Cryptology – EUROCRYPT'03*, ser. LNCS 2656, E. Biham (Ed.), Berlin, Germany: Springer-Verlag, pp. 416-432, 2003.

[13] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing", *Advances in Cryptology – ASIACRYPT'01*, ser. LNCS 2248, C. Boyd (Ed.), Berlin, Germany: Springer-Verlag, pp. 514-532, 2001.

[14] C. Cachin, M. Vukolić, "Blockchain Consensus Protocols in the Wild", arXiv:1707.01873, 2017.

[15] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery", *ACM Transactions on Computer Systems*, Vol. 20, Iss. 4, pp. 398-461, 2002.

[16] cc001 - Professional Lisk Delegate, http://www.liskdelegate.io/.

[17] S. Chatterjee, P. Sarkar, and R. Barua "Efficient Computation of Tate Pairing in Projective Coordinate over General Characteristic Fields", *Information Security and Cryptology - ICISC 2004*, ser. LNCS 3506, C. Park and S. Chee (eds.), Berlin, Germany: Springer-Verlag, pp. 168-181, 2005.

[18] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults", *The IEEE 26th Annual Symposium on Foundations of Computer Science – FOCS'85*, IEEE Computer Society, pp. 383-395, 1985.

[19] R. DeVoe, "EOS vs Ethereum: Larimer and Buterin Politely Debate Risk of Vote Buying", https://www.bitsonline.com/eos-vs-ethereum/.

[20] V. Dimitrov, L. Imbert, and P.K. Mishra, "Efficient and Secure Elliptic Curve Point Multiplication using Double-Base Chains", *Advance in Cryptology - ASIACRYPT'2005*, ser. LNCS 3788, B. Roy (ed.), Berlin, Germany - Springer-Verlag, pp. 59-78, 2005.

Algorithm 4. Encapsulated Point Doubling and Line Computation

Input: $T = (X_1, Y_1, Z_1)$ in Jacobian coordinates on $E(\mathbb{F}_q)$ and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{q^k})$.
Output: $2T = (X_3, Y_3, Z_3)$ in Jacobian coordinates and the numerator l_{DBL} of $l_{T,T}(Q)$.

Point Doubling

$a = -3$				General a			
1.	$\boxed{T_1 \leftarrow Z_1^2}$	**11.**	$T_4 \leftarrow X_1 \cdot T_3$	**1.**	$\boxed{T_1 \leftarrow Z_1^2}$	**13.**	$T_5 \leftarrow T_5^2$
2.	$T_2 \leftarrow X_1 + T_1$	**12.**	$T_4 \leftarrow 4 \cdot T_4$	**2.**	$T_2 \leftarrow T_1^2$	**14.**	$T_4 \leftarrow T_5 - T_4$
3.	$T_3 \leftarrow X_1 - T_1$	**13.**	$X_3 \leftarrow T_2^2$	**3.**	$T_2 \leftarrow a \cdot T_2$	**15.**	$T_5 \leftarrow T_3^2$
4.	$T_2 \leftarrow T_2 \cdot T_3$	**14.**	$X_3 \leftarrow X_3 - T_4$	**4.**	$T_4 \leftarrow X_1^2$	**16.**	$T_4 \leftarrow T_4 - T_5$
5.	$\boxed{T_2 \leftarrow 3 \cdot T_2}$	**15.**	$X_3 \leftarrow X_3 - T_4$	**5.**	$T_4 \leftarrow 3 \cdot T_4$	**17.**	$T_4 \leftarrow T_4 + T_4$
6.	$\boxed{T_3 \leftarrow Y_1^2}$	**16.**	$T_4 \leftarrow T_4 - X_3$	**6.**	$\boxed{T_2 \leftarrow T_2 + T_4}$	**18.**	$X_3 \leftarrow T_2^2$
7.	$Z_3 \leftarrow Y_1 + Z_1$	**17.**	$T_4 \leftarrow T_2 \cdot T_4$	**7.**	$\boxed{T_3 \leftarrow Y_1^2}$	**19.**	$X_3 \leftarrow X_3 - T_4$
8.	$Z_3 \leftarrow Z_3^2$	**18.**	$Y_3 \leftarrow T_3^2$	**8.**	$Z_3 \leftarrow Y_1 + Z_1$	**20.**	$X_3 \leftarrow X_3 - T_4$
9.	$Z_3 \leftarrow Z_3 - T_1$	**19.**	$Y_3 \leftarrow 8 \cdot Y_3$	**9.**	$Z_3 \leftarrow Z_3^2$	**21.**	$T_5 \leftarrow 8 \cdot T_5$
10.	$\boxed{Z_3 \leftarrow Z_3 - T_3}$	**20.**	$Y_3 \leftarrow T_4 - Y_3$	**10.**	$Z_3 \leftarrow Z_3 - T_1$	**22.**	$T_4 \leftarrow T_4 - X_3$
				11.	$\boxed{Z_3 \leftarrow Z_3 - T_3}$	**23.**	$T_4 \leftarrow T_2 \cdot T_4$
				12.	$T_5 \leftarrow X_1 + T_3$	**24.**	$Y_3 \leftarrow T_4 - T_5$

Line Computation

1. $T_4 \leftarrow Z_3 \cdot T_1$	**3.** $T_1 \leftarrow T_1 \cdot T_2$	**5.** $T_3 \leftarrow T_3 - T_2$	**7.** $T_d \leftarrow T_d + T_3$	**9.** $l_{DBL} \leftarrow T_k - T_d$
2. $T_3 \leftarrow T_3 + T_3$	**4.** $T_2 \leftarrow T_2 \cdot X_1$	**6.** $T_d \leftarrow x_Q \cdot T_1$	**8.** $T_k \leftarrow y_Q \cdot T_4$	

Return (X_3, Y_3, Z_3) 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 $E(\mathbb{F}_q)$ and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{q^k})$.
Output: $3T = (X_3, Y_3, Z_3)$ in Jacobian coordinates and the numerator l_{TRL} of $f_{3,T}(Q)$.

Point Tripling

1.	$\boxed{T_1 \leftarrow Z_1^2}$	**9.**	$T_4 \leftarrow 3 \cdot T_4$	**17.**	$Z_3 \leftarrow Z_3 - T_1$	**25.**	$T_8 \leftarrow T_3 \cdot T_6$
2.	$T_2 \leftarrow X_1 + T_1$	**10.**	$T_5 \leftarrow T_2^2$	**18.**	$T_5 \leftarrow T_4^2$	**26.**	$X_3 \leftarrow X_3 + T_8$
3.	$T_3 \leftarrow X_1 - T_1$	**11.**	$\boxed{T_4 = T_4 - T_5}$	**19.**	$Z_3 \leftarrow Z_3 - T_5$	**27.**	$X_3 \leftarrow 4 \cdot X_3$
4.	$T_2 \leftarrow T_2 \cdot T_3$	**12.**	$T_6 = T_2 + T_4$	**20.**	$T_6 \leftarrow T_6 - T_5$	**28.**	$T_8 \leftarrow T_6 + T_7$
5.	$\boxed{T_2 \leftarrow 3 \cdot T_2}$	**13.**	$T_6 = T_6^2$	**21.**	$X_3 \leftarrow X_1 \cdot T_5$	**29.**	$T_8 \leftarrow T_8 \cdot (-T_6)$
6.	$Y_3 \leftarrow 2Y_1$	**14.**	$T_6 = T_6 - T_5$	**22.**	$T_5 \leftarrow T_4 \cdot T_5$	**30.**	$T_8 \leftarrow T_8 - T_5$
7.	$\boxed{T_3 \leftarrow Y_3^2}$	**15.**	$Z_3 = Z_1 + T_4$	**23.**	$\boxed{T_7 \leftarrow T_3^2}$	**31.**	$Y_3 \leftarrow 4 \cdot Y_3$
8.	$T_4 \leftarrow X_1 \cdot T_3$	**16.**	$Z_3 = Z_3^2$	**24.**	$T_6 \leftarrow T_7 - T_6$	**32.**	$Y_3 \leftarrow Y_3 \cdot T_8$

Line Computation

1.	$T_5 \leftarrow X_1 \cdot T_4$	**6.**	$T_5 \leftarrow T_1 \cdot T_5$	**11.**	$T_1 \leftarrow T_1 \cdot T_4$	**16.**	$T_2 \leftarrow T_2 + T_2$
2.	$T_6 \leftarrow T_2 \cdot T_3$	**7.**	$T_7 \leftarrow T_7/2$	**12.**	$T_{d'} \leftarrow x_Q^2 \cdot T_1$	**17.**	$T_3 \leftarrow T_2^2$
3.	$T_6 \leftarrow T_5 - T_6$	**8.**	$T_6 \leftarrow T_6 + T_7$	**13.**	$T_d \leftarrow T_d + T_{d'}$	**18.**	$T_3 \leftarrow T_2 \cdot T_3$
4.	$T_5 \leftarrow -(T_5 + T_6)$	**9.**	$T_d \leftarrow x_Q \cdot T_5$	**14.**	$T_d \leftarrow T_d + T_6$	**19.**	$T_k \leftarrow y_Q \cdot T_3$
5.	$T_6 \leftarrow X_1 \cdot T_6$	**10.**	$T_1 \leftarrow T_1^2$	**15.**	$T_2 \leftarrow Y_1 \cdot Z_1$	**20.**	$l_{TRL} \leftarrow T_d - T_k$

Return (X_3, Y_3, Z_3) and l_{TRL}

[21] C. Doche, and L. Imbert, “Extended Double-Base Number System with Applications to Elliptic Curve Cryptography”, *Progress in Cryptology - INDOCRYPT'2006*, ser. LNCS 4329, B. Barua and T. Lange (eds.), Berlin, Germany: Springer-Verlag, pp. 335-348, 2006.

[22] K. Eisenträger, K. Lauter, and P. L. Montgomery, “Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evaluation”, *The Cryptographer's Track at RSA Conference - CT-RSA'2003*, ser. LNCS 2612, M. Joye (ed.), Berlin, Germany: Springer-Verlag, pp. 343-354, 2003.

[23] EOS.io. https://eos.io/.

[24] P. Feldman, ”A Practical Scheme for Non-Interactive Verifiable Secret Sharing”, *The 28th IEEE Symposium on Foundations of Computer Science – FOCS'87*, IEEE Computer Society, pp. 427-437, 1987.

[25] G. Frey, and H.-G. Rück, “A Remark Concerning m-Divisibility and the Discrete Logarithm Problem in the Divisor Class Group of Curves,” *Mathematics of Computation*, 62(206):865-874, 1994.

[26] S. Galbraith, K. Paterson, and N. Smart, ”Pairings for cryptographers”, *Discrete Applied Mathematics*, Vol. 156, Iss. 16, pp. 3113-3121, 2008.

[27] R. Gennaro, S. Goldfeder, and A. Narayanan, ”Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security”, *The 14th International Conference on Applied Cryptography and Network Security – ACNS 2016*, ser. LNCS 9696, M. Manulis , A-R. Sadeghi, S. Schneider (Eds.), Berlin, Germany: Springer-Verlag, pp. 156-174, 2016.

[28] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, ”Secure Applications of Pedersens Distributed Key Generation Protocol”, *Topics in Cryptology – CT-RSA 2003*, ser. LNCS 2612, M. Joye (Ed.), Berlin, Germany: Springer-Verlag, pp. 373-390, 2003.

[29] T. Hanke, M. Movahedi and D. Williams, ”DIFINITY Technology Overview Series – Consensus System (Rev. 1)”, https://dfinity.org/pdf-viewer/pdfs/viewer?file=../library/dfinity-consensus.pdf, 2018.

[30] F. Hess, N. P. Smart, and F. Vercauteren, “The Eta Pairing Revisited,” in *IEEE Transactions on Information Theory*, 52(10):4595-4602, 2006.

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, ”Secure Distributed Key Generation for Discrete-Log Based Cryptosystems”, *Journal of Cryptology*, Vol. 20, Iss. 1, pp. 51-83, 2007.

[32] Juniper Research. ”The Internet of Things: Consumer, Industrial & Public Services 2018-2023”, https://www.juniperresearch.com/researchstore/iot-m2m/internet-of-things/

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

[33] D. Larimer, ”DPOS Consensus Algorithm – The Missing White Paper”, https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper.

[34] D. Larimer, ”Response to Cosmos white paper’s claims on DPOS security” https://steemit.com/steem/@dantheman/response-to-cosmos-white-paper-s-claims-on-dpos-security.

[35] Lisk: Access the Power of Blockchain. https://lisk.io/.

[36] P. Longa, and C. Gebotys, “Setting Speed Records with the (Fractional) Multibase Non-Adjacent Form Method for Efficient Elliptic Curve Scalar Multiplication”, Cryptology ePrint Archive, Report 2008/118, 2008, http://eprint.iacr.org/2008/118.

[37] P. Longa, and A. Miri, “Fast and Flexible Elliptic Curve Point Arithmetic over Prime Fields”, *IEEE Transactions on Computers*, vol. 57, no. 3, pp. 289-302, 2008.

[38] P. Longa, and A. Miri, “New Composite Operations and Precomputation Scheme for Elliptic Curve Cryptosystems over Prime Fields”, *Public Key Cryptography - PKC’2008*, ser. LNCS 4939, R. Cramer (ed.), Berlin, Germany: Springer-Verlag, pp. 229-247, 2008.

[39] P. Longa, and A. Miri, “New Multibase Non-Adjacent Form Scalar Multiplication and its Application to Elliptic Curve Cryptosystems,” Cryptology ePrint Archive, Report 2008/052, 2008, http://eprint.iacr.org/2008/052.

[40] N. Meloni, “New Point Addition Formulae for ECC Applications”, *International Workshop on the Arithmetic of Finite Fields - WAIFI’2007*, ser. LNCS 4547, C. Carlet and B. Sunar (eds.), Berlin, Germany: Springer-Verlag, pp. 189-201, 2007.

[41] V. S. Miller, “Short Programs for Functions on Curves,” Unpublished manuscript, 1986, available at http://crypto.stanford.edu/miller/miller.pdf.

[42] A. Miyaji, M. Nakabayashi, and S. Takano, “New Explicit Conditions of Elliptic Curve Traces for FR-Reduction,” *IEICE Transactions on Fundamentals*, E84-A(5):1234-1243, 2001.

[43] N. El Mrabet and M. Joye, *Guide to Pairing-Based Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series, CRC Press, 2016.

[44] T. Pedersen, ”A Threshold Cryptosystem Without a Trusted Party”, *Advances in Cryptology – EUROCRYPT’91*, D.W. Davies (Ed.), LNCS 547, pp. 522-526, Springer-Verlag, 1991.

[45] A. Shamir, ”How to Share a Secret”, *Communications of the ACM*, Vol. 22, Iss. 11, pp. 612-613, 1979.

[46] J. H. Silverman. *The Arithmetic of Elliptic Curves*, Number 106 in Graduate Texts in Mathematics. Springer-Verlag, Berlin, Germany, 1986.

[47] M. Snider, K. Samani, and T. Jain, ”Delegated Proof of Stake: Features & Tradeoffs”, Multicoin Capital, 2018. https://multicoin.capital/2018/03/02/delegated-proof-stake-features-tradeoffs/.

[48] Steem. https://steem.io/.

[49] Tezos. https://tezos.com/.