Which Signature Scheme to Pick for Your Distributed Protocol?

Aniket Kate, Pratyay Mukherjee, Hamza Saleem, and Nibesh Shrestha

Supra Research

Abstract

Digital signatures are fundamental to secure, large-scale distributed computing systems. In these systems, groups of nodes sign messages, and the resulting signatures are aggregated for efficient verification in future communication rounds. EdDSA/ECDSA signatures are often chosen for their low computation costs, though they can impose higher demands on bandwidth and storage. Multi-signature and threshold-signature schemes, while more space-efficient, involve higher computational costs for creation and verification, which can restrict their scalability. However, with careful optimizations, these schemes can be effectively adapted for use in larger settings.

As distributed systems strive for low latency and high throughput, even with hundreds of nodes, carefully selecting the appropriate signature scheme becomes essential for performance. This work seeks to provide a thorough framework for choosing signature schemes that best suit high-performance distributed computing systems. Drawing from extensive experiments involving hundreds of cloud-based nodes, we present a practical guide for selecting signature schemes tailored to various communication patterns in distributed protocols.

1 Introduction

Signatures are indispensable in distributed systems, supporting applications from reliable broadcasts to secure multi-party computations. They serve several key roles: authenticating the sender's identity to ensure message integrity, enabling accountability by tracking message sources in cases of misbehavior, and providing public verifiability for critical operations like consensus agreement. As the cornerstone of trust, security, and efficiency, especially in blockchain systems, signatures are vital to decentralized operations. Selecting an appropriate signature scheme is therefore crucial for scalability, as it directly influences both the performance and the ability of distributed applications to expand to large networks.

Numerous signature schemes [9, 2, 12] have been proposed for distributed applications. In this work, we focus on Byzantine Fault Tolerant (BFT) consensus, which is fundamental to blockchain systems. Within BFT consensus protocols, three primary signature schemes are widely considered: (centralized) EdDSA/ECDSA signatures, threshold BLS signatures, and BLS multi-signatures.[1] These signature types offer different trade-offs that impact the performance and efficiency of BFT consensus protocols.

From a performance perspective, EdDSA/ECDSA signatures are computationally efficient to generate and verify, as they primarily involve less expensive curve operations. In contrast, BLS signatures require more complex operations over pairing-friendly curves which are computationally intensive. We provide a detailed breakdown of the various cryptographic operations for these signature schemes in Table 1. On e2-standard machine types in GCP[16], creating an EdDSA signature takes approximately 0.026 ms, while verifying it takes around 0.0721 ms. In comparison, a BLS signature requires about 0.205 ms for creation and 1.8982 ms for verification.

While EdDSA/ECDSA signatures are computationally efficient, they either lack aggregation capabilities. In many consensus protocols, nodes typically sign a common message and collect a quorum of signatures to

[1] In this document, whenever we refer to EdDSA/ECDSA, we mean the non-threshold variant, unless mentioned otherwise explicitly.

form a certificate. In the case of EdDSA/ECDSA, a certificate consists of a linear number of signatures since these signatures cannot be aggregated. Most consensus protocols involve a step where a designated node, or all nodes, multicast a certificate in each round of the protocol. Given that a single EdDSA signature is 64 bytes, the size of the certificate reaches at least 12.8 KB even with a moderate system size of 300 nodes. Consequently, multicasting these certificates would result in a minimum bandwidth consumption of 3.84 MB per round per node, which is substantial. This high bandwidth demand increases data transfer and storage costs, as certificates must also be retained for future use.

Table 1: Number of cryptographic operations for various signature schemes

Scheme	**Create**	**Verify (Individual)**	**Aggregation**	**Verify (Aggregated)**
EdDSA	$1\mathbb{G}$ mult	$2\mathbb{G}$ mult	-	-
BLS multi-sigs	$1\mathbb{G}_1$ mult + H2C	2 pairings + H2C	$(n-1)$ $\mathbb{G}_1$ add + H2C	$(n-1)$ $\mathbb{G}_2$ add + 2 pairings + H2C
BLS threshold	$1\mathbb{G}_1$ mult + H2C	2 pairings + H2C	n $\mathbb{G}_1$ mult + $(n-1)$ $\mathbb{G}_1$ add + H2C	2 pairings + H2C

We list only the computationally expensive operations, such as group operations and hash-to-curve operations, for simplicity, as scalar operations are comparatively inexpensive. Here, *H2C* denotes a hash-to-curve operation, *mult* represents multiplication, and *add* represents addition.

In contrast, BLS threshold and multi-signatures can be aggregated into a compact signature, offering significant space efficiency. Specifically, a BLS threshold signature is $O(k)$ in size, while a BLS multi-signature is $O(k+n)$, where k is the security parameter and n is the number of signers. Concretely, a BLS threshold signature is 48 bytes, and a BLS multi-signature is approximately 86 bytes—significantly smaller than an EdDSA certificate for a network of 300 nodes. Despite this spatial efficiency, BLS signatures are computationally expensive to create, verify, and aggregate, with verification and aggregation often being the most resource-intensive operation, which can limit their applicability in larger systems. Recent research supports this, indicating that BLS multi-signatures are generally unsuitable for systems with more than 40 nodes. Specifically, a study on the performance of EdDSA and BLS multi-signatures in the HotStuff [21] consensus protocol found that while BLS multi-signatures performed well in systems with up to 40 nodes, their performance degraded significantly at larger scales, whereas EdDSA remained more computationally efficient even as the system size increased [14].

Towards optimistic verification of BLS signatures. In a typical application of the BLS signature scheme, nodes sign a common message to generate individual signature shares, which are then sent to a designated node or broadcast to all nodes. The receiving nodes usually verify each individual signature share before aggregating them to form an aggregated signature once a quorum is reached. However, as mentioned, the verification of BLS signatures is a significant bottleneck as the system scales. To address this, we aggregate individual signatures without upfront individual verification, opting instead to verify only the aggregated signature. If the aggregated signature fails due to a malicious party's incorrect contribution, individual signatures are then verified (in parallel) to identify and penalize the faulty party. This approach bypasses the overhead of individual signature verification in the optimistic (and typical) case where all nodes sign correctly, enhancing efficiency.

To demonstrate the practical efficacy of this optimization, we implemented it on the Moonshot consensus protocol [8]. Moonshot, at a high level, supports block proposals in every round (in contrast to every two rounds in previous protocols [21, 11]) and commits the proposed block within three rounds. We integrated both EdDSA and BLS multi-signatures (with signatures in group $\mathbb{G}_1$) into Moonshot and observed that, even with a system size of 300 nodes, the performance of the BLS multi-signature implementation is only slightly slower than the EdDSA-based implementation, as illustrated in Figure 1. This highlights the practicality of the approach, offering reduced bandwidth and storage requirements. Additionally, we note that this practical method has been adopted in production implementations of consensus protocols [13].

This optimization expands the applicability of BLS signatures in distributed protocols, even at large scales. However, BLS threshold and multi-signatures present distinct trade-offs in practice, raising a key question: *Which BLS signature scheme is best suited for a given distributed protocol?* In this work, we

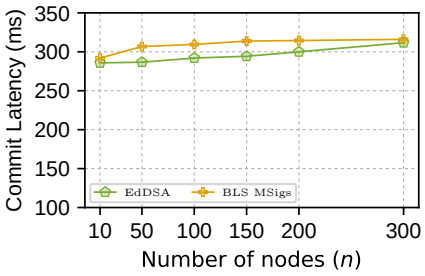

Figure 1: Commit latencies for Moonshot [8] at different system sizes utilizing EdDSA and BLS multi-signatures.

examine four common communication patterns in distributed protocols and show that, depending on the communication pattern, either BLS multi-signatures or BLS threshold signatures may be preferable.

Comparison of BLS threshold and multi-signatures. Both signature schemes are comparable in terms of creating and verifying individual signature shares, as shown in Table 1. Regarding size, threshold signatures are compact, requiring only $O(\kappa)$ bits, while multi-signatures need $O(\kappa+n)$ bits, growing linearly with system size (due to the storage requirement of all public keys). However, since this growth occurs only in bits, multi-signatures remain sufficiently compact even in large systems, without significantly impacting bandwidth or communication overhead.

In terms of setup, BLS threshold signatures typically require an initial distributed key generation (DKG) phase, which is more complex.[2] State-of-the-art DKG protocols [7, 6] generally necessitate at least $O(\kappa n^3)$ communication and multiple rounds. Importantly, this setup is typically performed only once at the start of the protocol.

For aggregating individual signature shares, BLS threshold signatures require Lagrange interpolation to produce a single threshold signature, which is significantly more computationally intensive than the simple aggregation process used to generate a multi-signature. Table 1 outlines the cryptographic operations involved in these aggregations, and Figure 2 shows the time required to create both BLS threshold signatures and multi-signatures across system sizes up to $n = 1000$. These evaluations were conducted on e2-standard machine types in GCP [16], assessing both variants in both the groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Generally, operations in group $\mathbb{G}_1$ are more efficient than those in $\mathbb{G}_2$, making it preferable to use signatures in $\mathbb{G}_1$ for better performance.

Verifying a threshold signature necessitates only two pairing operations. In contrast, multi-signature verification involves each verifier first computing an aggregate public key by combining the public keys of all nodes whose individual signatures are part of the multi-signature. This is followed by verifying the aggregated multi-signature against the computed aggregate public key, which also requires two pairing operations. The process of computing the aggregate public key can be optimized by initially calculating a complete aggregate public key for all participating nodes and then subtracting the public keys of the nodes whose signatures are not included in the multi-signature, for example, for high threshold settings. Figure 2 illustrates the time required to aggregate n public keys for system sizes up to $n = 1000$.

From the preceding discussion, it is clear that the aggregation phase of BLS threshold signatures is considerably expensive, whereas the signature verification phase of BLS multi-signatures is relatively costly. Consequently, the selection of the signature scheme hinges on the frequency of these operations within the protocol.

We examine four common communication patterns in distributed algorithms and recommend appropriate signature schemes for each. These patterns are categorized based on the number of proposers and the message complexity within the algorithm.

[2]While recent works [10, 5] eliminate DKG in the threshold BLS setting, they rely on SNARK proofs for compactness requiring high aggregator time.

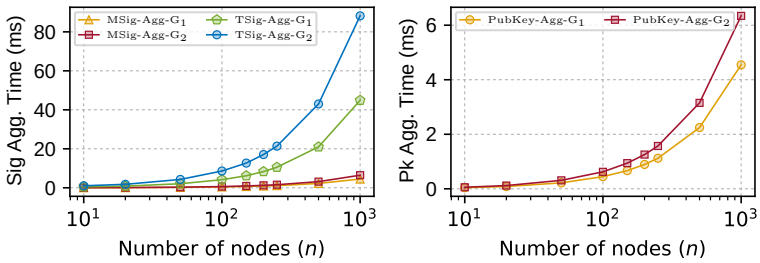

Figure 2: The performance of BLS signatures across various system sizes. The left figure illustrates the time required to create threshold signatures and multi-signatures, with the signatures represented in group $\mathbb{G}_1$ and $\mathbb{G}_2$. The right figure displays the time taken to aggregate public keys when the keys are in $\mathbb{G}_1$ and $\mathbb{G}_2$.

Table 2: Number of signature aggregation and verification operations for various communication patterns

Pattern	**Num of operations** (per party)	**Preferred scheme**
SPLM	1 agg. + 1 verify (aggregated)*	Multi sigs
SPQM	1 agg. + 1 verify (aggregated)	Multi sigs
MPQM	1 agg. + n verify (aggregated)	Threshold sigs
MPCM	n agg. + n verify (aggregated)	Multi sigs

Here, agg refers to aggregations. * In this pattern, a single party aggregates, and all parties verify the aggregated signature.

Single proposer linear messages. In this communication pattern, a single proposer sends a common message to all nodes, each of which generates an individual signature share by signing the message and then sends this share to a designated node. This designated node is responsible for verifying the individual signature shares and combining them into a single aggregated signature. Subsequently, the designated node multicasts the aggregated signature to all other nodes. As noted above, there is an optimization, in that individual signature shares can be aggregated without upfront verification, which helps reduce the overhead associated with the costly verification of each share during a typical operation, when the final verification succeeds. Once the aggregation is complete, the aggregator verifies the aggregated signature and multicasts it to the rest of the system, where all other nodes perform verification solely on the aggregated signature. See Figure 3 for a pictorial representation.

Many consensus protocols, such as HotStuff [21], Jolteon [11], and HotStuff-2 [15], adopt this communication pattern to achieve linear message complexity (and linear communication complexity when using threshold signatures). Since this pattern depends on a single aggregator, it is crucial to minimize the computational workload at this node to ensure a quicker dissemination of the aggregated signature to other nodes. As illustrated in Figure 2, the overhead associated with aggregating signatures using BLS multi-signatures is considerably lower across all system sizes compared to BLS threshold signatures. Although multi-signatures necessitate all verifiers to compute the aggregated public keys to verify the multi-signature, the cost of public key aggregation is significantly less than the that of aggregating a threshold signature (due to the Lagrange coefficients). Therefore, for this communication pattern, BLS multi-signatures demonstrate superior performance over threshold signatures.

Single proposer quadratic messages. In this communication pattern, a single proposer sends a common message to all nodes. Each node then signs the message to generate an individual signature share and multicasts this share to all nodes in the system. Each node is responsible for aggregating a quorum of these signature shares into a single aggregated signature, which it then verifies (using optimistic verification). Typically, this aggregated signature is also sent to all nodes to ensure that any node that has not received a quorum of individual signature shares in a timely manner can verify the aggregated signature. This

approach allows nodes to receive the aggregated signature in two communication steps rather than three, as in the previous approach. This pattern is commonly used in latency-optimal consensus protocols, including PBFT [3], Simplex [4], and Moonshot [8]. A pictorial representation is presented in Figure 4.

In this setting, each node only needs to verify a single signature. When verifying a single signature, it is computationally cheaper to aggregate the individual signatures and public keys than to perform Lagrange interpolation to obtain a threshold signature, regardless of system size. Therefore, BLS multi-signatures are preferable for this setting as well.

Multi-proposers quadratic messages. In this pattern, there are n distinct proposers, each proposing a unique message. Each proposer sends its message to all other nodes, and each node receives up to n distinct messages, signs each received message, and sends the corresponding signature share back to the proposer. Each proposer then aggregates the individual signature shares for its own message, verifies the aggregated signature, and multicasts it to all other nodes. Upon receiving these aggregated signatures, each node verifies up to n different aggregated signatures. This results in a communication pattern with n proposers and quadratic messaging overhead. It can be seen as n parallel invocations of the single-proposer linear messaging pattern illustrated in Figure 3.

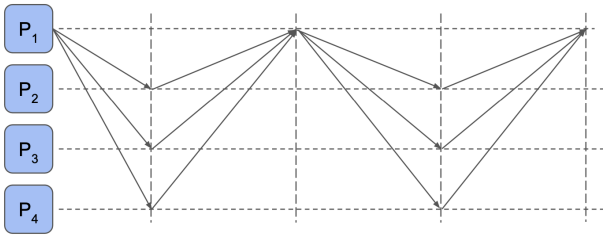

Figure 3: Single proposer linear message pattern

This structure, where each proposer sends a message to all nodes, collects signature shares, aggregates these shares, and then multicasts the aggregated signature, is commonly used in a distributed computing primitive called consistent broadcast protocol [20]. Here, the communication pattern effectively involves n parallel consistent broadcast instances, each with a different proposer. This approach is widely used in DAG-based consensus protocols, including Bullshark [19], Shoal [18], and Sailfish [17].

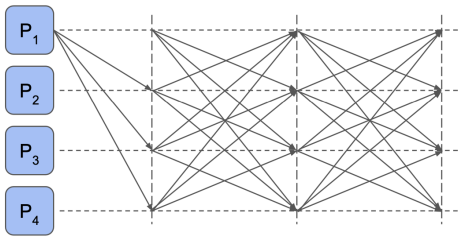

Figure 4: Single proposer quadratic message pattern

In this communication pattern, each node aggregates signatures for one message and, upon receipt, verifies n aggregated signatures. With multi-signatures, verifying each aggregated signature requires an aggregated public key, leading to $O(n)$ public key aggregations for each node. In contrast, using BLS threshold signatures simplifies this process, as each node only needs to perform a single Lagrange interpolation to generate a single threshold signature. This single interpolation is significantly less costly than the $O(n)$ public key aggregations required for multi-signatures across all system sizes. Therefore, BLS threshold signatures are more suitable for this pattern than multi-signatures.

Multi-proposers cubic messages. In this pattern, there are n distinct proposers, each proposing a unique message. Each proposer broadcasts its message to all nodes, who each receive up to n unique messages and

sign them. Nodes then multicast their signature shares to all other nodes, meaning that each node is responsible for aggregating up to n^2 individual signature shares and verifying up to n aggregated signatures. This setup results in a cubic number of messages, and nodes receive the aggregated signatures within two communication steps. It can be seen as n parallel invocations of the single-proposer quadratic messaging pattern illustrated in Figure 4.

This pattern is commonly used in DAG-based consensus protocols, including Bullshark, Shoal, and Sailfish, particularly when latency optimization is pursued using round-optimal reliable broadcast protocol [1].

This pattern requires verifying n aggregated signatures, necessitating either n Lagrange interpolations or n public key aggregations. As illustrated in Figure 2, generating a single threshold signature involves considerable computational expense; thus, creating n threshold signatures would be significantly more costly than performing n public key aggregations and aggregate signatures using multi-signatures. Therefore, BLS multi-signatures are the more efficient choice for this communication pattern.

Remarks. This work focuses on BFT consensus protocols as a primary use case, but our findings are broadly applicable to other distributed protocols that rely on signatures and quorum collection. While we specifically evaluated widely adopted BLS signatures, exploring alternative schemes, including post-quantum solutions, represents an exciting avenue for future research.

References

[1] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

[2] Dan Boneh, Sergey Gorbunov, Hoeteck Wee, and Zhenfei Zhang. Bls signature scheme. *Tech. Rep.*, 2019.

[3] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, number 1999 in 99, pages 173–186, 1999.

[4] Benjamin Y Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In *Theory of Cryptography Conference*, pages 452–479. Springer, 2023.

[5] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, pages 356–370. ACM, 2023.

[6] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5359–5376, 2023.

[7] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534. IEEE, 2022.

[8] Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing chain-based rotating leader bft via optimistic proposals. In *International Conference on Dependable Systems and Networks (DSN)*, 2024.

[9] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[10] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hints: Threshold signatures with silent setup. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 3034–3052. IEEE, 2024.

[11] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International conference on financial cryptography and data security*, pages 296–315. Springer, 2022.

[12] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[13] Monad Labs. Monadbft, 2023. Accessed on Oct 24, 2024.

[14] Zhuolun Li, Alberto Sonnino, and Philipp Jovanovic. Performance of eddsa and bls signatures in committee-based consensus. In *Proceedings of the 5th workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems*, pages 1–5, 2023.

[15] Dahlia Malkhi and Kartik Nayak. Hotstuff-2: Optimal two-phase responsive bft. *Cryptology ePrint Archive*, 2023.

[16] [n.d.]. Google Cloud Platform - general purpose machines. `https://cloud.google.com/compute/docs/general-purpose-machines`. [Online; accessed 06/03/2024].

[17] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards improving the latency of dag-based bft. In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025 (To appear).

[18] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness. In *International Conference on Financial Cryptography and Data Security*, 2024.

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

[20] TK Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.

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