Performance of EdDSA and BLS Signatures in Committee-Based Consensus

Zhuolun Li
University of Leeds
sczl@leeds.ac.uk

Alberto Sonnino
Mysten Labs
University College London (UCL)
alberto@mystenlabs.com

Philipp Jovanovic
University College London (UCL)
p.jovanovic@ucl.ac.uk

ABSTRACT

We present the first performance comparison of EdDSA and BLS signatures in committee-based consensus protocols through large-scale geo-distributed benchmarks. Contrary to popular beliefs, we find that *small* deployments (less than 40 validators) can benefit from the small storage footprint of BLS multi-signatures while larger deployments should favor EdDSA to improve performance. As an independent contribution, we present a novel way for committee-based consensus protocols to verify BLS multi-signed certificates by manipulating the aggregated public key using pre-computed values.

CCS CONCEPTS

• **Security and privacy** → **Digital signatures**; • **Networks** → **Network performance analysis**.

KEYWORDS

Digital signature, Consensus, Blockchain

ACM Reference Format:
Zhuolun Li, Alberto Sonnino, and Philipp Jovanovic. 2023. Performance of EdDSA and BLS Signatures in Committee-Based Consensus. In *The 5th workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED 2023), June 19, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3584684.3597265

1 INTRODUCTION

Consensus protocols run at the core of blockchains to order clients' transactions into a sequence agreed by all honest validators. The popularity of blockchains raised the interest in developing high-performance consensus systems, with early studies proposing committee-based protocols to improve over Bitcoin's [27] low throughput of 7 transactions per second. These protocols have since been shown to increase blockchain throughput and reduce latency [2, 22], and they are rapidly becoming the standard in recent proof-of-stake architectures [11, 24, 25]. The blockchain literature provides a large variety of efficient committee-based consensus protocols. They improve on the state-of-the-art through various techniques, ranging from efficient data-sharing layers [10, 30] and robust DAG-based protocols [14, 28] to multi-mode protocols adapting to faults and network conditions [15, 18, 23, 26].

Unfortunately, these works pay little attention to their choice of signature scheme. Our inspection of their codebases indicates that most use either EdDSA [10, 11, 24, 25, 28] or BLS [14, 19, 20, 26] but do not justify their choice. This is unfortunate as digital signatures require CPU-intensive operations and are extensively used in committee-based consensus: block proposers authenticate their block proposals by signing them, validators counter-sign block proposals to indicate their support, and certificates containing a quorum of signatures are used to commit and finalize transactions. On the one hand, EdDSA signatures provide very fast signature generation and verification; on the other hand, BLS multi-signing enables small certificates and nearly constant-time certificate verification regardless of the committee size. It is a popular belief that BLS is preferable to EdDSA for large deployments where large EdDSA certificates are slow to propagate and verify. There is, however, no empirical evidence supporting this belief.

We address this gap by providing the first performance comparison of EdDSA and BLS signatures in committee-based consensus (to the best of our knowledge). We demonstrate through large-scale geo-distributed benchmarks that the choice of the signature scheme is a major factor determining the system's performance. We find that contrary to popular beliefs, deployments with a relatively *small* committee size (less than 40 validators) can benefit from the small storage footprint of BLS multi-signatures while larger deployments should favor EdDSA to improve performance. In a nutshell, the computational overhead of BLS verification becomes prohibitive when validators verify a large number of counter-signed block proposals; at the point where it offsets the benefits of small and efficient certificates. We select HotStuff [31] as an example of a committee-based consensus protocol for our experiments. As an independent contribution, we present a novel way for committee-based consensus protocols to verify BLS multi-signed certificates by pre-computing a fixed number of group elements to manipulate the aggregated public key. This technique outperforms a traditional BLS verification process (even in the presence of Byzantine faults) requiring to re-compute the appropriate aggregated public key upon each certificate verification.

Contributions. In summary, we make the following contributions:

- We perform the first performance comparison of EdDSA and BLS signatures in committee-based consensus (to the best of our knowledge) through large-scale geo-distributed benchmarks.
- We analyze the performance implications of each signature scheme and identify that small deployments are most suited

ApPLIED 2023, June 19, 2023, Orlando, FL, USA

ACM ISBN 979-8-4007-0128-3/23/06.
https://doi.org/10.1145/3584684.3597265

to take advantage of BLS multi-signatures while larger deployments should favor EdDSA.
- We present a novel and more efficient way for committee-based consensus protocols to take advantage of BLS multi-signatures to verify certificates.

2 BACKGROUND

We recall BLS multi-signatures and provide an overview of typical committee-based consensus protocols.

BLS multi-signatures. The Boneh-Lynn-Shacham (BLS) signature scheme [4] is an efficient signature scheme using pairing-friendly elliptic curves. BLS supports multi-signing and public-key aggregation, making it very popular for various blockchain projects. We start by recalling the standard BLS signature scheme. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be groups of prime order q such that there exists an efficiently computable and non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We denote by g_1, g_2, and g_T the canonical generators of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, respectively, and let $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_1$. We denote the security parameter by λ and $\xleftarrow{\$}$ denotes sampling uniformly at random. The BLS signature scheme consists of the following algorithms:

- **BLS.Setup**(1^λ): Setup and output a bilinear group $par = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$.
- **BLS.KeyGen**(par): Given the parameters par, output a pair of public/secret keys $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_q^*$ and $\mathsf{pk} = g_2^{\mathsf{sk}} \in \mathbb{G}_2$.
- **BLS.Sign**(par, sk, m): Given a message $m \in \{0,1\}^*$, output a signature $\sigma = \mathsf{H}_1(m)^{\mathsf{sk}} \in \mathbb{G}_1$.
- **BLS.Verify**$(par, \mathsf{pk}, \sigma, m)$: Given a public key $\mathsf{pk} \in \mathbb{G}_2$, a signature $\sigma \in \mathbb{G}_1$ and a message $m \in \{0,1\}^*$, output 1 if $e(\sigma, g_2) = e(\mathsf{H}_1(m), \mathsf{pk})$ and 0 otherwise.

BLS signatures can support multi-signing with public key aggregation. A multi-signature scheme (MSP) allows n signers to generate a short signature σ, on the *same* message m (the size of the signature is independent of the number of signers). To verify the multi-signature one needs all the signer's public keys aggregated into a single key apk, m, and σ.

- **MSP.Setup**(1^λ): Output **BLS.Setup**(1^λ).
- **MSP.KeyGen**(par): Output **BLS.KeyGen**(par).
- **MSP.Sign**(par, sk, m): Output **BLS.Sign**(par, sk, m).
- **MSP.SigAggr**$(\{\sigma_1, \ldots, \sigma_n\})$: Output $\sigma = \prod_{i=1}^{n} \sigma_i$.
- **MSP.KeyAggr**$(par, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\})$: Output $apk = \prod_{i=1}^{n} \mathsf{pk}_i$.
- **MSP.Verify**(par, apk, σ, m): Output **BLS.Verify**(par, apk, σ, m).

An MSP scheme is usually vulnerable to rogue key attacks if the signers do not prove their ownership of the corresponding secret keys [3]. This issue does not pose a challenge for committee-based blockchains, as validators are required to demonstrate knowledge of their secret key sk by signing a specific transaction prior to becoming eligible to join a future committee.

We note that threshold signatures are a generalization of multi-signatures requiring only a threshold $t \leq n$ of signatures (rather than $t = n$) to compute σ. Threshold signatures require complex distributed key generation mechanisms to generate the key pair of every signer when there is no natural trusted third party to run the setups protocol [1, 17, 21]. Furthermore, they do not extend naturally to settings where validators have different voting powers and are thus unsuitable for consensus protocols. As a result, this paper does not consider them as no committee-based blockchains deploy them within their consensus protocol (to the best of our knowledge).

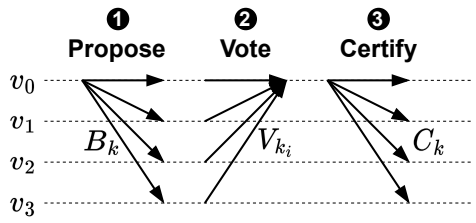

Figure 1: High-level overview of one round of a typical committee-based consensus protocol. The committee is formed of four validators (v_0, v_1, v_2, v_3). The leader v_0 proposes B_k for round k; validators reply with a vote V_k over B_k; the leader collects a quorum of votes into a certificate C_k and disseminates it, and validators verify C_k.

Committee-based consensus. Committee-based blockchains typically divide time into a sequence of epochs (lasting roughly a day [24, 25, 29]). They elect a committee of $n = 3f + 1$ validators for each epoch (usually through proof-of-stake [24, 25]), where f is the maximum number of faulty validators that the system can tolerate. The elected committee then 'extends' the blockchain by sequencing clients' transactions using a Byzantine fault tolerant (BFT) protocol.

For the sake of this paper, we only present the aspects of committee-based consensus protocols where signatures intervene the most (and omit other aspects such as synchronizers [8], leader-election modules [7], and view-changes [5, 15]). Typical committee-based consensus protocols operate in a round-by-round manner, electing a leader in each round among the validators to balance validator participation. Figure 1 provides a high-level overview of one round of a typical committee-based consensus protocol running with four validators, v_0, v_1, v_2, and v_3. The leader v_0 of round k disseminates a block B_k extending the longest chain of blocks it knows[1] (❶). Validators then vote for at most one leader's proposal for each round by counter-signing it unless the proposal is malformed or conflicts with a longer chain that they know (❷); validators send their votes V_k back to the leader. The leader aggregates a quorum of $2f + 1$ votes into a certificate C_k and distributes it to the validators; validators accept C_k if it is correctly signed by a quorum (❸).

The protocol then repeats for several rounds (usually two or three) in order to commit B_k. For instance, the original HotStuff protocol [31] commits B_k when there exist three consecutive certified blocks in the chain, C_k, C_{k+1}, C_{k+2}. More recent variants of HotStuff, such as Jolteon [15], only require two consecutive rounds; and state-of-the-art DAG-based protocols [10, 14, 28] allow multiple validators to disseminate proposals in parallel. The general protocol

[1] Usually leaders collect batches of transactions to propose, referred to as blocks, hence the protocol forms a chain of blocks (or a 'blockchain').

flow, however, remains similar. If the leader fails or is unresponsive for a long period of time, the validators run a *view-change* sub-protocol to elect a new leader [5]; changing leaders are expensive and severely degrades performance.

The key motivation for BLS multi-signatures in committee-based consensus is to reduce the size of certificates (which grow linearly with the committee size) and allow their nearly constant-time verification (by verifying a single multi-signature rather than the $2f+1$ votes individually).

3 CACHED BLS MULTI-SIGNATURE VERIFICATION

We extend the BLS multi-signature scheme presented in Section 2 with a new function **MSP.KeyDisAggr** that subtracts a set of public keys from the aggregate public key apk.

- **MSP.KeyDisAggr**$(par, apk, \{-\mathsf{pk}_1, \ldots, -\mathsf{pk}_n\})$: Given the aggregate public key apk, the opposite of all public keys $PK = \{-\mathsf{pk}_1, \ldots, -\mathsf{pk}_n\}$, output $apk^* = apk \prod_{i=1}^{j} -\mathsf{pk}_i$.

This function requires a single elliptic curve addition per key to remove from the aggregate. We now show how to incorporate it in the normal flow of committee-based consensus protocols depicted in Figure 1 of Section 2.

Protocol description. Every validator in the committee is initialized with the public parameters par output by **MSP.Setup**(1^λ). Each validator locally runs **MSP.KeyGen**(par) to generate their public/secret keypair $(\mathsf{pk}, \mathsf{sk})$ and publishes pk (see Section 2). Each validator stores the public key $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ of all other valiators. They also compute and store the aggregated public key apk = **MSP.KeyAggr**$(par, \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\})$ as well as the opposite[2] of all validator's public keys $\{-\mathsf{pk}_1, \ldots, -\mathsf{pk}_n\}$.

- **Step ❶: Propose.** The leader of round k collects a set of clients' transactions l and creates a block proposal $m = (k, l, \cdot)$, where the dot '$\cdot$' denotes omitted protocol-specific fields. The leader then signs m by calling σ_B = **MSP.Sign**(par, sk, m) using its secret key sk and disseminates $B_k(m, \sigma_B)$ to the other validators.
- **Step ❷: Vote.** Validators first parse $B_k = (m, \sigma_B)$ and then verify it via **MSP.Verify**$(par, \mathsf{pk}, \sigma_B, m)$ where pk is the leader's public key. If the check passes and all other protocol-specific conditions are met, they counter-sign B_k and send their vote V_k = **MSP.Sign**$(par, \mathsf{sk}, H(B_k))$ to the leader (where H is a collision-resistant hash-function).
- **Step ❸: Certify.** The leader verifies each incoming vote by calling **MSP.Verify**$(par, \mathsf{pk}, V_k, H(B_k))$, where pk is the public key of the voter. As soon as it receives $2f+1$ valid votes $\{V_{k_1}, \ldots, V_{k_n}\}$, it aggregates them calling σ_C = **MSP.SigAggr**$(\{V_{k_1}, \ldots, V_{k_n}\})$. It then computes a bitmap b indicating which validators *did not* contribute to σ_C. This is achieved by deterministically attributing an index to each validator that corresponds to its position in the bitmap. This bitmap allows to reduce the size of the certificate that would otherwise contain the public key of each signer. The leader then disseminates the certificate $C_k = (\sigma_C, b)$ to the validators. Upon receiving C_k, validators use the bitmap b to identify the validators who did not contribute to σ_C, retrieve their previously cached set of $\{-\mathsf{pk}_1, \ldots, -\mathsf{pk}_j\}$, and compute apk^* = **MSP.KeyDisAggr**$(par, apk, \{-\mathsf{pk}_1, \ldots, -\mathsf{pk}_n\})$. They then verify the certificate calling **MSP.Verify**$(par, apk^*, \sigma_C, H(B_k))$.

This approach has two main advantages. (i) The bitmap allows reducing the certificate size by 32B per signer compared to straightforward implementations including the public key of each signer in the certificate (as it is the case in many production systems [11, 24, 25]); certificates are part of the forever-stored blockchain so any message compression becomes substantial over time. (ii) Before verifying a certificate validators compute apk^* with at most f elliptic curve additions, while a straightforward BLS multi-signature verification would require $2f+1$ to recompute the aggregated verification key every time. Furthermore, practical leaders' implementations wait around 50-100ms after collecting the first $2f+1$ votes to give extra time to the remaining validators to vote. As a result, typical certificates contain close to $n = 3f+1$ votes in the common case (happy path), and computing apk^* requires only one or two elliptic curve additions.

4 PERFORMANCE COMPARISON

We select HotStuff [31] as an example of a committee-based consensus protocol for our experiments. We select this protocol because it is the quorum-based consensus protocol most used in production blockchains; Celo [6], Cypherium [9], Flow [13], Diem [11], and Aptos [24] all run a variant of HotStuff. Furthermore, it shares many design traits with Tendermint [29] (its closest ancestor). We specifically run our benchmarks on a 2-chain HotStuff variant called Jolteon [15]; we chose this variant because Diem [11], Aptos [24], and Flow [13] run it in production.

Implementation. We implement BLS multi-signatures on top of the original open-source implementation of Jolteon[3]. It is implemented in Rust, uses Tokio[4] for asynchronous networking, ed25519-dalek[5] for signatures, and data-structures are persisted using RocksDB[6]. It uses TCP to achieve reliable point-to-point channels, necessary to correctly implement the distributed system abstractions. Since this implementation uses EdDSA (over the curve Ed25519), we modify its `crypto` module to use BLS multi-signatures as described in Section 3. We use the BLS implementation of Filecoin (over the curve BLS12-381)[7]with modifications to realize the cached BLS multi-signature scheme we propose. The implementation of both signature schemes is designed to achieve 128-bit security. We are open-sourcing our BLS-enabled implementation[8].

Evaluation setup. We evaluate the throughput and latency of HotStuff/Jolteon equipped with BLS multi-signatures through experiments on Amazon Web Services (AWS). We then compare its performance with the baseline implementation using EdDSA for various committee sizes.

[2]The opposite of a public key is obtained by negating the y-axis value, i.e., (x, y) becomes $(x, -y)$.

[3]https://github.com/asonnino/hotstuff
[4]https://tokio.rs
[5]https://github.com/dalek-cryptography/ed25519-dalek
[6]https://rocksdb.org
[7]https://github.com/filecoin-project/bls-signatures
[8]https://github.com/radiken/hotstuff-digital-signature-benchmarking

We deploy a testbed on AWS, using `t3.medium` instances across 4 different AWS regions: N. Virginia (us-east-1), N. California (us-west-1), Sydney (ap-southeast-2), and Frankfurt (eu-central-1). Validators are distributed across those regions as equally as possible. The selection of regions in the experiment emulates a geographically sparse distributed network that closely resembles the distribution of existing blockchain nodes [9]. Each machine provides up to 5 Gbps of bandwidth, 2 virtual CPUs (1 physical core) on a 2.5 GHz, Intel Xeon Platinum 8175, 4 GB memory, and runs Linux Ubuntu server 20.04. In the following sections, each measurement in the graphs is the average of 5 independent runs, and the error bars represent one standard deviation. Our baseline experiment parameters are a block size of 500KB, a transaction size of 512B, and one benchmark client per party submitting transactions at a fixed rate for 5 minutes. The leader timeout value is set to 5 seconds. When referring to *latency*, we mean the time elapsed from when the client submits the transaction to when the transaction is committed by one validator. We measure it by tracking sample transactions throughout the system. For each committee size configuration, we gradually increase the input rate until we observe fluctuations in latency or reductions in throughput, which are indicative of the system reaching its capacity limits.

Analysis. Figure 2 and Figure 3 show that a 4-validators deployment can process 50,000 tx/s while keeping the latency below 1.5 seconds, regardless of the signature scheme. Similarly, prior to reaching system capacity, 20- and 40-validator deployments can process around 80,000 tx/s while keeping the latency around 4 to 5 seconds, regardless of the signature scheme[10].

Increasing the committee size to 60 validators drops the performance of our EdDSA-based implementation to around 60,000 tx/s and increases the latency to 5 seconds (Figure 2). This performance drop is explained by both the additional bandwidth required to broadcast messages to many validators and the CPU overhead required to verify a large number of signatures. Indeed, the leader needs to verify at least $2f + 1 = 41$ votes every round (step ❷ of Figure 1) and validators need to verify 41 signatures to validate each certificate[11] (step ❸ of Figure 1). Our BLS-based implementation suffers a more significant performance drop: Figure 3 indicates the system can only process up to 20,000 tx with a latency of about 15 seconds. It appears that this performance difference is due to the time required by the leader to verify individual votes (step ❷ of Figure 1). EdDSA allows the leader to efficiently verify votes, while BLS verification is about 100x slower and monopolizes the leader's CPU. This causes the leader's slow down, which affects both throughput and latency.

Figure 2 indicates that even larger deployments of 80 validators further drop the performance of the EdDSA-based implementation to about 40,000 tx/s (with a latency of 6 seconds). Figure 3 shows that our BLS-based implementation barely manages to process

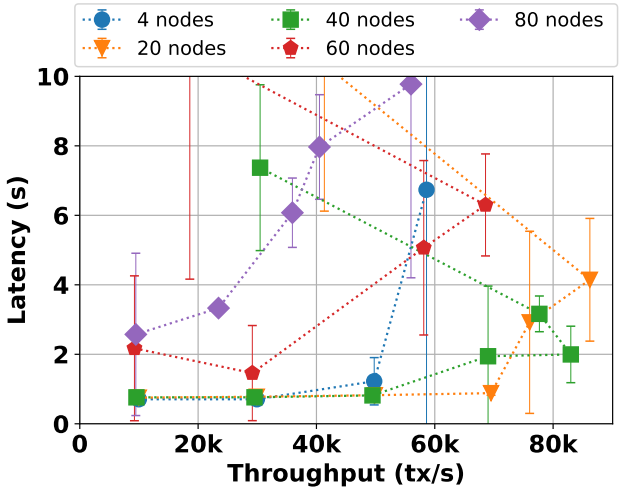

Figure 2: EdDSA-based implementation for 4, 20, 40, 60, and 80 validators over a WAN.

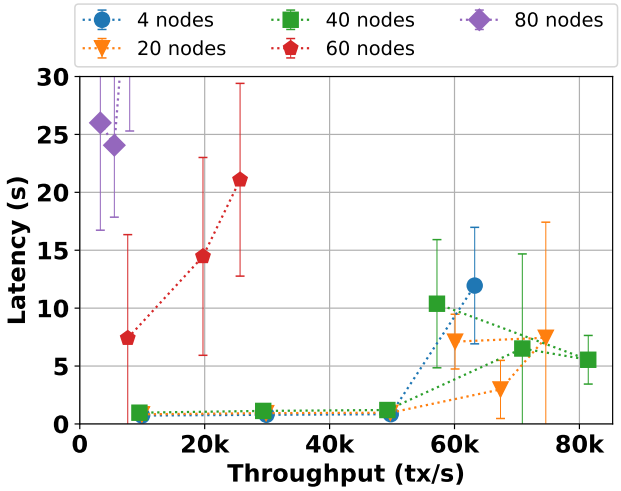

Figure 3: BLS-based implementation for 4, 20, 40, 60, and 80 validators over a WAN.

transactions; it can only process a few thousand tx/s with a latency of over 25 seconds. The time required by the leader to verify votes exceeds 5 seconds most of the time (the leader-timeout value), at which point validators believe the leader crashed and initiate a view-change sub-protocol to elect a new leader; this scenario repeats often and greatly degrades performance.

Key takeaways. Our experiments demonstrate no apparent performance benefit of BLS signatures. Contrary to popular belief, large deployments do not benefit from the aggregation properties of BLS. Despite BLS multi-signatures enabling small certificates and nearly constant-time certificate verification (regardless of the committee size), the CPU overhead of individual votes verification greatly offsets this benefit. As a result, large deployments should favor EdDSA. Small deployments (up to 40 validators) do not place an

[9] In May 2023, approximately half of the Ethereum nodes are located in North America; the remaining nodes are distributed across Europe and Asia Pacific [12].
[10] It may seem surprising that the system achieves a higher throughput with 20 and 40 validators than with 4. This is, however, a known result [10, 15], the extra capacity provided by the additional validators allows for better resource utilization. To the best of our knowledge, the reason for this behaviour of multi-core consensus systems is still unknown and an open research problem.
[11] The 'batch-verify' feature of EdDSA greatly speeds up certificate verification.

excessive CPU burden on the leader, EdDSA and BLS-based deployments perform similarly and they may thus take advantage of the aggregation properties of BLS. Small BLS multi-signed certificates can provide a significant storage benefit: storing an EdDSA certificate requires 2.5 KB (for a 40 nodes deployment) while a BLS multi-signed certificate only requires about 100 B. This difference may become significant over time since certificates are part of the forever-persisted blockchain.

5 FUTURE WORK

In a concurrent work to ours, Gelbmann [16] investigated the performance of BLS aggregate signatures over gossip networks. A potential future research direction is to connect Gelbmann's evaluation, which focuses on a lower layer compared to a consensus protocol, with our work. This would provide a more comprehensive understanding of the performance of BLS signatures in a decentralized network, allowing for a decomposition of the factors influencing their effectiveness.

Building upon the contributions from [10, 15] and the present paper, a potential direction for future research involves exploring the hardware bottlenecks of multi-core consensus systems. Experimental results have shown that increasing the committee size in small-scale systems (from 4 to 40 nodes in our configurations) enhances performance. However, the specific bottlenecks that impede such improvement in larger-scale systems remain unidentified. Investigating these bottlenecks could pave the way for efficient strategies to enhance the performance of multi-core consensus systems.

Another potential direction for future work involves reducing the storage footprint of blockchains. While recent research on committee-based consensus tends to focus on enhancing computational performance in consensus protocols, less attention has been given to improving storage efficiency. Our results demonstrate that BLS signatures effectively compress information for authentication without compromising throughput and latency in small-scale consensus systems. However, in a wider scope, methods for information compression and its trade-off between computational efficiency in blockchains have not been thoroughly investigated. The significance of such studies will only increase over time in the context of ever-growing blockchains.

ACKNOWLEDGEMENTS

This work is partially funded by Mysten Labs.

REFERENCES

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2022. Bingo: Adaptively Secure Packed Asynchronous Verifiable Secret Sharing and Asynchronous Distributed Key Generation. Cryptology ePrint Archive, Paper 2022/1759. https://eprint.iacr.org/2022/1759

[2] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2017. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936* (2017).

[3] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II*. Springer, 435–464.

[4] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International conference on the theory and application of cryptology and information security*. Springer, 514–532.

[5] Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In *OsDI*, Vol. 99. 173–186.

[6] Celo. 2023. Build Together and Prosper. https://celo.org.

[7] Shir Cohen, Rati Gelashvili, Lefteris Kokoris Kogias, Zekun Li, Dahlia Malkhi, Alberto Sonnino, and Alexander Spiegelman. 2022. Be aware of your leaders. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*. Springer, 279–295.

[8] Shir Cohen, Guy Goren, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Proof of Availability & Retrieval in a Modular Blockchain Architecture. *Cryptology ePrint Archive* (2022).

[9] Cypherium. 2023. Web3-Ready Blockchain. https://www.cypherium.io.

[10] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.

[11] Diem. 2023. Welcome to the Diem Project. https://www.diem.com.

[12] ethernodes.org. 2023. Countries - ethernodes.org - The Ethereum Network & Node Explorer. https://www.ethernodes.org/countries

[13] Flow. 2023. Build Powerful, Secure, and Scalable Web3 Apps. https://flow.com.

[14] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1187–1201.

[15] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2021. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. *arXiv preprint arXiv:2106.10362* (2021).

[16] Lukas Gelbmann. [n. d.]. BLS Cosigning via a Gossip Protocol. ([n. d.]).

[17] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 295–310.

[18] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. 2010. The next 700 BFT protocols. In *Proceedings of the 5th European conference on Computer systems*. 363–376.

[19] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding dumbo: Pushing asynchronous bft closer to practice. *Cryptology ePrint Archive* (2022).

[20] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 803–818.

[21] Aniket Kate, Yizhou Huang, and Ian Goldberg. 2012. Distributed key generation in the wild. *Cryptology ePrint Archive* (2012).

[22] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. *CoRR* abs/1602.06997 (2016). arXiv:1602.06997 http://arxiv.org/abs/1602.06997

[23] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 45–58.

[24] Aptos Labs. 2023. Committed to developing products and applications on the Aptos blockchain that redefine the web3 user experience. https://aptoslabs.com.

[25] Mysten Labs. 2023. Build without boundaries. https://sui.io.

[26] Yuan Lu, Zhenliang Lu, and Qiang Tang. 2022. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2159–2173.

[27] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review* (2008), 21260.

[28] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2718.

[29] Tendermint. 2023. Building the most powerful tools for distributed networks. https://tendermint.com.

[30] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2021. DispersedLedger: High-Throughput byzantine consensus on variable bandwidth networks. *arXiv preprint arXiv:2110.04371* (2021).

[31] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 347–356.