Scalable Practical Byzantine Fault Tolerance with Short-Lived Signature Schemes

Xinxin Fan
IoTeX
Menlo Park, CA, USA
xinxin@iotex.io

ABSTRACT

The Practical Byzantine Fault Tolerance (PBFT) algorithm is a popular solution for establishing consensus in blockchain systems. The execution time of the PBFT consensus algorithm has an important effect on the blockchain throughput. Digital signatures are extensively used in PBFT to ensure the authenticity of messages during the different phases. Due to the round-based and broadcast natures of PBFT, nodes need to verify multiple signatures received from their peers, which incurs significant computational overhead and slows down the consensus process. To address this issue, we propose an efficient short-lived signature based PBFT variant, which utilizes short-length cryptographic keys to sign/verify messages in PBFT for a short period of time and blockchain-aided key distribution mechanisms to update those keys periodically. We also present efficient algorithms for accelerating the software implementation of the BLS threshold signature scheme. Our extensive experiments with three elliptic curves and two signature schemes demonstrate the efficacy of using short-lived signature schemes for improving the scalability of PBFT significantly.

KEYWORDS

Practical Byzantine Fault Tolerance, Scalability, Short-Lived Public Key, ECDSA, BLS Threshold Signature

ACM Reference Format:
Xinxin Fan. 2018. Scalable Practical Byzantine Fault Tolerance with Short-Lived Signature Schemes. In *Proceedings of 28th Annual International Conference on Computer Science and Software Engineering (CASCON'18)*. ACM, New York, NY, USA, 12 pages.

1 INTRODUCTION

A blockchain constitutes a distributed ledger that records transactions in an append-only fashion across a network of random agents, nodes or peers. At the center of the blockchain operation is the maintenance of the consistent global view of the information recorded on the blockchain, which forms the backbone for enabling users to interact with each other in a trustless and decentralized manner. The security of the consensus model is perhaps the most critical aspect for a blockchain platform. The two key properties of a consensus model 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 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, and reach consensus in the present of Byzantine nodes as long as the number of those nodes within the system is limited.

The seminal Practical Byzantine Fault Tolerance (PBFT) algorithm, proposed by Castro and Liskov [8], provides a practical Byzantine state machine replication [36] that tolerates Byzantine failures with low overhead. Essentially, a group of replicas running the PBFT algorithm are ordered in a sequence with one replica being the leader and others the validators. The leader node is changed in round-robin manner and in every round all the nodes communicate with each other asynchronously through a three-phase protocol. The final result is that all honest nodes reach an agreement on the order of the record by either accepting or rejecting it. PBFT provides safety and liveness guarantees provided that the number of Byzantine nodes is less than one third of the overall nodes in the system for any given window of vulnerability. In other words, to tolerate f Byzantine faults, the group needs at least $3f+1$ members. In practice, PBFT and its variants have been adopted by a handful of blockchain platforms such as Hyperledger [21], Tendermint [41], Zilliqa [45], IoTeX [22], just to name a few.

Although PBFT offers the salient property like instant transaction finality, it only works well in its classical form with small consensus group size due to the broadcast nature of the three-phase protocol. More specifically, upon receiving the block proposal from the leader, all the validators need to exchange their opinions in the subsequent phases by broadcasting and verifying multiple signed messages. Using digital signatures at high security levels (e.g., 128-bit), while providing strong security guarantees, incurs significant computational and communication overhead in PBFT, thereby limiting its scalability and throughout. We note that a typical round of PBFT consensus process in blockchain systems is rather fast (i.e., in a few tens of seconds) for a group of tens of nodes. Therefore, it is possible to utilize digital signatures at low or medium security levels (e.g., 56- or 80-bit) in PBFT and update the short-length key pairs periodically, which enables us to maintain the throughput and improve the scalability of PBFT simultaneously. The extensive experiments with two short-lived signature schemes demonstrate that the verification time for multiple signatures received from peers can be reduced significantly in PBFT. We also propose novel key updating techniques to refresh the short-length key pairs with the aid of the blockchain as well as efficient algorithms for accelerating the software implementation of the BLS threshold signature scheme at the medium security level.

The rest of the paper is organized as follows. Section 2 give an overview of elliptic curve and pairing-based cryptography as well as the ECDSA and (threshold) BLS signature schemes, followed by the detailed description of the short-lived signature based PBFT variant and the corresponding blockchain-aided key distribution schemes in Section 3. In Section 4, we describe various optimization techniques for efficient software implementations of two short-lived signature schemes. Section 5 presents our experimental results for the performance of the short-lived signature schemes under the PBFT setting, which is followed by a summary of the related work in Section 6. Finally, we conclude this paper in Section 7.

2 PRELIMINARIES

2.1 Elliptic Curve Cryptography and ECDSA

2.1.1 Elliptic Curve Cryptography. Let $\mathbb{F}_q$ be a finite field with $q = p^l$ elements, where p is a prime and l 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

$$E(\mathbb{F}_q) : y^2 + a_1x + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and the curve discriminant $\Delta \neq 0$, together with an additional *point-at-infinity*, denoted by O. The points on an elliptic curve form an additive Abelian group, where O is the identity element and the group operation is defined by the well-known chord-and-tangent rule [20]. The order of $E(\mathbb{F}_q)$ is denoted by $\#E(\mathbb{F}_q)$ and the order of a point $P \in E(\mathbb{F}_q)$ is defined as the smallest non-negative integer n such that $nP = O$, where $n|\#E(\mathbb{F}_q)$ and nP is the addition of the point P to itself $n-1$ times. Let $\mathbb{G} = \langle P \rangle$ be a cyclic subgroup of $E(\mathbb{F}_q)$ generated by the point P, such that the Elliptic Curve Discrete Logarithm Problem (ECDLP) is intractable. For more details about elliptic curve cryptography, the interested reader is referred to [20].

2.1.2 Elliptic Curve Digital Signature Algorithm (ECDSA). The ECDSA is the elliptic curve analogue of the Digital Signature Algorithm (DSA). Let $\mathbb{G} = \langle P \rangle$ be a cyclic subgroup of $E(\mathbb{F}_q)$ generated by the point P with prime order n and identity element O, and $H : \{0, 1\}^* \rightarrow \mathbb{Z}_n^*$ be a collision-resistant hash function. The ECDSA works as follows:

- **KeyGen**(): Choose a random $d' \overset{R}{\longleftarrow} \mathbb{Z}_n$, set $Q = d'P$ and output a private/public key pair $(sk, pk) = (d', Q)$.
- **Sign**(sk, m): Given a private key sk and a message $m \in \{0, 1\}^*$, the signer does the following:
  1. Select a random $k' \overset{R}{\longleftarrow} \mathbb{Z}_n$ and compute $R = k'P = (x_1, y_1)$.
  2. Compute $r = x_1 \pmod{n}$. If $r = 0$ then goto step 1.
  3. Compute $s = k'^{-1}(H(m) + d' \cdot r) \mod n$. If $s = 0$ then goto step 1.
  4. Output the signature (r, s).
- **Verify**(pk, m, σ): Given the public key pk, the message m, and the signature (r, s), the verifier does the following:
  1. Check that $r, s \in [1, n-1]$. If any verification fails, reject the signature.
  2. Compute $w = s^{-1} \pmod{n}, u_1 = H(m) \cdot w \pmod{n}$ and $u_2 = r \cdot w \pmod{n}$.
  3. Compute $R' = u_1P + u_2Q = (x_1', y_1')$. If $R' = O$, reject the signature. Otherwise, compute $v = x_1' \pmod{n}$.
  4. Accept the signature if and only if $v = r$.

2.2 Bilinear Pairing and BLS Signature

2.2.1 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 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 [16, 32].

2.2.2 The BLS Signature Scheme. In [6], 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 $\alpha \overset{R}{\longleftarrow} \mathbb{Z}_n$, 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 [4, 5], including threshold signatures, multisignatures, aggregate signatures, and blind signatures.

2.2.3 The BLS Threshold Signature Scheme. In [5], 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 [40]. 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 $f(z) \in \mathbb{Z}_n$ 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**$(\sigma_{i_1}, \dots, \sigma_{i_{t+1}})$: Given $t+1$ valid signature shares $\sigma_{i_1}, \dots, \sigma_{i_{t+1}}, \{i_1, \dots, i_{t+1}\} \subset \{1, \dots, N\}$, output signature $\sigma = \prod_{j=1}^{t+1} \sigma_{i_j}^{\lambda_{i_j}}$, where $\lambda_{i_k} = \prod_{j=1, j \neq k}^{t+1} \frac{0-\mathrm{Id}_{i_k}}{\mathrm{Id}_{i_j}-\mathrm{Id}_{i_k}}$, $k = 1, \dots, t+1$ 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.

3 SCALABLE PBFT USING SHORT-LIVED SIGNATURE SCHEMES

3.1 Key Observation on the PBFT Algorithm

PBFT runs in rounds with one node acting as a leader and others as validators in each round. Every round of PBFT consists of the following three phases as illustrated in Figure 1:

1. **Pre-prepare phase**: The leader initiates the consensus process by sending a signed PRE-PREPARE message that is a block proposal containing a certain number of transactions.
2. **Prepare phase**: Upon receiving the PRE-PREPARE message, every node in the consensus group checks the correctness and validity of the block and multicasts a signed PREPARE message (i.e., 'YES/NO') to all the other nodes.
3. **Commit phase**: Based on the analysis of the received PREPARE messages, each node multicasts a signed COMMIT message (i.e., 'YES/NO') to the consensus group. The block proposal is committed to the blockchain only if a sufficient number of nodes in the consensus group agree on it.

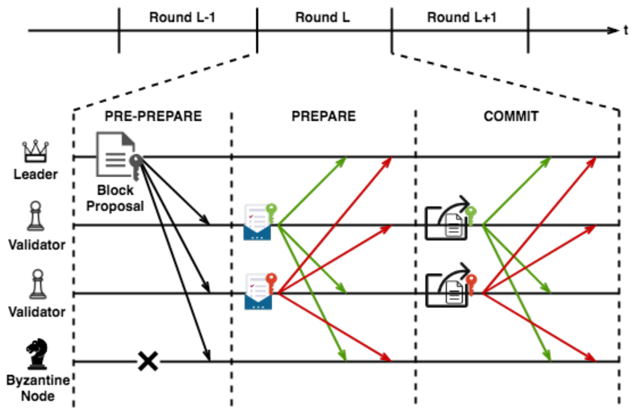


Figure 1: Overview of Three Phases in PBFT

At the end of the three phases, all the honest nodes in the consensus group have the same view regarding to the state of blockchain by either accepting or rejecting the block proposal, thereby achieving the instant transaction finality. In PBFT, both the **Prepare** and **Commit** phases involve intensive interactions among nodes within the consensus group. Given that a consensus group of N players can tolerate $f = \lfloor (N-1)/3 \rfloor$ Byzantine nodes in PBFT, each honest node needs to collect and verify $2f+1$ digital signatures for both phases, respectively, which limits the application of PBFT to small consensus groups due to significant communication and computational overhead. For improving the scalability of PBFT and reducing the computational cost, it is crucial to speeding up the signature verification process. One key observation is that one round of PBFT within a small consensus group can be completed in as little as a few seconds. As a result, in lieu of utilizing digital signatures at the high security levels (e.g., 128-bit), nodes can sign the PREPARE and COMMIT messages with short-lived private keys at the low or medium security levels (e.g., 56- or 80-bit) consecutively for a number of rounds before refreshing those keys. Introducing short-lived key pairs into PBFT brings the following benefits:

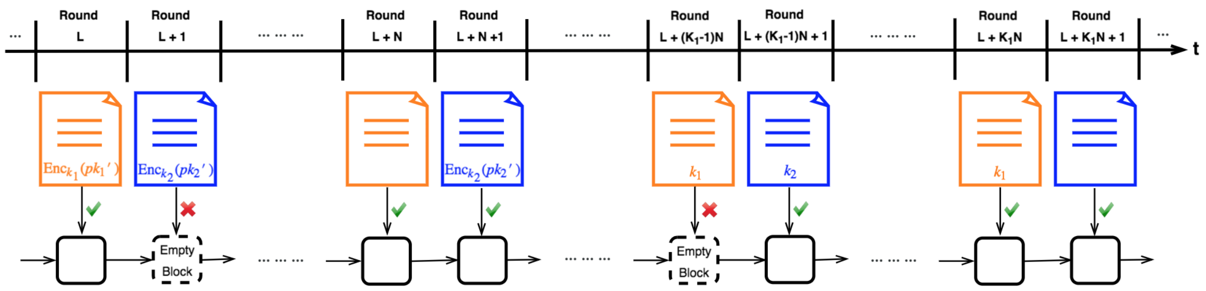
- The signature size is smaller, which reduces the communication overhead;
- The signature verification is faster, which reduces the computational overhead.

The lifetime of a short-lived key pair depending on the difficulty of breaking underlying cryptosystems with the state-of-the-art cryptanalysis techniques and appropriate security margins should be kept when using short-lived key pairs in practice.

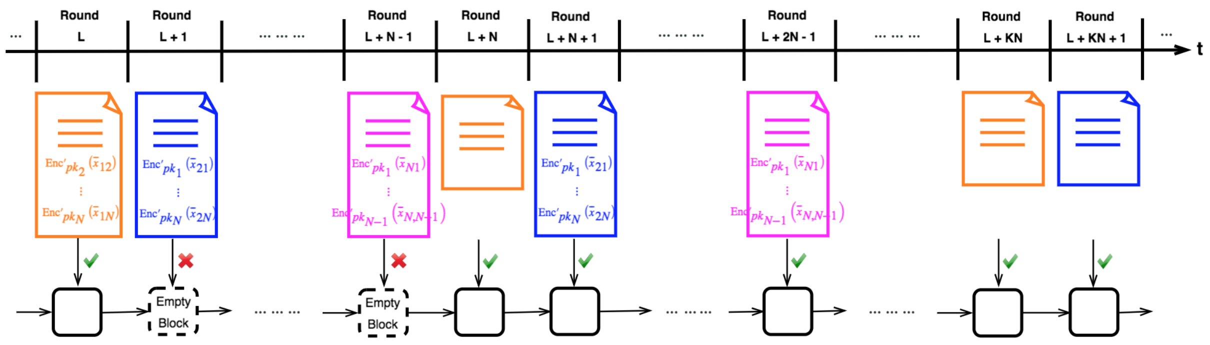
3.2 Security Strength of Short Public Keys

In this work we aim to evaluate the performance when applying the short-lived ECDSA and BLS signature schemes to the **Prepare** and **Commit** phases in PBFT. For the BLS signature, we focus on the threshold variant due to its non-interactive and signature aggregation properties. Three elliptic curves at the low (i.e., 56-bit) and medium (i.e., 70- and 80-bit) security levels are chosen in our experiments. The security strength of those curves, which provides guidance on setting the lifetime of a short-length key pair, has been extensively studied during the past few years.

Regarding to the short-lived ECDSA signatures, we use elliptic curves secp112r1 and sect163k1 specified in [39], which have security levels of 56-bit and 80-bit, respectively. According to [7], the ECDLP on secp112r1 can be solved on a single PS3 with six Synergistic Processing Units (SPUs) in about 1.75 years. On the other hand, breaking the ECDLP on sect163k1 would take about $2.16 \cdot 10^9$ days using $5.19 \cdot 10^6$ FPGAs at the cost of around 10 - 100 billion USD [42]. For implementing the short-lived BLS threshold signature scheme, we utilize the Tate pairing defined over a Miyaji-Nakabayashi-Takano (MNT) curve [31] with embedding degree $k = 6$, which offers an approximate security level of 70-bit. The most recent report [19] for cracking discrete logarithms on a 170-bit MNT curve with embedded degree $k = 3$ requires 2.97 years and no attempt has been made for the case of $k = 6$. While these curves are not recommended for protecting systems requiring high security levels, they are adequate for securing short-lived applications like the round-based PBFT algorithm. In particular, a short-lived key pair can be used safely for at least hundreds of rounds in PBFT,



(a) Key Distribution for the ECDSA Signature Scheme



(b) Key Share Distribution for the BLS Threshold Signature Scheme

Figure 2: Blockchain-Aided Key Distribution for Short-Lived Signatures in the PBFT Algorithm

based on the state-of-the-art techniques for solving the ECDLP on the elliptic curves in question.

3.3 Blockchain-Aided Key Distribution

One remaining challenge for using short-lived signature schemes in PBFT is to refresh the key pair after a certain number of rounds. We note that a blockchain does provide a reliable communication channel for handling key distribution in a dynamic and distributed network environment where entities might behave arbitrarily. Based on this observation, we describe efficient, blockchain-aided key distribution techniques for the ECDSA and BLS threshold signature schemes in this subsection. We assume that node i starts using a new, short-lived key pair (sk_i, pk_i) at round L for the consecutive T rounds, during which each node in the consensus group acts as the leader exactly K times (i.e., $K = T/N$).

3.3.1 Blockchian-Aided Key Distribution for the ECDSA Signature Scheme. To update the short-lived key pair (sk_i, pk_i) in ECDSA, node i in the consensus group does the following:

1. Once node i becomes the leader for the first time, it generates a new key pair (sk'_i, pk'_i) as well as a secret key k_i.
2. Node i encrypts the new public key pk'_i using a symmetric-key algorithm (e.g., AES) and adds the ciphertext $\text{Enc}_{k_i}(pk'_i)$ into the 'extra data' field in its block proposal.
3. If the block fails to be appended to the blockchain for some reason, node i will repeat the Step 2 next time it acts as the leader. This step continues until the block containing the ciphertext $\text{Enc}_{k_i}(pk'_i)$ has been committed in the blockchain successfully.
4. When node i works as the leader for the K_1-th time ($K_1 < K$), it aims to release the secret key k_i to the public by adding it into the 'extra data' field in its block proposal.
5. If the block cannot pass the consensus process, node i will repeat the Step 4 next time it becomes the leader. This step continues until the block containing the secret key k_i has been appended to the blockchain successfully.
6. After the block proposal containing the secret key k_i occurs in the blockchain, other nodes in the consensus group are able to decrypt the ciphertext $\text{Enc}_{k_i}(pk'_i)$ and retrieve the updated public key pk'_i of node i.
7. Node i will be replaced by another node in the system provided that it fails to distribute the updated public key pk'_i to its peers during the lifetime of the key pair (sk_i, pk_i).

The above two-phase key distribution process is illustrated in Figure 2a. Essentially every node in the consensus group first generates a new short-lived key pair and commits the encrypted public key to the blockchain, followed by the release of the secret key for decryption towards the end of life of the old key pair. If a node fails to distribute the updated public key, it is highly likely that the node is under attack and should be evicted from the consensus group.

3.3.2 Blockchian-Aided Key Share Distribution for the BLS Threshold Signature Scheme. Due to the lack of a trusted dealer in blockchain networks, the key updating for the BLS threshold signature scheme involves a distributed key generation (DKG) protocol such as the one proposed by Pedersen in [34]. The Pedersen's DKG scheme requires every node in the consensus group to distribute the shares of a randomly generated secret to all its peers. To update the short-lived key pair share (sk_i, pk_i) in the BLS threshold signature scheme, node i in the consensus group does the following:

1. Once node i becomes the leader for the first time, it chooses at random a polynomial $f_i(z)$ of degree t over $\mathbb{Z}_n$:
$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_{it}z^t,$$
where $f_i(0) = a_{i0} = \bar{x}_i$ is a random secret that node i selects. Node i then computes the shares $\bar{x}_{ij} = f_i(\mathsf{Id}_j) \pmod n$ for $j \in [1, N], j \neq i$, where Id_j is the identifier of node j.
2. For $j \in [1, N], j \neq i$, node i encrypts the shares $\bar{x}_{ij}$ using a public-key algorithm and obtains a set of ciphertexts $\{\text{Enc}'_{pk_j}(\bar{x}_{ij})\}$. Node i also computes a set of witnesses $\{W_{ik} = a_{ik}P\}$ for $k \in [0, t]$. Both ciphertext and witness sets are then added into the 'extra data' field in node i's block proposal.
3. If the block fails to be appended to the blockchain for some reason, node i will repeat the Step 2 next time it acts as the leader. This step continues until the block containing the ciphertext and witness sets has been committed in the blockchain successfully.
4. When all the peers in the consensus group complete the above steps, node i is able to compute the new key pair share (sk'_i, pk'_i) by following the Pedersen's DKG scheme.
5. Node i will be replaced by another node in the system provided that it is marked by its peers as 'disqualified' in the DKG scheme or fails to commit the block containing the ciphertext and witness sets during the lifetime of the key pair share (sk_i, pk_i).

The above key share distribution process is illustrated in Figure 2b. Basically every node in the consensus group acts as the trusted dealer and commits the encrypted shares of a randomly selected secret to the blockchain. By collecting the corresponding shares from the blockchain, each node is able to calculate the new short-lived key pair share. If a node provides incorrect shares or fails to distribute the shares, it will be evicted from the consensus group due to potential attacks.

4 OPTIMIZATION TECHNIQUES FOR IMPLEMENTING SHORT-LIVED SIGNATURE SCHEMES

In this section, we discuss various optimization techniques used in our implementations of the short-lived ECDSA and BLS threshold signature schemes.

4.1 Efficient Implementation of the ECDSA Signature Scheme

The parameters of the two elliptic curves secp112r1 and sect163k1 are summarized in Appendix A.1 and A.2, respectively. While secp112r1 is an elliptic curve defined over a 112-bit prime field, sect163k1 is a Koblitz curve defined over a 163-bit binary field. For efficient implementation of elliptic curve arithmetic, we utilize various optimization techniques proposed in the literature such as the lambda coordinates [33], the window-Non-Adjacent Form (NAF) method [20], the pre-computation [43], etc. Furthermore, BLAKE2 hash function [35] is used in our implementation due to its high performance on different computing platforms.

For speeding up the verification of multiple ECDSA signatures in the PBFT algorithm, we implement the efficient batch verification technique proposed in [10], which is applicable to the modified ECDSA scheme in [1]. The security of the modified ECDSA scheme is equivalent to the standard ECDSA, but the modified signature is the pair (R, s) instead of (r, s) in the original scheme.

4.2 Efficient Implementation of the BLS Threshold Signature Scheme

The parameters of the MNT curve with embedding degree $k = 6$ is summarized in Appendix A.3. To improve the performance of the threshold BLS signature scheme, we propose a multibase variant of Miller's algorithm for computing the Tate pairing and generalize the method of encapsulating the computation of the line function with the group operations described in [9] to the multibase case, as detailed in the following subsections.

4.2.1 Tate Pairing on Elliptic Curves. 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 $\text{div}(f) \equiv \sum_{P\in E} \text{ord}_P(f)\langle P\rangle$, 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 $f(D) \equiv \prod_{P\in E} f(P)^{a_P}$. A divisor D is called a *principal* divisor if $D = \text{div}(f)$ for some rational function f. A divisor $D = \sum_{P\in E} a_P\langle P\rangle$ 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 $\text{div}(f_{n,P}) = nD_P = n\langle P\rangle - n\langle \mathcal{O}\rangle$. 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* [15] 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 [30], 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 [2], 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.*

4.2.2 Multibase Number Representation. Various *multibase number representations* (MSRs) have been proposed to accelerate elliptic curve scalar multiplication, see [12, 13, 25, 28] for examples. 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 [28], 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 MSR, Longa *et al.* [25, 28] proposed *Multibase Non-Adjacent Form* (*mb*NAF) and its window-based variants including *Window-w Multibase Non-Adjacent Form* (*wmb*NAF) and *Fractional Window-w Multibase Non-Adjacent Form* (Frac-*wmb*NAF). Combined with optimized composite operations and precomputation schemes [27], the multibase methods have set new speed records for computing the elliptic curve scalar multiplication [25].

4.2.3 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-*wmb*NAF method [25] to record the scalar, which has been shown to achieve the highest performance among window-based methods for standard elliptic curves in Weierstrass form.

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 $\mathcal{D} = \{0, \pm1, \pm3, \cdots, \pm t\}$ 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-*wmb*NAF 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.

We now 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 $d = k/w$, 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 [3]. 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 $\mathbb{F}_{q^k}$.

4.2.4 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 [27] 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 [29]. 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 [27]. 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.* [14], 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:

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

Algorithm 1. A Multibase Variant of Miller's Algorithm for Computing Tate Pairing

```
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 $i \in \{1, 3, \cdots, t\}$, $t \geq 3$ is an odd integer
2. Compute $f_{\pm i} = f_{\pm i,P}(Q)$ for $i \in \{1, 3, \cdots, t\}$, where
   $\mathrm{div}(f_i) = i\langle P\rangle - \langle iP\rangle - (i-1)\langle O\rangle$
[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 $f' \leftarrow f'^{b_i} \cdot \frac{l_{T,T}(Q)}{v_{2T}(Q)} \cdot \frac{l_{2T,T}(Q)}{v_{3T}(Q)} \cdots \frac{l_{2T,(b_i-2)T}(Q)}{v_{b_iT}(Q)}, T \leftarrow b_iT$
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 $f'^{(q^k-1)/n}$
```

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 $5 \le s \le t$, $f_{\pm s}$ 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 [3]. 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 [27]. 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 $(t - 3)M_k + 1M_d + 1I + \left((5t - 2) + \frac{(k+d)(t-1)}{2}\right) M + (t + 5)S$.

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 $t \ge 3$ is an odd integer and
        $\mathrm{div}(f_i) = i\langle P\rangle - \langle iP\rangle - (i-1)\langle O\rangle$.
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 [27] (see pp. 238-240 of [27])
2. $f'_1 \leftarrow 1, f'_{-1} \leftarrow 1$
3. $T_1 \leftarrow -\left(2y_P^2 + (3x_P^2 + a)(x_Q - x_P)\right), T_2 \leftarrow 2y_P y_Q$
   $f'_2 \leftarrow T_1 + T_2, f'_{-2} \leftarrow T_1 - T_2$
4. $T_1 \leftarrow x_Q - x_P, T_2 \leftarrow T_1\left((x_P - x_{2P})T_1 + (3x_P^2 + a)\right) + 2y_P^2$,
   $T_3 \leftarrow 2y_P y_Q, f'_3 \leftarrow T_2 - T_3, f'_{-3} \leftarrow T_2 + T_3$
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 $i \in \{1, 3, \cdots, t\}$
```

4.2.5 *Encapsulated Composite Operations and Line Computations.*

In [9], 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 [26] 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 [26]:

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

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:

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

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 [26]:

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

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:

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

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

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}

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

$a = -3$

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

General a

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

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 [26]:

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

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 [14] 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 $f_{3,T}$

```
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.  $\boxed{T_1 \leftarrow Z_1^2}$
2.  $T_2 \leftarrow X_1 + T_1$
3.  $T_3 \leftarrow X_1 - T_1$
4.  $T_2 \leftarrow T_2 \cdot T_3$
5.  $\boxed{T_2 \leftarrow 3 \cdot T_2}$
6.  $Y_3 \leftarrow 2Y_1$
7.  $\boxed{T_3 \leftarrow Y_3^2}$
8.  $T_4 \leftarrow X_1 \cdot T_3$
9.  $T_4 \leftarrow 3 \cdot T_4$
10. $T_5 \leftarrow T_2^2$
11. $\boxed{T_4 = T_4 - T_5}$
12. $T_6 = T_2 + T_4$
13. $T_6 = T_6^2$
14. $T_6 = T_6 - T_5$
15. $Z_3 = Z_1 + T_4$
16. $Z_3 = Z_3^2$
17. $Z_3 \leftarrow Z_3 - T_1$
18. $T_5 \leftarrow T_4^2$
19. $Z_3 \leftarrow Z_3 - T_5$
20. $T_6 \leftarrow T_6 - T_5$
21. $X_3 \leftarrow X_1 \cdot T_5$
22. $T_5 \leftarrow T_4 \cdot T_5$
23. $\boxed{T_7 \leftarrow T_3^2}$
24. $T_6 \leftarrow T_7 - T_6$
25. $T_8 \leftarrow T_3 \cdot T_6$
26. $X_3 \leftarrow X_3 + T_8$
27. $X_3 \leftarrow 4 \cdot X_3$
28. $T_8 \leftarrow T_6 + T_7$
29. $T_8 \leftarrow T_8 \cdot (-T_6)$
30. $T_8 \leftarrow T_8 - T_5$
31. $Y_3 \leftarrow 4 \cdot Y_3$
32. $Y_3 \leftarrow Y_3 \cdot T_8$
Line Computation
1.  $T_5 \leftarrow X_1 \cdot T_4$
2.  $T_6 \leftarrow T_2 \cdot T_3$
3.  $T_6 \leftarrow T_5 - T_6$
4.  $T_5 \leftarrow -(T_5 + T_6)$
5.  $T_6 \leftarrow X_1 \cdot T_6$
6.  $T_5 \leftarrow T_1 \cdot T_5$
7.  $T_7 \leftarrow T_7/2$
8.  $T_6 \leftarrow T_6 + T_7$
9.  $T_d \leftarrow x_Q \cdot T_5$
10. $T_1 \leftarrow T_1^2$
11. $T_1 \leftarrow T_1 \cdot T_4$
12. $T_{d'} \leftarrow x_Q^2 \cdot T_1$
13. $T_d \leftarrow T_d + T_{d'}$
14. $T_d \leftarrow T_d + T_6$
15. $T_2 \leftarrow Y_1 \cdot Z_1$
16. $T_2 \leftarrow T_2 + T_2$
17. $T_3 \leftarrow T_2^2$
18. $T_3 \leftarrow T_2 \cdot T_3$
19. $T_k \leftarrow y_Q \cdot T_3$
20. $l_{TRL} \leftarrow T_d - T_k$
Return $(X_3, Y_3, Z_3)$ and $l_{TRL}$
```

at the image point Q can be computed as follows:

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

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 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.

we summarize the cost of our encapsulated composite operations and line computations as well as previous work for computing the Tate pairing in Table 1. When compared to the Chatterjee *et. al.*'s work (where $k = 2$ and $d = 1$) [9], our formulae trade $2M$ for $2S$ in the encapsulated point doubling and line computation, and $1M$ for $1S$ in the encapsulated point mixed addition and line computation, respectively. We also give the computation cost of the encapsulated composite operations and line computations on general pairing-friendly curves.

4.2.6 *Finite Field Arithmetic.*

For the MNT curve in our implementation, $p \equiv 1 \bmod 3$ (where $\xi = -2$ is a cubic non-residue in $\mathbb{F}_p$) and $\gamma = -\sqrt[3]{-2}$ is a quadratic non-residue in $\mathbb{F}_{p^3}$. We can construct the extension field $\mathbb{F}_{p^6}$ as a tower of finite extensions:

$$\begin{aligned}
\mathbb{F}_{p^3} &= \mathbb{F}_p[X]/(X^3 - \xi), \\
\mathbb{F}_{p^6} &= \mathbb{F}_{p^3}[Y]/(Y^2 - \gamma).
\end{aligned}$$

Hence an element $\eta \in \mathbb{F}_{p^6}$ can be represented in any of the following three ways:

$$\begin{aligned}
\eta &= \eta_0 + \eta_1\sqrt[6]{\xi} \\
&= (\eta_{0,0} + \eta_{0,1}\sqrt[3]{\xi} + \eta_{0,2}\sqrt[3]{\xi^2}) + (\eta_{1,0} + \eta_{1,1}\sqrt[3]{\xi} + \eta_{1,2}\sqrt[3]{\xi^2})\sqrt[6]{\xi} \\
&= \eta_{0,0} + \eta_{1,0}\sqrt[6]{\xi} + \eta_{0,1}\sqrt[3]{\xi} + \eta_{1,1}\sqrt{\xi} + \eta_{0,2}\sqrt[3]{\xi^2} + \eta_{1,2}\sqrt[6]{\xi^5},
\end{aligned}$$

where $\eta_0, \eta_1 \in \mathbb{F}_{p^3}$ and $\eta_{0,0}, \eta_{0,1}, \eta_{0,2}, \eta_{1,0}, \eta_{1,1}, \eta_{1,2} \in \mathbb{F}_p$. With the above notations, the group homomorphism $\phi : E'(\mathbb{F}_{p^3}) \longrightarrow E(\mathbb{F}_{p^6})$ defined as $\phi((x, y)) \mapsto (x, y\sqrt[6]{\xi})$ can be computed at no cost. For the arithmetic in $\mathbb{F}_{p^3}$, Karatsuba's method reduces a multiplication and a squaring in $\mathbb{F}_{p^3}$ to 6 multiplications and 6 squarings in $\mathbb{F}_p$, respectively. Moreover, inversion in $\mathbb{F}_{p^3}$ costs $1I + 9M + 3S$ in $\mathbb{F}_p$ via the following direct inversion formulae: if $\mu = \mu_0 + \mu_1 X + \mu_2 X^2 \in \mathbb{F}_{p^3}$, then $\mu^{-1} = \nu_0 + \nu_1 X + \nu_2 X^2$ where $\nu_0 = (\mu_0^2 + 2\mu_1\mu_2)\Delta^{-1}$, $\nu_1 = -(2\mu_2^2 + \mu_0\mu_1)\Delta^{-1}$, $\nu_2 = (\mu_1^2 - \mu_0\mu_2)\Delta^{-1}$ and $\Delta = \mu_0(\mu_0^2 + 6\mu_1\mu_2) + 2(2\mu_2^3 - \mu_1^3)$. Since $\mathbb{F}_{p^6}$ is a tower of cubic and quadratic extensions, it is more efficient to use Karatsuba over Karatsuba for multiplication, and use Complex over Karatsuba for squaring in $\mathbb{F}_{p^6}$ based on the exhaustive testing described in [11]. Therefore, the computation cost for the multiplication and the squaring in $\mathbb{F}_{p^6}$ is $18M$ and $12M$, respectively. Furthermore, inversion in $\mathbb{F}_{p^6}$ can be reduced to 1 inversion, 2 multiplications, and 2 squarings in $\mathbb{F}_{p^3}$.

4.2.7 *Fast Hashing to $\mathbb{G}_2'$.*

To implement the BLS scheme on the MNT curve, we require a point $Q \in E(\mathbb{F}_{p^6})$ of order n. Since the

Table 1: Cost of Group Operations and Line Computations for Computing the Tate Pairing

Reference	Coordinate	k and d	General a	Small a	$a = -3$
Point Doubling and Line Computation					
Zhao *et. al.* [44]	Affine	General	$1I + (4 + 3.5k)M + 2S$	–	–
Chatterjee *et. al.* [9]	Jacobian	$k = 2, d = 1$	$8M + 6S$	$7M + 6S$	$8M + 4S$
This work	Jacobian	$k = 2, d = 1$	$6M + 8S$	$5M + 8S$	$7M + 5S$
(see Algorihtm 3)		General	$(5 + d + k)M + 8S$	$(4 + k + d)M + 8S$	$(6 + d + k)M + 5S$
Point (Mixed) Addition and Line Computation					
Zhao *et. al.* [44]	Affine	General	$1I + (3 + 2.5k)M + 1S$		
Chatterjee *et. al.* [9]	Jacobian	$k = 2, d = 1$	$11M + 3S$		
This work	Jacobian	$k = 2, d = 1$	$10M + 4S$		
(see Algorithm 4)		General	$(9 + d + k)M + 4S$		
Point Tripling and Line Computation					
Zhao *et. al.* [44]	Affine	General	$1I + (9 + 2k)M + 4S + 1M_k$	–	–
This work	Jacobian	$d = 1$	$(12 + k)M + 10S$	$(11 + k)M + 10S$	$(13 + k)M + 8S$
(see Algorithm 5)		$d = 2$	$(16 + k)M + 10S$	$(15 + k)M + 10S$	$(17 + k)M + 8S$
		$d \geq 3$	$(13 + 2d + k)M + 11S$	$(12 + 2d + k)M + 11S$	$(14 + 2d + k)M + 9S$

MNT curve has quadratic twists $E'(\mathbb{F}_{p^3})$, we only need to generate a point $Q' \in E'(\mathbb{F}_{p^3})$ of order n. The standard approach is to first generate a random point on $E'(\mathbb{F}_{p^3})$ and then multiply by the cofactor $c = \#E'(\mathbb{F}_{p^3})/n$ which is of a size in bits approximately the same as p^2. In [38], Scott *et. al.* proposed a fast method for the cofactor multiplication on $\mathbb{G}'_2 = \langle Q' \rangle$ using an efficiently computable homomorphism. Here, we adapt their idea to the MNT curve in question. Noting that $n = (z^2 - z + 1)/3, p = 2n + z$ and the trace of the Frobenius $\tau = z + 1$, we can represent the cofactor c in terms of p and z as follows:

$$c = \frac{\#E'(\mathbb{F}_{p^3})}{n} = \frac{p^3 + 1 + \tau^3 - 3\tau p}{n} = 2p^2 + (2z + 3)p - z - 2.$$

Let π_p be the p-power Frobenius map on E and ϕ be the group homomorphism defined above. Then $\psi = \phi^{-1}\pi_p\phi$ is an endomorphism of E' such that $\psi : \mathbb{G}'_2 \longrightarrow \mathbb{G}'_2$ and for any point $Q' \in \mathbb{G}'_2$ the identity $[p]Q' = [\tau]\psi(Q') - \psi^2(Q')$ holds [17]. Using this identity, we can calculate cQ as follows:

$$\begin{aligned} cQ' &= [-z - 2]Q' + (2z + 3)[p]Q' + 2[p^2]Q' \\ &= [2]Q' + [4z + 2]\psi(Q') + [4z]\psi^2(Q') \\ &= 2Q' + \psi(4zQ' + 2Q') + \psi^2(4zQ'). \end{aligned}$$

Hence, the cost for computing cQ' is one point doubling, one scalar multiplication by $2z$, three applications of the homomorphism ψ and three point additions.

4.2.8 Final Exponentiation. For the MNT curve used in our implementation, the output of the multibase variant of Miller's algorithm must be exponentiated to the power of $(p^6 - 1)/n$. The final exponentiation can be further broken down into the following three components:

$$(p^6 - 1)/n = (p^3 - 1) \cdot (p + 1) \cdot [(p^2 - p + 1)/n].$$

Let $f' = f'_0 + f'_1\sqrt[6]{\xi} \in \mathbb{F}_{p^6}$ be the output of Miller's algorithm. Then exponentiating f' to $p^3 - 1$ can be trivially computed with a conjugation with respect to $\mathbb{F}_{p^3}$, a multiplication and an inversion in $\mathbb{F}_{p^6}$. Using the basis described in Subsection 4.2.6, the conjugation of f' with respect to $\mathbb{F}_{p^3}$ is calculated as $\bar{f}' = f'_0 - f'_1\sqrt[6]{\xi}$. The second part of the final exponentiation can be computed with an application of the Frobenius and one multiplication in $\mathbb{F}_{p^6}$. The remaining exponentiation to $(p^2 - p + 1)/n$ can be dealt with the method proposed in [37]. Noting that $n = (z^2 - z + 1)/3$ and $p = 2n + z$, we obtain the following relation:

$$\begin{aligned} \frac{p^2 - p + 1}{n} &= \frac{(2n + z)^2 - (2n + z) + 1}{n} \\ &= \frac{2n(2n + 2z - 1) + (z^2 - z + 1)}{n} = 2(p + z) + 1. \end{aligned}$$

Therefore, the last part of the final exponentiation can be calculated with an application of the Frobenius, one squaring, two multiplications, and a simple exponentiation to the power of z in $\mathbb{F}_{p^6}$.

5 PERFORMANCE EVALUATION

In this section, we present the experimental results to validate the performance of the short-lived ECDSA and BLS threshold signature schemes under the PBFT setting. We implement three software libraries for the elliptic curves secp112r1, sect163k1 and the MNT curve, respectively. Those libraries are written in C and complied using the GNU C Compiler (GCC) on a MacBook Pro with an Intel Core i7 2.2 GHz CPU and 16GB RAM. We consider PBFT groups varying in size from 52 to 502. Recall from Section 3.1 that in both **Prepare** and **Commit** phases each node needs to verify $2f + 1$ signatures received from its peers, where f is number of Byzantine nodes in PBFT. For the ECDSA signature scheme, we implement the batch verification approach [10] for verifying $2f + 1$ modified ECDSA signatures. On the other hand, for the BLS threshold signature scheme, we set the threshold as $2f$ for different PBFT group sizes. The performance of verifying multiple signatures using the short-lived signature schemes is illustrated in Figure 3, where N is the PBFT group size and S is the number of signatures that need to be verified for each node.

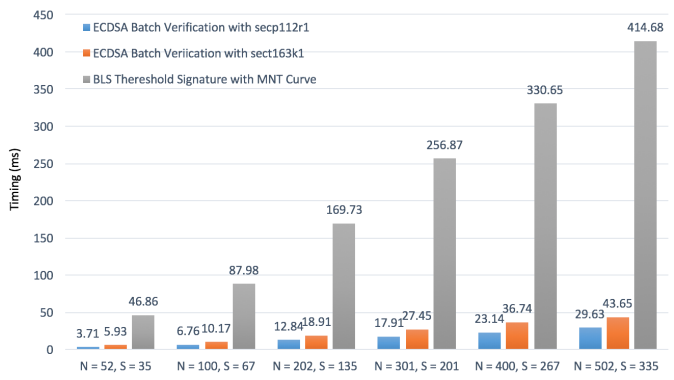


Figure 3: Performance of Verifying Multiple Signatures Using the Short-Lived Signature Schemes in PBFT

From Figure 3, we note that using the short-lived signature schemes during the PBFT consensus process is able to increase the performance of verifying multiple signatures significantly, thereby improving the scalability of PBFT accordingly. In particular, the modified ECDSA coupled with the batch verification is around 10 times faster than the BLS threshold signature scheme. The reason lies in the slower signature share combination process and the computation of the two Tate pairings for verification in the BLS threshold signature scheme.

With respect to the communication overhead, the leader only needs to sends one signed message in each round of PBFT. Each validator, on the other hand, needs to collect $2f + 1$ signatures from its peers. Note that the size of a modified ECDSA (resp. the BLS threshold signature scheme) is three (resp. one) field elements. Table 2 summarizes the communication overhead for each validator in PBFT when using the different elliptic curves and signature schemes. It is not difficult to find out the communication overhead with the BLS threshold signature scheme is at least 50% less than that of the modified ECDSA scheme.

Table 2: Communication Overhead for Validators in PBFT (in Bytes)

Group Size	# Signatures	Validator		
		secp112r1	secp163k1	MNT
52	35	1, 470	2, 205	700
100	67	2, 814	4, 221	1, 340
202	135	5, 670	8, 505	2, 700
301	201	8, 442	12, 663	4, 020
400	267	11, 214	16, 821	5, 340
502	335	14, 070	21, 105	6, 700

6 RELATED WORK

ByzCoin [23] reduced the communication overhead of PBFT from $O(n^2)$ to $O(n)$ by replacing the MAC based all-to-all communication with a primitive called scalable collective signing (CoSi). Cosi is composed of four rounds of communication and a collective signature, which is structured as a tree of Schnorr signature, is generated and verified by all members of the group at the end. As a result, each node does not need to collect individual signatures form its peers any more.

In [24], Li *et al.* proposed Gosig, a new BFT protocol that combines crypto-based leader selection and multi-round voting schemes to improve the scalability under a permissioned blockchain setting. In particular, the author adopted an (insecure) BLS multi-signature scheme to reduce the storage and communication overhead and the PKI to rule out the potential attacks for the aggregated signatures.

In [18], Gueta *et al.* presented SBFT, a scalable decentralized trust infrastructure for blockchains. SBFT implemented a new BFT algorithm for addressing the scalability and decentralization issues. The BLS threshold signature scheme at the higher security level (i.e., 128-bit) has been extensively used in the SBFT design to reduce the communication overhead through signature aggregation.

7 CONCLUSIONS

In this paper we propose an efficient variant for the classical PBFT consensus algorithm, which takes advantage of the short-lived signature schemes to accelerate the signature verification process across the different phases of PBFT. When combining with novel blockchain-aided key distribution schemes, the scalability of the PBFT can be improved significantly. The extensive experiments with three elliptic curves and two signature schemes further highlight the advantages of the proposed approach for reducing the communicational and computation overhead of PBFT.

A PARAMETERS FOR ELLIPTIC CURVES

A.1 Elliptic Curve secp112r1

The elliptic curve secp112r1 [39] is given by the equation $E(\mathbb{F}_p)$: $y^2 = x^3 - 3x + b$ over a prime field $\mathbb{F}_p$, where

$p =$ `db7c 2abf62e3 5e668076 bead208b`

$b =$ `659e f8ba0439 16eede89 11702b22`

The base point $G = (x, y)$ is:

$x =$ `0948 7239995a 5ee76b55 f9c2f098`

$y =$ `a89c e5af8724 c0a23e0e 0ff77500`

The order n of G and the cofactor h are:

$n =$ `db7c 2abf62e3 5e7628df ac6561c5`

$h =$ `01`

A.2 Elliptic Curve sect163k1

The Koblitz curve sect163k1 [39] is given by the equation $E(\mathbb{F}_{2^{163}})$: $y^2+xy = x^3+x^2+1$ over a binary field $\mathbb{F}_{2^{163}}$, where the binary field is defined by the irreducible polynomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$.

The base point $G = (x, y)$ is:

$x =$ `02 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8`

$y =$ `02 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9`

The order n of G and the cofactor h are:

$n =$ `04 00000000 00000000 00020108 a2e0cc0d 99f8a5ef`

$h =$ `02`

A.3 MNT Curve with Embedding Degree $k = 6$

The MNT curve with embedding degree $k = 6$ [31] is given by the equation $E(\mathbb{F}_p) : y^2 = x^3 - 3x + b$ over a prime field $\mathbb{F}_p$, where

$p =$ `7ddca613 a2e3ddb1 749d0195 bb9f14cf 44626303`

$b =$ `21c3f3ac 7864d1f1 f99273d0 f828d365 7d8cfd4e`

The order n of G and the cofactor h are:

$n =$ `3eee5309 d171eed8 ba4e12de f44414fd 17d369b7`

$h =$ `02`

The parameters p and n can be represented in terms of an 80-bit integer $z =$ `dbd7d316ead514bb8f95` (in hexadecimal) such that $n = (z^2 - z + 1)/3$ and $p = 2n + z$.

REFERENCES

[1] A. Antipa, D. Brown, R. Gallant, R. Lambert, R. Struik, and S. Vanstone, "Accelerated Verification of ECDSA Signatures", *Selected Areas in Cryptography - SAC'2005*, ser. LNCS 3897, B. Preneel and S. Tavares (eds.), Berlin, Germany: Springer-Verlag, pp. 307-318, 2005.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery, "Solving a 112-bit Prime Elliptic Curve Discrete Logarithm Problem on Game Consoles using Sloppy Reduction", *International Journal of Applied Cryptography*, vol. 2, no. 3, pp. 212-228, Inderscience Enterprises Ltd., 2012

[8] 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.

[9] 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.

[10] J. H. Cheon and J. H. Yi, "Fast Batch Verification of Multiple Signatures", *Public Key Cryptography - PKC 2007*, ser. LNCS 4450, T. Okamoto and X. Wang (eds.), Berlin, Germany: Springer-Verlag, pp. 442-457, 2007.

[11] A.J. Devegili, C. Ó hÉigeartaigh, M. Scott, and R. Dahab, "Multiplication and Squaring on Pairing-Friendly Fields," Cryptology ePrint Archive, Report 2006/471, 2006, http://eprint.iacr.org/2006/471.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] S. Galbraith, and M. Scott, "Exponentiations in Pairing-Friendly Groups Using Homomorphisms," *Pairing-Based Cryptography - Pairing'2008*, ser. LNCS 5209, S. Galbraith and K. Paterson (eds.), Berlin, Germany: Springer-Verlag, pp. 211-224, 2008.

[18] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, A. Tomescu, "SBFT: a Scalable Decentralized Trust Infrastructure for Blockchains", CoRR, abs/1804.01626, 2018.

[19] A. Guillevic, F. Morain, and E. Thomé, "Solving Discrete Logarithms on a 170-Bit MNT Curve by Pairing Reduction", *Selected Areas in Cryptography – SAC 2016*, ser. LNCS 10532, R. Avanzi, H. Heys (Eds.), Berlin, Germany: Springer-Verlag, pp. 559-578, 2016.

[20] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer Professional Computing Series, Springer, 2004.

[21] Hyperledger. https://www.hyperledger.org/.

[22] IoTeX. https://www.iotex.io/.

[23] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing", *The 25th USENIX Security Symposium (USENIX Security)*, pp. 279-296, 2016.

[24] P. Li, G. Wang, X. Chen, W. Xu, "Gosig: Scalable Byzantine Consensus on Adversarial Wide Area Network for Blockchains", CoRR, abs/1802.01315, 2018.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] V. S. Miller, "Short Programs for Functions on Curves," Unpublished manuscript, 1986, http://crypto.stanford.edu/miller/miller.pdf.

[31] 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.

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

[33] T. Oliveira, J. López, D. F. Aranha, and F. Rodríguez-Henríquez, "Two is the Fastest Prime: Lambda Coordinates for Binary Elliptic Curves", *Journal of Cryptographic Engineering*, Vol. 4, Iss. 1, pp. 3-17, 2014.

[34] 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.

[35] M-J. Saarinen and J-P. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)". RFC 7693 (Informational), https://tools.ietf.org/html/rfc7693, 2015.

[36] F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial", *ACM Computing Surveys*, Vol. 22, Iss. 4, pp. 299-319, 1990.

[37] M. Scott, N. Benger, M. Charlemagne, P. L. J. Dominguez, and E. Kachiza, "On the Final Exponentiation for Calculating Pairings on Ordinary Elliptic Curves," Cryptology ePrint Archive, Report 2008/490, 2008, http://eprint.iacr.org/2008/490.

[38] M. Scott, N. Benger, M. Charlemagne, P. L. J. Dominguez, and E. Kachiza, "Fast Hashing to G_2 on Pairing Friendly Curves," Cryptology ePrint Archive, Report 2008/530, 2008, http://eprint.iacr.org/2008/530.

[39] Standards for Efficient Cryptography Group, "SEC 2: Recommended Elliptic Curve Domain Parameters (Version 1.0)", 2000, http://www.secg.org/SEC2-Ver-1.0.pdf.

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

[41] Tendermint. https://tendermint.com/.

[42] E. Wenger and P. Wolfger, "Harder, Better, Faster, Stronger: Elliptic Curve Discrete Logarithm Computations on FPGAs", *Journal of Cryptographic Engineering*, Vol. 6, Iss. 4, pp. 287-297, Springer-Verlag, 2016.

[43] W. Yu, S. A. Musa, G. Xu, and B. Li, "A Novel Pre-Computation Scheme of Window τNAF for Koblitz Curves", IACR Cryptology ePrint Archive, Report 2017/1020, https://eprint.iacr.org/2017/1020, 2017.

[44] C. Zhao, F. Zhang, and J. Huang, "Efficient Tate Pairing Computation Using Double-Base Chains," *Science in China Series F: Information Sciences*, vol. 51, no. 8, pp. 1096-1105, 2008.

[45] Zilliqa. https://zilliqa.com/.