arXiv:2302.00418v1 [cs.DC] 1 Feb 2023

基于委员会的共识中 EdDSA 和 BLS 签名的性能

Zhuolun Li[1],Alberto Sonnino[2,3], 和 Philipp Jovanovic[3]

[1] 利兹大学[2] Mysten Labs[3] 伦敦大学学院（UCL）

摘要。 我们通过大规模地理分布式基准测试，首次对基于委员会的共识协议中的 EdDSA 和 BLS 签名进行了性能比较。与普遍看法相反，我们发现，*small* 部署（少于 40 个验证者）可以受益于 BLS 多重签名较小的存储占用，而更大规模的部署则应优先采用 EdDSA 以提升性能。作为一项独立贡献，我们提出了一种新方法：通过利用预计算值操作聚合公钥，使基于委员会的共识协议能够验证 BLS 多重签名证书。

Key关键词： 数字签名 · 共识 · 区块链 区块链

1 引言

共识协议在区块链的核心运行，将客户端交易排序为所有诚实验证者一致认可的序列。区块链的普及激发了人们开发高性能共识系统的兴趣，早期研究提出了基于委员会的协议，以改进比特币每秒仅处理 [24]7 笔交易的低吞吐量。此后，研究表明，这些协议能够提高区块链吞吐量并降低延迟 [1,19],，而且它们正迅速成为近期权益证明架构中的标准 [10, 21, 22]。区块链领域的文献中，存在大量高效的基于委员会的共识协议。它们通过各种技术改进了现有最佳方案，涵盖高效的数据共享层 [9,27] 、稳健的基于 DAG 的协议 [12,25] ，以及能够适应故障和网络状况的多模式协议 [13,15,20,23]。

遗憾的是，这些研究很少关注签名方案的选择。我们对其代码库的检查表明，大多数研究使用 EdDSA [9,10,21,22,25] 或 BLS [12,16,17,23] ，但没有说明选择依据。这一点令人遗憾，因为数字签名需要执行 CPU 密集型操作，并且在基于委员会的共识中得到广泛应用：区块提议者通过签名来验证其区块提案，验证者通过对区块提案进行反签名来表示支持，而包含法定人数签名的证书则用于提交并最终确定交易。一方面，EdDSA 签名能够快速生成和验证；

另一方面，BLS 多重签名可生成较小的证书，并且无论委员会规模如何，证书验证时间都近乎恒定。人们普遍认为，在大规模部署中，BLS 优于 EdDSA，因为较大的 EdDSA 证书传播和验证速度较慢。然而，目前没有实证证据支持这一观点。

我们通过首次比较 EdDSA 和 BLS 签名在基于委员会的共识中的性能（据我们所知），弥补了这一空白。大规模地理分布式基准测试表明，签名方案的选择是决定系统性能的一个主要因素。研究发现，与普遍看法相反，规模相对较小的委员会（验证者少于 40 个）可以受益于 BLS 多重签名较小的存储占用，而规模较大的部署则应优先采用 EdDSA，以提升性能。简而言之，当验证者需要验证大量共同签署的区块提案时，BLS 验证的计算开销会变得难以承受，最终抵消小型、高效证书带来的优势。我们选用 HotStuff [28] 作为实验中的基于委员会的共识协议示例。作为一项独立贡献，我们提出了一种新颖的方法：通过预先计算固定数量的群元素来操作聚合公钥，从而使基于委员会的共识协议能够验证 BLS 多重签名证书。即使存在拜占庭故障，这项技术也优于传统的 BLS 验证流程；传统流程在每次验证证书时都需要重新计算相应的聚合公钥。

贡献。 总而言之，我们作出了以下贡献：

– 据我们所知，我们首次通过大规模地理分布式基准测试，对基于委员会的共识中的 EdDSA 和 BLS签名进行了性能比较。– 我们分析了每种签名方案对性能的影响，并发现小规模部署最适合利用 BLS 多重签名，而较大规模的部署则应优先采用 EdDSA。– 我们提出了一种新颖且效率更高的方法，使基于委员会的共识协议能够利用 BLS 多重签名来验证证书。

2 背景

我们回顾 BLS 多重签名，并概述典型的基于委员会的共识协议。

BLS 多重签名。Boneh-Lynn-Shacham（BLS）签名方案 [3]是一种利用配对友好椭圆曲线的高效签名方案。BLS 支持多重签名和公钥聚合，因此在各种区块链项目中非常流行。我们首先回顾标准的 BLS 签名方案。设 $\mathbb{G}_1$、 $\mathbb{G}_2$ 和 $\mathbb{G}_T$ 为素数阶 q 群，且存在一个可高效计算的非退化双线性映射 e：$\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$。

我们分别用 g_1、g_2 和 g_T 表示 $\mathbb{G}_1$、$\mathbb{G}_2$ 和 $\mathbb{G}_T$的规范生成元，并令 $\mathrm{H}_1 : \{0,1\}^* \to \mathbb{G}_1$。我们用 λ 表示安全参数，$\xleftarrow{\$}$ 表示均匀随机采样。BLS 签名方案由以下算法组成：

– BLS.Setup(1^λ)：设置并输出一个双线性群 $par = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$。– BLS.KeyGen($par$)：给定参数 par，输出一对公钥/私钥 (pk,sk)，其中 sk $\xleftarrow{\$} \mathbb{Z}_q^*$，pk $= g_2^{\mathsf{sk}} \in \mathbb{G}_2$。– BLS.Sign($par$, sk m)：给定一条消息 $m \in \{0,1\}^*$，输出签名 $\sigma = \mathrm{H}_1(m)^{\mathsf{sk}} \in \mathbb{G}_1$。– BLS.Verify($par$, pk σ, m)：给定公钥 pk $\in \mathbb{G}_2$、签名 $\sigma \in \mathbb{G}_1$ 和一条消息 $m \in \{0,1\}^*$，若 $e(\sigma, g_2) = e(\mathrm{H}_1(m), \mathsf{pk})$，则输出 1；否则输出 0。

BLS签名支持通过公钥聚合进行多重签名。多重签名方案（MSP）允许 n 个签名者针对相同的消息 m 生成简短签名 σ（签名的大小与签名者数量无关）。要验证该多重签名，需要将所有签名者的公钥聚合为一个公钥 apk, m 和 σ。

– MSP.Setup(1^λ)：输出 BLS.Setup(1^λ)。– MSP.KeyGen(par)：输出 BLS.KeyGen(par)。– MSP.Sign(par, sk m)：输出 BLS.Sign(par, sk m)。– MSP.SigAggr($\{\sigma_0, \ldots, \sigma_n\}$)：输出 $\sigma = \prod_{i=1}^n \sigma_i$。– MSP.KeyAggr($par$, {pk1, .. . , pk$_n$})：输出 $apk = \prod_{i=1}^n \mathsf{pk}_i$。– MSP.Verify($par, apk, \sigma, m$)：输出 BLS.Verify($par, apk, \sigma, m$)。

如果签名者不证明其拥有相应的私钥，MSP 方案通常容易受到恶意公钥攻击 [2]。基于委员会的区块链不存在这一问题，因为验证者会通过签署一笔特殊交易来证明其知晓自己的私钥 sk，然后才有资格加入未来的委员会。

我们注意到，门限签名是多重签名的一种推广形式，只需达到签名数量阈值 $t \le n$（而不是 $t = n$）即可计算出 σ。当不存在可由其运行设置协议的天然可信第三方时，门限签名需要复杂的分布式密钥生成机制，以生成每个签名者的密钥对 [14, 18]。此外，当验证者拥有不同的投票权时，门限签名无法自然地扩展到此类设置，因此不适用于共识协议。因此，本文不考虑门限签名，因为据我们所知，没有基于委员会的区块链在其共识协议中使用门限签名。

基于委员会的共识。 基于委员会的区块链通常将时间划分为一系列纪元（每个纪元约持续一天 [21, 22, 26]）。每个纪元都会选出一个由 $n = 3f + 1$ 个验证者组成的委员会（通常通过权益证明 [21, 22]），其中 f 是系统能够容忍的故障验证者的最大数量，

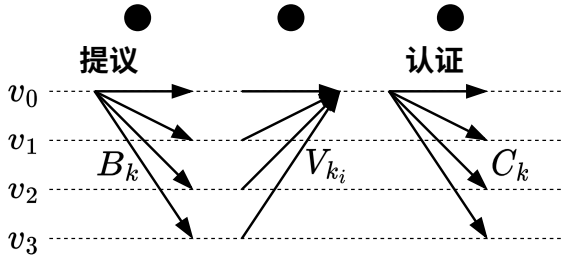

图 1. 典型基于委员会的共识协议一轮流程的高层概览。委员会由四名验证者组成（v_0、v_1、v_2、v_3）。领导者 v_0 为第 k 轮提出提案B_k；验证者针对 B_k 投票 V_k；领导者将法定人数的投票汇集成证书 C_k 并进行传播，验证者则进行验证 C_k。

系统能够容忍的。随后，选出的委员会会通过拜占庭容错（BFT）协议对客户端交易进行排序，从而“扩展”区块链。

本文仅介绍基于委员会的共识协议中最常涉及签名的部分（省略同步器等其他部分，[7]，领导者选举模块 [6]，和视图变更 [4,13]）。典型的基于委员会的共识协议按轮次运行，在每一轮中从验证者中选举一名领导者，以平衡验证者的参与。图1 高层概览了一个由四名验证者运行的典型基于委员会的共识协议中的一轮流程：v_0、v_1、v_2和v_3。第 v_0 轮的领导者 k 会传播一个区块，B_k 该区块扩展了其所知的最长区块链[4] (❶)。随后，验证者会为每一轮最多一个领导者的提案投票，除非该提案格式错误，或与其所知的更长区块链冲突，否则便对其进行共同签名（❷）；验证者将投票 V_k 发送回领导者。领导者将法定人数的 $2f+1$ 投票汇集成证书，C_k并将其分发给验证者；若证书经法定人数正确签名，验证者便予以接受 C_k （❸）。

随后，协议会重复多个轮次（通常为两轮或三轮），以提交 B_k。例如，原始的 HotStuff 协议 [28] 会在区块链中存在三个连续的已认证区块时提交 B_k，即 C_k、C_{k+1}、C_{k+2}。较新的 HotStuff 变体（如 Jolteon） [13]，只需要连续两个轮次；而最先进的基于 DAG 的协议 [9,12,25] 则允许多个验证者并行传播提案。不过，协议的总体流程仍然相似。如果领导者失效或长时间无响应，验证者会运行 *view-change* 子协议以选举新的领导者 [4]；更换领导者的代价高昂，并会严重降低性能。

在基于委员会的共识中，采用 BLS 多重签名的主要动机是缩小证书的大小（证书大小会随着委员会规模线性增长）

[4] 通常，领导者会收集交易批次并提出，这些交易批次被称为区块，因此该协议会形成区块链（或称“区块链”）。

并实现近乎恒定时间的验证（只需验证一个多重签名，而非逐一验证 $2f+1$ 票投票）。

3 缓存的BLS多重签名验证

我们通过新增函数2 来扩展第 **MSP.KeyDisAggr** 节中介绍的 BLS 多重签名方案，该函数可从聚合公钥中减去一组公钥 apk。

– **MSP.KeyDisAggr**(par，apk，{–pk$_1$，…，–pk$_n$}）：给定聚合公钥 apk，以及所有公钥 $PK = \{–\text{pk}_1, \ldots, –\text{pk}_n\}$ 的相反数，输出$apk^* = apk \prod_{i=1}^{j} –\text{pk}_i$。

此函数每移除一个公钥，都需要进行一次椭圆曲线加法。下面说明如何将其纳入第 1 节图 2所示的基于委员会的共识协议常规流程中。

协议描述。委员会中的每位验证者都会使用**MSP.Setup**(1^λ)生成的公共参数 par 进行初始化。每位验证者在本地运行**MSP.KeyGen**(par)，生成自己的公钥/私钥对（pk，sk），并公布 pk（见第2节）。每位验证者都会存储所有其他验证者的公钥 {pk$_1$，…，pk$_n$}。他们还会计算并存储聚合公钥apk = **MSP.KeyAggr**(par，{pk$_1$，…，pk$_n$})，以及所有验证者公钥 {–pk$_1$，…，–pk$_n$} 的相反数。

– **步骤 ❶：提案**。第 k 轮的领导者收集一组客户端交易 l，并创建一个区块提案 $m = (k, l, \cdot)$，其中点号“·”表示省略了协议特定字段。随后，领导者使用其私钥 sk 调用 σ_B =**MSP.Sign**(par，sk m) 对 m 签名，并将 B_k (m, σ_B) 传播给其他验证者。

– **步骤 ❷：投票**。验证者首先解析 $B_k = (m, \sigma_B)$，然后通过**MSP.Verify**(par，pk σ_B，m) 对其进行验证，其中 pk 是领导者的公钥。如果检查通过且满足所有其他协议特定条件，验证者便会对 B_k 进行共同签名，并向领导者发送其投票 V_k = **MSP.Sign**(par，sk $H(B_k)$)（其中 H 是一种抗碰撞哈希函数）。

– **步骤 ❸：认证**。领导者调用 **MSP.Verify**(par,pk $V_k, H(B_k)$)，其中 pk 是投票者的公钥，用于验证每一张收到的投票。一旦收到 $2f+1$ 有效投票 $\{V_{k_1}, \ldots, V_{k_n}\}$，便调用$\sigma_C$ = **MSP.SigAggr**($\{V_{k_1}, \ldots, V_{k_n}\}$) 对其进行聚合。随后，它会计算一个位图 b，指出哪些验证者没有为 σ_C 作出贡献。具体做法是确定性地为每个验证者分配一个索引，使其对应于该验证者在位图中的位置。该位图可缩小证书的大小，否则证书将包含每个签名者的公钥。随后，领导者将证书 $C_k = (\sigma_C, b)$ 分发给验证者。收到C_k后，验证者使用位图 b 识别未为 σ_C 作出贡献的验证者，取回此前缓存的 {–pk$_1$，……，–pk$_j$}，并

计算 $apk^* = \textbf{MSP.KeyDisAggr}(par, apk, \{-\text{pk}_1, \ldots\ldots, -\text{pk}_n\})$。然后，他们调用 $\textbf{MSP.Verify}(par, apk^*, \sigma_C, H(B_k))$ 验证证书。

这种方法有两个主要优点。(i) 与在证书中包含每位签名者公钥的直接实现相比（许多生产系统都采用这种做法 [10, 21, 22]），位图可使每位签名者的证书大小减少 32B；证书会永久存储在区块链上，因此随着时间推移，任何消息压缩都能带来显著收益。(ii) 在验证证书之前，验证者计算 apk^* 所需的椭圆曲线加法最多为 f 次，而直接进行 BLS 多重签名验证则每次都需要重新计算聚合验证密钥$2f + 1$ 。此外，实际应用中的领导者实现通常会在收集到第一批 $2f + 1$ 投票后等待约 50–100 毫秒，让其余验证者有额外时间进行投票。因此，在常见情况（正常路径）下，典型证书包含接近 $n = 3f + 1$ 份投票，而计算 apk^* 只需进行一到两次椭圆曲线加法。

4 性能比较

我们选择 HotStuff [28] 作为实验中基于委员会的共识协议示例。我们选择该协议，是因为它是生产环境区块链中使用最广泛的基于法定人数的共识协议；Celo [5], 、Cypherium [8],、Flow [11], 、Diem [10], 和 Aptos [21] 都运行 HotStuff 的变体。此外，它与 Tendermint [26] （其最接近的先驱）有许多共同的设计特征。具体而言，我们在一种名为 Jolteon [13]的双链 HotStuff 变体上运行基准测试；选择这一变体，是因为 Diem [10], 、Aptos [21], 和 Flow [11] 都在生产环境中运行它。

实现。 我们在 Jolteon 的原始开源实现之上实现了 BLS 多重签名[5]。该实现使用 Rust 编写，采用 Tokio[6]进行异步网络通信，并使用 ed25519-dalek[7] 进行签名，数据结构则使用 RocksDB 持久化[8]。该实现使用 TCP 来建立可靠的点对点信道，这是正确实现分布式系统抽象所必需的。由于该实现使用 EdDSA（基于 Ed25519 曲线），我们修改其 加密模块，使其采用第 3 节所述的 BLS 多重签名。我们使用 Filecoin 的 BLS 实现（基于 BLS12-381 曲线）[9]。我们将开源支持 BLS 的实现[10]。

[5] https://github.com/asonnino/hotstuff[6] https://tokio.rs[7] https://github.com/dalek-cryptography/ed25519-dalek[8] https://rocksdb.org[9] https://github.com/filecoin-project/bls-signatures[10] https://github.com/radiken/hotstuff-digital-signature-benchmarking

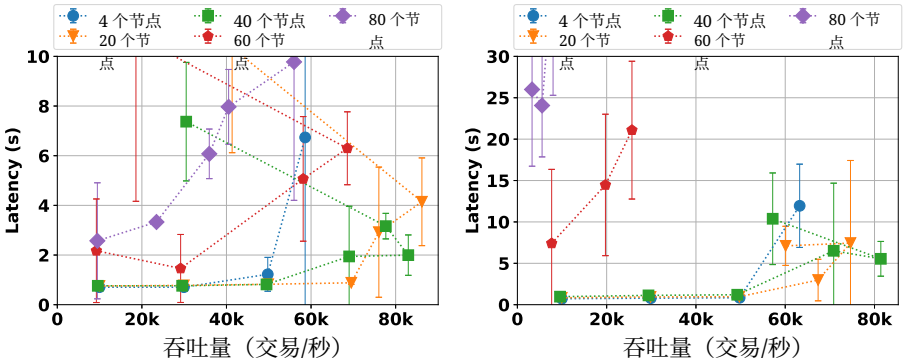

图 24、20、40、60、80 。基于 EdDSA 的实现；以及验证者（广域网）。**图 3**4、20、40、60、80 。基于 BLS 的实现；以及验证者（广域网）。

评估设置。 我们通过在 Amazon Web Services (AWS) 上进行实验，评估采用 BLS 多重签名的 HotStuff/Jolteon 的吞吐量和延迟。随后，我们将其性能与使用 EdDSA 的基线实现进行比较，并考察不同委员会规模下的表现。

我们在 AWS 上部署了一个测试平台，使用 t3.medium 实例，分布于 4 个不同的 AWS 区域：北弗吉尼亚（us-east-1）、北加利福尼亚（us-west-1）、悉尼（ap-southeast-2）和法兰克福（eu-central-1）。验证者尽可能均匀地分布在这些区域中。每台机器提供最高 5 Gbps 的带宽、2 个虚拟 CPU（1 个物理核心），配备 2.5 GHz 英特尔至强 Platinum 8175 处理器和 4 GB 内存，运行 Linux Ubuntu 服务器 20.04。在以下各节中，图表中的每项测量结果均为 5 次独立运行的平均值，误差线表示一个标准差。我们的基准实验参数为：区块大小为 500KB，交易大小为 512B，每个参与方配备一个基准测试客户端，以固定速率提交交易，持续 5 分钟。领导者超时值设为 5 秒。这里所说的延迟，是指客户端提交交易到某个验证者提交该交易之间经过的时间。我们通过跟踪系统中的样本交易来测量这一时间。

分析。 图 2和图3 表明，无论采用哪种签名方案，部署 4 个验证者时，每秒可处理 50,000 笔交易，同时将延迟控制在 1.5 秒以内。同样，部署 20 个或 40 个验证者时，无论采用哪种签名方案，每秒可处理约 80,000 笔交易，同时将延迟控制在约 4 至 5 秒[11]。

将委员会规模增加到 60 名验证者后，我们基于 EdDSA 的实现性能降至约 60,000tx/s，延迟则增加到

[11] 20 个和 40 个验证者的吞吐量高于 4 个验证者，这看起来可能令人惊讶。不过，这是一个已知的结果： [9, 13], 额外验证者所提供的额外容量有助于提高资源利用率。

5 秒（图 2）。性能下降的原因既包括向大量验证者广播消息所需的额外带宽，也包括验证大量签名所需的 CPU 开销。事实上，领导者每轮（图 $2f+1=41$ 的步骤 ❷）至少需要验证 1 张投票，而验证者需要验证 41 个签名才能验证每份证书[12]（图 ❸ 的步骤 1）。基于 BLS 的实现性能下降更为显著：图3 显示，系统最多只能处理 20,000 笔交易，延迟约为 15 秒。这种性能差异似乎是由于领导者验证单个投票所需的时间造成的（图 ❷ 的步骤 1）。EdDSA 能让领导者高效地验证投票，而 BLS 验证的速度约慢 100 倍，并会独占领导者的 CPU。这会拖慢领导者，进而影响吞吐量和延迟。

图 2 表明，即使将部署规模进一步扩大到 80 个验证者，基于 EdDSA 的实现性能也会下降至约 40,000 交易/秒（延迟为 6 秒）。图 3 显示，我们基于 BLS 的实现几乎无法处理交易：每秒只能处理几千笔交易，延迟超过 25 秒。领导者验证投票所需的时间大多数情况下超过 5 秒（即领导者超时值），此时验证者会认为领导者已崩溃，并启动视图切换子协议以选举新的领导者；这种情况频繁发生，严重降低了性能。

主要结论。我们的实验表明，BLS签名并未带来明显的性能优势。与普遍看法相反，大规模部署无法从 BLS 的聚合特性中受益。尽管 BLS 多重签名能够生成较小的证书，并且证书验证时间几乎恒定（不受委员会规模影响），但验证单个投票所产生的 CPU 开销大幅抵消了这一优势。因此，大规模部署应优先选择 EdDSA。小规模部署（最多 40 个验证者）不会给领导者带来过重的 CPU 负担，采用 EdDSA 和基于 BLS 的部署性能相近，因此可以利用 BLS 的聚合特性。较小的 BLS 多重签名证书还能带来显著的存储优势：存储 EdDSA 证书需要 2.5 KB（对于包含 40 个节点的部署），而 BLS 多重签名证书仅需约 100 B。随着时间推移，这一区别可能变得十分显著，因为证书是永久保存在区块链中的一部分。

致谢

本研究部分由 Mysten Labs 资助。

参考文献

1. Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., Danezis, G.：区块链时代的共识。arXiv 预印本 arXiv:1711.03936（2017）[12] EdDSA 的“批量验证”功能大幅加快了证书验证速度。

2. Boneh, D., Drijvers, M., Neven, G.：面向更小区块链的紧凑型多重签名。载于：密码学进展——ASIACRYPT 2018：第24届密码学理论与应用国际会议，布里斯班，昆士兰州，澳大利亚，2018年12月2日至6日，会议论文集，第二部分。第435–464页。施普林格（2018） 3. Boneh, D., Lynn, B., Shacham, H.：基于 Weil 配对的短签名。载于：密码学理论与应用国际会议。第514–532页。施普林格（2001） 4. Castro, M., Liskov, B., 等：实用拜占庭容错。载于：OSDI。第99卷，第173–186页（1999） 5. Celo：携手共建，共享繁荣。https://celo.org（2023） 6. Cohen, S., Gelashvili, R., Kogias, L.K., Li, Z., Malkhi, D., Sonnino, A., Spiegelman, A.：关注你的领导者。载于：金融密码学与数据安全：第26届国际会议，FC 2022，格林纳达，2022年5月2日至6日，修订精选论文。第279–295页。施普林格（2022） 7. Cohen, S., Goren, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.：模块化区块链架构中的可用性与检索证明。密码学电子印本档案（2022） 8. Cypherium：面向 Web3 的区块链。https://www.cypherium.io（2023） 9. Danezis, G., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A.：Narwhal 和 Tusk：基于有向无环图的内存池与高效拜占庭容错共识。载于：第十七届欧洲计算机系统会议论文集。第34–50页（2022） 10. Diem：欢迎来到 Diem 项目。https://www.diem.com（2023） 11. Flow：构建强大、安全且可扩展的 Web3 应用。https://flow.com（2023） 12. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.：Dumbo-NG：吞吐量无关延迟的快速异步拜占庭容错共识。载于：2022年 ACM SIGSAC计算机与通信安全会议论文集。第1187–1201页（2022） 13. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.：Jolteon 和 Ditto：具有异步回退机制的网络自适应高效共识。arXiv 预印本 arXiv:2106.10362（2021） 14. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.：基于离散对数的密码系统的安全分布式密钥生成。载于：密码学理论与应用国际会议。第295–310页。施普林格（1999） 15. Guerraoui, R., Kneˇzevi´c, N., Qu´ema, V., Vukoli´c, M.：接下来的700种拜占庭容错协议。载于：第五届欧洲计算机系统会议论文集。第363–376页（2010） 16. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.：Speeding Dumbo：推动异步拜占庭容错走向实用。密码学电子印本档案（2022） 17. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.：Dumbo：更快的异步拜占庭容错协议。载于：2020年 ACM SIGSAC计算机与通信安全会议论文集。第803–818页（2020） 18. Kate, A., Huang, Y., Goldberg, I.：现实环境中的分布式密钥生成。密码学电子印本档案（2012） 19. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.：通过集体签名增强比特币的安全性和性能，并实现强一致性。CoRR **abs/1602**06997 20**06997** 20161602.(), http://arxiv.org/abs/ 。. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.：Zyzzyva：推测式拜占庭容错。载于：第二十一届 ACM SIGOPS操作系统原理研讨会论文集。第45–58页（2007）

21. Labs, A.：致力于在 Aptos 区块链上开发产品和应用，重新定义 Web3 用户体验。https://aptoslabs.com（2023）22. Labs, M.：无界构建。https://sui.io（2023）23. Lu, Y., Lu, Z., Tang, Q.：Bolt-Dumbo Transformer：与流水线拜占庭容错一样快速的异步共识。载于：2022 年 ACM SIGSAC计算机与通信安全会议论文集。第 2159–2173 页（2022）24. Nakamoto, S.：比特币：一种点对点电子现金系统。《去中心化商业评论》，第 21260 页（2008）25. Spiegelman, A., Giridharan, N., Sonnino, A., Kokoris-Kogias, L.：Bullshark：让 DAG 拜占庭容错协议走向实用。载于：2022 年 ACM SIGSAC计算机与通信安全会议论文集。第 2705–2718 页（2022）26. Tendermint：为分布式网络打造最强大的工具。https://tendermint.com（2023）27. Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.：Dispersedledger：在可变带宽网络上实现高吞吐量的拜占庭共识。arXiv 预印本 arXiv:2110.04371（2021）28. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.：HotStuff：具备线性性和响应性的拜占庭容错共识。载于：2019 年 ACM分布式计算原理研讨会论文集。第 347–356 页（2019）