Decentralized Proposer-as-a-Service (DPaaS): Improving Decentralization and Trustworthiness in Ethereum PBS

Chenyang Liu
Duke University
chenyang.liu@duke.edu

Arnav Jindal
Duke University
arnav.jindal7@gmail.com

Ittai Abraham
a16z crypto
ittaia@gmail.com

Matthew Lentz
Duke University
mlentz@cs.duke.edu

Kartik Nayak
Duke University
kartik@cs.duke.edu

Abstract

Proposer-Builder Separation (PBS) in Ethereum aims to improve decentralization and scalability by offloading block construction to specialized builders, but its current MEV-Boost implementation relies on trusted relays, introducing centralization as well as security and performance concerns. We propose Decentralized Proposer-as-a-Service (DPaaS) to eliminate relays while preserving compatibility with Ethereum's consensus layer by distributing the combined proposer and relay roles across Proposer Entities (PEs) running in independent Trusted Execution Environments (TEEs). DPaaS presents itself to Ethereum as a single validator using threshold BLS signatures, while a new Byzantine broadcast protocol decentralizes the auctioneer role with censorship-resistance and availability-backed guarantees. Our evaluation, deployed across four independent cloud hosts and driven by real-world traces, shows that DPaaS achieves $\leq$ 6 ms bid processing latency, 68.7 ms latency from the end of auction to block proposal, and 3.3% more MEV earnings – demonstrating that DPaaS can offer security and decentralization benefits while providing strong performance.

1 Introduction

Maximal Extractable Value (MEV) [19, 28] refers to the maximum value that can be extracted from block production by changing the order of transactions in a block. Despite adding economic incentives to DeFi systems like Ethereum, MEV contributes greatly to centralization because it disproportionately favors participants with richer resources [8, 11, 27, 93]. This has led to the *Proposer-Builder Separation (PBS)* [27] architecture proposed by the Ethereum community: every slot (12 seconds), a randomly selected *validator* acts as the *proposer* to select a block for the slot by soliciting bids from *builders* who try to construct high-value blocks. PBS's vision is to have a more open and competitive builder market while enabling all validators to capture MEV.

Unfortunately, it is hard to realize PBS in practice due to three key constraints. First, the solution must ensure fair exchange [23]: proposers should receive *available* and *valid* blocks with payment from builders, while builders demand their block payloads remain *private* unless proposed for the slot. Given the mutual distrust between proposers and builders, there needs to be a trust anchor; however, any inherent trust in the system should be reduced as much as possible. Second, the solution must permit a low-latency, high-throughput implementation, given that MEV is very sensitive to latency [78, 92] and most valuable bids arrive right at the end of slot [78]. This creates tension with the first constraint, since cryptographic trust anchors (e.g., Zero-Knowledge Proofs [33, 49]) can't meet the sub-second budget. Third, the solution should be readily and incrementally deployable. Changes to the core Ethereum protocol incur substantial specification and coordination complexity [32], historically taking years to reach mainnet [12, 54, 70].

Today, PBS is implemented via MEV-Boost [40], a sidecar protocol that introduces *relays* to serve as the trust anchor, taking on the role of both auctioneer and escrow. Unfortunately, relays are *unconditionally trusted* by both parties, with no mechanisms to constrain the behavior: a compromised relay can expose non-winning payloads and steal MEV [38], deny block availability to a proposer, or bias the auction by censoring a builder. Relays are also *structurally centralized*, concentrating trust, confidentiality, and liveness in an entity; additionally, with 90% of Ethereum blocks routed through 5 relays [91], leading to single points of failure on the critical path. Moreover, relays incur performance overheads due to their *commit-and-reveal* scheme between the proposer and relay, leading to less MEV captured from an earlier end of auction [78]. Despite the above, MEV-Boost operating as a sidecar (i.e., avoiding any core protocol changes), enabled its wide-spread adoption and deployment.

Several efforts [20, 41, 66] have sought to realize PBS without trusted relays, but none satisfies all three constraints simultaneously. TEE-Boost [41] leverages TEE attestations to facilitate part of the fair exchange (i.e., proving the block is valid); however, it fails to address block availability. A proposal by Noyes and Policharla [66] extends TEE-Boost to

address the availability issue by employing Silent Threshold Encryption [45]; however, this involves changes to the core protocol for validators and conflicts with its dynamic availability goals. Enshrined PBS (ePBS) [20] presents an alternative design that requires staking builders to avoid misbehavior (e.g., submitting an invalid block); however, this involves significant changes to the core protocol and does not prevent the use of sidecar protocols, which would impose lower entry barriers on builders and often yield higher revenue [90]. Finally, none of the above address potential biases on builders in the auction.

Our insight is that these constraints can be reconciled by decentralizing the combined proposer and relay roles into a Decentralized Proposer-as-a-Service (DPaaS): an architecture built on a distributed set of TEEs. Users enroll as validators through a given deployment that meets their security policy (e.g., different cloud providers and/or TEEs). DPaaS handles all of the responsibilities for a proposer, including sourcing blocks from builders, running the auction, and proposing the block. To avoid centralized trust, DPaaS ensures that individual (or a small fraction of) compromised TEEs do not lead to the same loss of security as with relays. By combining the proposer and relay roles, DPaaS reduces the latency between the end-of-auction and proposal and allows the user to capture more MEV from later (higher) bids.

To realize DPaaS, we needed to address several challenges. First, DPaaS must appear as a single validator to the network despite being distributed across TEEs, and must enable builders and enrolled users to justifiably regard it as trustworthy. Second, DPaaS must ensure all security goals are met, despite the presence of a small fraction of faulty TEEs [15] and a partially synchronous network [22]. Third, DPaaS must match performance of centralized relays, despite acting as a distributed auctioneer, for practical deployment. In this paper, we solve the challenges and make the following contributions:

- We propose Decentralized Proposer-as-a-Service (DPaaS), a readily deployable PBS solution as a sidecar protocol for Ethereum that improves trustworthiness compared to the current relay-based solution while remaining compatible with both the core Ethereum protocols and the broader builder ecosystem.
- We formalize the Censorship-resistant and availability-backed Byzantine Broadcast (Crab-BB) problem, develop a protocol primitive, and build our decentralized auction protocol on top to achieve all of the security goals.
- We implement a prototype of DPaaS, which we validated end-to-end in a local Ethereum testnet. We evaluated it using a deployment across four TEEs with diverse cloud providers, availability zones, and TEE implementations. Our evaluation, driven by real-world bid traces, demonstrates DPaaS (on average) processes bids in less than 6 ms and can propose blocks within 68.7 ms once the auction completes (leading to 3.3% more MEV).

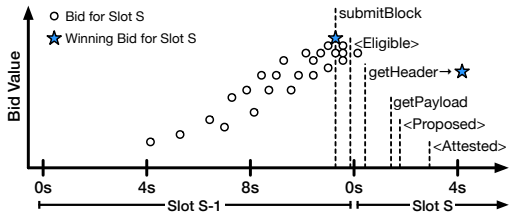

Figure 1: Example slot timeline with MEV-Boost.

2 Background, Goals, and Motivation

2.1 Ethereum and MEV-Boost

Ethereum currently operates under a Proof-of-Stake (PoS) consensus protocol known as Gasper [13]. Participants become *validators* by staking 32 ETH [26]. Ethereum divides time into 12-second *slots*; in each slot, a designated validator (i.e., the *proposer*) selects a block while a committee of validators (*attesters*) vote on the block. Originally, the proposer was responsible for sourcing the transactions and constructing the proposed block themselves. Proposer Builder Separation (PBS) [27] decomposes this work, such that *builders* construct profitable blocks and offer them to the proposer (with a bid).

Today, MEV-Boost [40] implements PBS via a sidecar protocol. MEV-Boost uses trusted third parties known as *relays* to support fair exchange between builders and proposers. Relays act as block aggregators, auctioneers, and escrows through several key APIs: `submitBlock` for builders, `getHeader`, and `getPayload` for proposers. Fig. 1 shows an example timeline of a slot, focusing on the winning bid (star). Builders start constructing blocks for slot S during slot $S-1$ and submit their blocks and bids via the `submitBlock` API. After receiving a bid, the relay validates it by simulating the block payload transactions; if successful, the relay includes the bid in the auction (*Eligible*). Eventually, the proposer calls `getHeader` to obtain the block header for the winner of the auction from the relay. The proposer signs the block header and submits it with the call to `getPayload` to obtain the block payload. The relay verifies the signature from the proposer and propagates the Signed Beacon Block (signed block header and beacon block body) [29]; after a delay, typically 1 second, the relay also releases the payload to the proposer so it can propagate the Signed Beacon Block as well. Finally, attesters vote on the block by the 4-second deadline [57].

2.2 Goals for PBS

The PBS proposal [27] identifies several goals, both explicitly and implicitly, that any solution should meet to be deployed and adopted. We specify these as follows:

G1 Bid Selection The winning block must have the *highest* bid among *all* validated bids.

G2 Block Validity The winning block payload must include the builder's bid as payment to the proposer. Additionally, all honest validators must consider the block valid.

G3 Block Availability The proposer must be able to propose the winning block to the validators.

G4 Failed Trade Privacy The proposer must not learn or reveal the (plaintext) payloads for any non-winning blocks.

For **G1**, we note that this implies that block proposal should not be biased by the proposer or third party. Additionally, MEV-Boost constrains this to select the highest bid from the bids that are both eligible (i.e., validated by the relay) and not canceled by the builder [42, 62].

Beyond these core goals, we set forth three additional goals:

G5 Trustworthy Any solution should be trustworthy, rather than relying *purely on inherent* trust in the components.

G6 Compatibility Any solution must not modify the Ethereum consensus protocol and preserve the builder market (e.g., bid cancellation). This is important for deployment, since changes to the core protocol often take several years (or more) and require a hard fork.

G7 Performance Any solution should provide reasonable performance as compared to relays, both from the perspective of builders (e.g., time for a bid to become eligible) and proposers (e.g., time to finalize the auction).

2.3 Other PBS Proposals

MEV-Boost meets the goals of PBS (**G1**-**G4**), but does so by relying entirely on a trusted relay that facilitates fair exchange between builders and proposers. A compromised or malicious relay can violate *all* of the goals. This is not only observed in theory but in practice: there have been several attacks against relays, leading to the loss of funds [38]. This is why we argue for the importance of **G5**, and it has led the community to consider other alternative solutions to supporting PBS.

Via Proofs To achieve **G2** and **G4**, a builder can prove block validity without revealing its payload to the proposer. Zero-Knowledge Proofs (ZKPs) are meant for this, and are already pursued by Ethereum L1&L2 [33, 49]. Despite advances, proof generation remains too slow: state-of-the-art SP1 Hypercube generates proofs for 95.4% of Ethereum blocks within 10 s and 99.7% within 12 s using 16 NVIDIA RTX 5090 GPUs [83]. While seemingly sufficient for Ethereum's 12 s slot timescale, this does not take into account: 1) time for prior block to propagate (2-4 s into the slot), 2) time for builder to construct a profitable block, 3) time to verify the proof.

One alternative to ZKP is to leverage Trusted Execution Environments (TEEs), which can provide proofs in the form of *attestations* but incur inherent trust in the hardware (i.e., CPU) and the manufacturer (e.g., secret provisioning). Attestations provide cryptographic evidence, based on the manufacturer-provisioned hardware root-of-trust, that convey to a verifier that software with a specific identity (measurement) ran in the TEE on valid trusted hardware. Major cloud providers have adopted TEE implementations, including Intel SGX [18], Intel TDX [53], and AMD SEV [55].

TEE-Boost [41] proposed to use TEEs for builders to prove the validity of their block to a proposer without revealing its contents. TEE-Boost does not address **G3**, since a builder could win the auction but choose not to send the block to the proposer. A follow-up proposal by Paradigm [66] extended TEE-Boost through the use of Silent Threshold Encryption [45]. The builder TEE encrypts the block for the validator committee assigned to the slot; if the proposer chooses the block, it sends the encrypted block to the committee, which can decrypt it once a threshold attests to the block. While this addresses **G3**, it does not address **G1** as builders' bid can be censored by the proposer; additionally, it poses issues with supporting Ethereum's dynamic availability goals [94] due to the threshold required for decryption, and modifies the core Ethereum protocol (**G6**).

Via Staking Instead of proving that a block is valid, an alternative is to stake builders to punish misbehavior (e.g., invalid block) via slashing. This is the core idea behind the Enshrined PBS (ePBS) proposal [90], which is currently under development. However, ePBS requires builders to adapt to the in-protocol mechanism, deposit stake, and lose out on bidding strategies due to a single bid auction [85]. Moreover, ePBS delegates the auction to the proposer: while an honest proposer will select the block with the highest bid, this is not enforced (and thus does not meet **G1** for builders).

Reflection TEEs are promising in avoiding issues with compatibility and performance. However, naive use of TEEs is problematic for two reasons: 1) there are a number of vulnerabilities discovered over time [14,47,77,89], and 2) they do not address availability concerns due to offloading to untrusted software (e.g., hypervisor). Therefore, we need to consider a *security-in-depth* approach that limits the effects as a result of any single point of failure. Additionally, we need to support unbiased auctions (per **G1**), enabling builders to know that their work constructing a block will pay off if they are the highest bidder, while enabling proposers to receive the best available block with payment. These observations motivate our design for DPaaS, which we discuss next.

3 Overview

In this section, we provide an overview of the DPaaS system architecture (Sec. 3.1), discuss our threat model (Sec. 3.2), and end with our design roadmap (Sec. 3.3).

3.1 Architecture

Fig. 2 shows the architecture of DPaaS. DPaaS realizes the Proposer-Builder Separation (PBS) abstraction in Ethereum in a decentralized manner by executing the combined roles of the

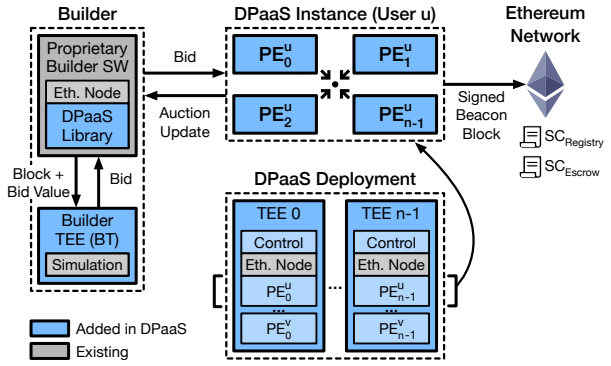

Figure 2: Overview of DPaaS architecture.

proposer and relay across a set of distributed TEEs. A DPaaS *deployment* consists of a fixed set of n TEEs spread across one or more infrastructure providers; we call any principal setting up and running a DPaaS deployment an *operator*. Each DPaaS deployment hosts an arbitrary number of DPaaS *instances*, each of which is made up of n Proposer Entities (PEs) across the TEEs. Each TEE contains management software (Control) that enables the operator to spawn a new DPaaS instance when a user wants to enroll as a validator (proposer) in Ethereum through DPaaS. In addition, each TEE operates an Ethereum node to interface with the network for playing validator's role. The PEs in a given instance (e.g., PE_i^u for user u) jointly execute the role of a validator on behalf of the user; our focus, in this paper, is when the validator becomes a proposer for a given slot. Therefore, PEs are responsible for running an auction over bids from builders, determining the winning block, and proposing a Signed Beacon Block to the attester committee by the slot deadline.

On the builder side, most of the software remains unchanged (e.g., proprietary algorithm for ordering transactions); however, the software must be modified to use the DPaaS Library for handling block submission and auction updates. After constructing a block, the builder software uses the DPaaS Library to submit the block and a desired bid value to a DPaaS Builder TEE (BT). The BT first performs block validation (e.g., contains payment to proposer, no invalid transactions) by leveraging an existing block simulation library. If successful, the BT creates a bid request containing a sealed payload to the PEs, which serves to: 1) protect the confidentiality of the block payload while allowing later reconstruction via secret sharing, and 2) provide a proof to the PEs that the bid is valid via TEE attestation. The DPaaS Library handles auction updates from individual PEs (e.g., "new highest bid seen by a PE") by aggregating and coalescing them into DPaaS instance-level updates (e.g., "new highest bid that all PEs can agree on").

DPaaS introduces two new smart contracts: registry (SC_R) and escrow (SC_E). SC_R stores metadata for all registered PEs, allowing builders to identify slots with DPaaS proposers and determine the PEs (e.g., endpoint information). SC_E manages user-to-operator payments when blocks are successfully proposed through the operator's deployment, thus providing monetary incentive for operators to maintain availability.

Motivation for Operators/Deployments To reduce design complexity, we could instead let each enrolled user set up and manage n TEEs to host their own DPaaS instance; however, this does not make sense economically, since the PEs must be long running to handle validator duties and the cost for a CVM is relatively high compared to expected earnings. Therefore, we focus on supporting sharing of infrastructure (managed by an operator) to drive down these costs, which we analyze later in Sec. 9. We note that this sharing is similar to Distributed Validator Technology (DVT) [30], which is deployed at scale on Ethereum [59, 81], which distributes a validator's key and duties across a set of nodes. In contrast to DVT, DPaaS avoids leveraging (trusted) relays for proposer duties and is more trustworthy through PEs running in attested TEEs rather than relying on (assumed) diversity of infrastructure.

3.2 Threat Model

DPaaS involves three classes of mutually distrusting principals: builders, which construct and submit blocks; operators, which provision and manage DPaaS deployments; and users, which enroll as validators/proposers. Because DPaaS spans multiple independently administered parties and infrastructure domains, we consider compromise at each trust boundary rather than assuming a single uniform adversary. The adversary seeks to violate any of the security goals (**G1–G5**). We assume partially synchronous communication [22]: there exists an unknown Global Synchronization Time (GST), after which network delays are bounded by a known Δ. We further assume loosely synchronized clocks (on the order of milliseconds), as is typical for cloud hosts [6].

Builder. Each builder hosts its own proprietary software (e.g., block construction algorithm) along with the DPaaS Builder TEE (BT) on a cloud provider it trusts. From the perspective of users, any software outside of the BT is completely untrusted and may be compromised by the adversary, while the TEE platform and software inside of the BT are trusted for integrity. For instance, the adversary may attempt to equivocate when sending bids to PEs in the user's DPaaS instance, but they are not able to forge platform attestations or induce unintended changes to the BT code and data at runtime. We consider DoS attacks via Sybil builder bid submission as out of scope; relays are vulnerable to similar attacks, and we could adopt similar reputation mechanisms to address them.

Operator. Operators coordinate the infrastructure hosting the TEEs and can therefore terminate or delay them, affecting availability. However, they cannot violate the integrity of uncompromised PEs. Such availability attacks are also

economically discouraged because operators receive success-contingent payments from enrolled users only when blocks are successfully proposed.

PEs. The *static* adversary may compromise up to f of the $n \geq 5f-1$ PEs' confidentiality, integrity, and availability. Compared with $n \geq 3f+1$, $n \geq 5f-1$ can achieve better theoretical lower bound latency [2, 52, 56] which is essential in PBS. We will discuss more on the trade-off in Appendix E. This assumption is justified by deploying PEs across multiple cloud providers (and/or availability zones), and by using heterogeneous TEE implementations (*e.g.*, Intel SGX/TDX, AMD SEV-SNP, and Arm CCA). We assume major cloud providers will not collude, while availability zones within a provider support fault isolation with independent power and networking infrastructure (reducing the risk of correlated outages). Heterogeneous TEEs further reduce correlated compromise because they rely on different processor vendors, firmware stacks, and isolation mechanisms. Prior attacks [14, 47, 76, 77, 89] showed that many TEE attacks exploit implementation-specific weaknesses in hardware or software. We consider physical attacks [16, 21, 58, 79] against TEEs in scope as one possible approach for PE compromise; however, such attacks are difficult to launch, especially across cloud providers.

This threat model represents a significant improvement over those assumed by relays: a single-domain compromise or arbitrary off-chain operator behavior can silently violate relay security guarantees, whereas DPaaS remains secure as long as at most f PEs are faulty. In DPaaS, operator misbehavior is largely confined to availability degradation and is economically disincentivized, since operators are rewarded only for successful block proposals. We discuss how enrolling users (Sec. 4) and builders (Sec. 5.1) can determine whether a given operator's configuration meets their trust assumptions.

Note on Asymmetry We assume the adversary may violate the integrity of some PEs, up to a bound (f out of n), but not BTs. We want to capture the strongest threat model for which our DPaaS design remains correct. Compromising BTs can only result in missed slots, which are easily detected and mitigated (e.g., blocklist); however, proposer compromise is far more serious (e.g., no privacy or fair exchange guarantees). Recent PBS proposals [39, 41, 66] also introduce TEEs at builders and highlight the fact that bugs in relays are more common than TEE integrity vulnerabilities (in addition to the scope of BT compromise noted above).

3.3 Design Roadmap

The DPaaS design consists of two phases. In the *offline phase*, the PEs establish an attested validator identity that allows users and builders to verify the deployment (Sec. 4). In the *online phase*, the PEs jointly play the combined role of relay and proposer including collecting bids for the auction (Sec. 5) and determining the winner and recovering winning block (Sec. 6). We defer our security analysis to Appendix B and C.

Algorithm 1 User Enrollment

D_i = Deployment on TEE_i, U = User, O = Operator

1: U $\rightarrow$ O: Sends enrollment request
2: O $\rightarrow$ D_i: Sends instance request with base manifest M^b
   $M^b = \{m_j^b\}$ and m_j^b = [Endpoint$_j$, Placement$_j$]
3: D_i: Starts a new PE_i process with M^b as input
4: PE_i: Generate $(sk_{\mathrm{V}}^i, pk_{\mathrm{V}}^i)$ and $(pk_{\mathrm{asym}}^i, sk_{\mathrm{asym}}^i)$ key pairs
5: $\mathsf{PE}_i \rightarrow \mathsf{PE}_j$: Connect to all other PE_j via RA-TLS $[m_i^b, \mathrm{meas}_i]$
6: $\mathsf{PE}_i \rightarrow \mathsf{PE}_j$: Send pk_{V}^i and pk_{asym}^i to all other PEs
7: $\mathsf{PE}_i \leftrightarrow \mathsf{PE}_j$: Complete DKG for $(sk_{\mathrm{p_sign}}^i, pk_{\mathrm{p_sign}}^i)$ key pair
8: PE_i: Generate the local manifest m_i and $\mathrm{Sign}(sk_{\mathrm{V}}^i, m_i)$
   m_i = [Endpoint$_i$, Placement$_i$, pk_{V}^i, pk_{asym}^i]
9: $\mathsf{PE}_i \rightarrow \mathsf{PE}_j$: Send m_i and $\mathrm{Sign}(sk_{\mathrm{V}}^i, m_i)$ to all other PEs
10: PE_i: Construct full manifest $M = \{(m_i, \mathrm{Sign}(sk_{\mathrm{V}}^i, m_i))\}$
11: $\mathsf{PE}_i \rightarrow D_i \rightarrow O$: Return M and $\sigma_i = \mathrm{Sign}(sk_{\mathrm{V}}^i, M)$
12: O $\rightarrow$ U: Return M and $\sigma = \sum_{i=0}^{n-1} \sigma_i$ to the user
13: U $\rightarrow$ PE_i: Connect to all PE_i via RA-TLS $[m_i, \mathrm{meas}_i]$
14: U: Compute aggregate $pk_{\mathrm{V}} = \sum_{i=0}^{n-1} pk_{\mathrm{V}}^i$ and validate σ
15: U: Generate withdrawal key pair $(sk_{\mathrm{W}}, pk_{\mathrm{W}})$
16: U $\rightarrow$ SC_D: Transaction Deposit(pk_{W}, pk_{V})
17: U $\rightarrow$ SC_R: Transaction Register(pk_{V}, M, sign(sk_{W}, "Reg"))
18: U $\rightarrow$ SC_E: Transaction Escrow(pk_{W}, pk_{V})
19: U $\rightarrow$ O: Sends enrollment finalization message
20: O: Checks state of SC_R, SC_E, and SC_D

4 DPaaS Design - Offline Setup

In this section, we will first describe the overview of the offline workflow for setting up a verifiable DPaaS instance, user enrollment, and verification from the user/builders, then we will elaborate on the key components among the steps.

4.1 Overview

Any principal can become a DPaaS *operator* by provisioning and operating a *deployment*. The operator chooses the size n and deploys n TEE-backed hosts through *cloud providers*. To realize the failure-isolation assumptions in our threat model in Sec. 3.2, the operator should place these hosts across *different* providers, availability zones, and TEE implementations. The operator acts only as an infrastructure coordinator: it provisions hosts and loads the attested DPaaS VM image, which includes the control software, Ethereum node, and PE application. Users and builders will reject deployments that do not: 1) execute a known image, 2) match the advertised placement information, and 3) conform to their trust assumptions (e.g., with respect to fault isolation).

We present the enrollment workflow in Algorithm 1. Initially, this starts with the user sending a request to enroll with the deployment operator (Line 1). The operator sends

a request to the control software running in each TEE in the deployment to launch a new PE with a base manifest that includes endpoint (i.e., IP address and port) and placement (i.e., provider, region, zone) information (Lines 2-3). The launched PEs start by generating two key pairs (Line 4): one to serve as the partial key for the aggregate validator key ($sk^i_{\mathrm{V}}, pk^i_{\mathrm{V}}$), and another for receiving encrypted bids from builders ($pk^i_{\mathrm{asym}}, sk^i_{\mathrm{asym}}$). The PEs will connect to each other through RA-TLS and perform mutual attestation based on the contents of the base manifest M^b and a known measurement (Line 5). If successful, the PEs exchange the public parts for their generated keys (Line 6) and then execute (n,n)-distributed key generation (DKG) to enable threshold signatures (Line 7). Each PE constructs its local manifest m_i, signs and exchanges it with all other PEs, and constructs the full manifest M (Lines 8-10). Afterward, each PE returns M along with its signature σ_i to the operator, who in turn sends it to the user with the aggregate signature σ (Lines 11-12). At this point, the user connects to all of the PEs and validates that their attestations match the manifest data along with a known measurement (Line 13). If successful, the user computes the public part of the aggregate signing key pk_{V} via the partial keys from each PE, and generates a withdrawal key pair $(sk_{\mathrm{W}}, pk_{\mathrm{W}})$ that they will use for becoming a validator (Lines 14-15). Finally, the user issues three smart contract transactions: 1) deposit to become a validator with signing key pk_{V}, 2) registering the manifest file to the pk_{V} so that builders can identify PEs for a given slot proposer, and 3) set funds aside in escrow for the operator to claim after successful proposals (Lines 16-18). The operator checks the state of the three smart contracts for successful enrollment (Lines 19-20); for instance, that pk_{V} serves as the validator key for an active validator in SC_D.

4.2 Details

Remote Attestation (Lines 5 and 13) Through remote attestation, users can verify the DPaaS instance configuration, and all PEs in the same DPaaS instance can mutually attest to one another. For mutual attestation, every PE establishes connections to all PEs through RA-TLS and verifies: 1) the PE runs on valid trusted hardware, 2) the PE's image matches a known, correct measurement, 3) the PE is deployed by the correct cloud provider, and 4) the PE has a specific placement within the cloud provider's infrastructure (*e.g.*, region, availability zone). We set the user data field in the attestation (e.g., report_data in AMD SEV-SNP) to be the hash of the base manifest (m^b_i). The same procedure follows for the user, except the user data field is set to the signed local manifest (m_i and $\mathrm{Sign}(sk^i_{\mathrm{V}}, m_i)$), which includes the public keys.

Validator Keys (Lines 4, 7, 14, 15, 16) In Ethereum, each validator can possess two distinct BLS key pairs [25]: 1) a validator key pair $(sk_{\mathrm{V}}, pk_{\mathrm{V}})$ for validator operations (e.g., block proposal and attestation), and 2) a withdrawal key pair $(sk_{\mathrm{W}}, pk_{\mathrm{W}})$ for controlling access to funds. We use this to allow PEs to manage the validator key sk_{V} for operational duties, while allowing users to retain sole control over the withdrawal key sk_{W} (and thus their funds).

Each PE_i first generates $(sk^i_{\mathrm{V}}, pk^i_{\mathrm{V}})$ independently. Moreover, to support threshold signatures [10] (with $t = f+1$), each party generates a partial BLS key pair $(sk^i_{\mathrm{p_sign}}, pk^i_{\mathrm{p_sign}})$ which enables interpolating the signature from any t PEs to that of a signature over the same message with sk_{V} (without PEs learning sk_{V}). DPaaS employs (n,n)-distributed key generation (DKG) protocol [69, 84] to achieve this without relying on a trusted third party. Since this only takes place during initial setup, we consider n out of n appropriate; if a PE misbehaves, then the procedure repeats until all PEs obtain shares that pass consistency checks. Finally, each PE_i generates a public-secret key pair $(pk^i_{\mathrm{asym}}, sk^i_{\mathrm{asym}})$, which builders use to encrypt secret shares for the auction (see Sec. 5).

Even though DPaaS is decentralized, the Ethereum network sees it as a single entity with the $(pk_{\mathrm{V}}, pk_{\mathrm{W}})$ identity. Moreover, sk_{V} is never exposed to any PEs or the user, and the withdrawal key sk_{W} is only known to the user.

Contracts (Lines 16, 17, 18, 20) SC_D refers to Ethereum's deposit contract. A user enrolls as a validator through DPaaS by depositing with a withdrawal key they control (pk_{W}) tied to the aggregate signing key from DPaaS PEs (pk_{V}). SC_R refers to the DPaaS registry contract, which stores PE metadata. The Register API allows deposited validators to associate a public signing key pk_{V} with a manifest M; the contract validates the signature against the pk_{W} and verifies it belongs to a validator. All parties can read the metadata locally with no additional cost (e.g., builders can determine PE endpoints for a DPaaS proposer slot). SC_E refers to the DPaaS escrow contract, deployed by an operator, which manages payments from enrolled users. The Escrow API allows a user to transfer funds to the contract associated with a given pk_{W} and pk_{V}. The Claim API allows an operator to recover a reward from these funds whenever a block proposed by pk_{V} is committed. We include additional details on these contracts in Appendix F.

5 DPaaS Design - Online Bid Collection

In this section, we discuss the first part of the online phase for DPaaS, focusing on how builders submit their bids to PEs (through the BT) and how each PE individually validates received bids to determine the set of eligible bids for the auction. We denote the builders participating in the slot as $\mathcal{B}$, where each $\mathtt{b} \in \mathcal{B}$ (identified by public key) submits one or more bids from the start of previous slot ($t = -12$s) to this slot's attestation deadline ($t = 4$s). For simplicity, we define each bid visible to every PE as a tuple $(\mathtt{b}, \mathtt{bv}, \mathtt{seq}, \mathtt{h}, \mathtt{att}, \mathtt{spl})$, which respectively refer to the builder, bid value (in $\mathbb{R}^*$), sequence number (in $\mathbb{N}$), block hash, TEE attestation generated by BT,

and sealed payload.

5.1 Builder Side

In practice, the first bid for slot S usually arrives 2–4 seconds into slot $S-1$, as builders must wait for the prior block to propagate to construct a valid block. Unlike prior proposals [41] and prototypes [39, 60], our design keeps untrusted builder software outside the TCB: the BT only validates the final block, handles secret sharing of the payload, and attests to the bid information.

PE Lookup and Verification. While relays have few, well-known endpoints that builders can use, DPaaS deployments and PEs are more decentralized. Each slot in an epoch identifies the validator assigned as the proposer by their pk_{V}; these assignments are known one epoch (i.e., 6.4 minutes) in advance. The builder queries the metadata stored in the DPaaS registry contract (SC_R) to check if the pk_{V} is enrolled through DPaaS; if so, the builder retrieves the corresponding manifest M from the storage. If the configuration (e.g., providers, TEE implementations) meets the builder's trust assumptions, the builder connects to the PE endpoints in M via RA-TLS (similar to what the user does in Sec. 4.2). If verification succeeds, the builder submits the block to the BT.

BT- Validation. The BT must validate the block and bid value (bv) submitted by a builder before generating the attested bid for the PEs. We support two approaches, *stateful* and *stateless*, which tradeoff TCB size and performance.

A stateful BT runs an execution node inside of the TEE. The BT reads the plain state and trie nodes from storage, checks the block's parent hash against the trie nodes, and simulates the transactions. If any transaction fails to simulate, or payment to the proposer is less than the bv, the BT returns an error. Note that the BT can employ full-disk integrity protection for the untrusted storage (*e.g.*, dm-integrity [86]) to detect unauthorized modifications, which is more performant than checking plain state and trie node consistency.

A stateless BT removes the need to run an execution node inside of the TEE, instead leveraging ideas from stateless validation [34] that uses *execution witnesses* to support block validation without needing access to the entire global state. In addition to the block and bid value, the builder is responsible for supplying the execution witness to the BT, which it can construct once it simulates the block (identifying state touched by the transactions in the block). The key difference is that the stateless BT needs to reconstruct the necessary subset of plain state from the witness, such that simulation can proceed as in the stateful case.

Overall, the stateless BT incurs more performance overhead as a result of the builder witness generation and BT state reconstruction and verification (see Sec. 8.2); however, the stateless BT achieves a smaller TCB by avoiding the need to rely on the storage (also reducing the cost of hosting a BT).

BT- Data Preparation & Submission. Finally, BT prepares the necessary data for constructing the bid. Following the definition in Sec. 6.2, the bid consists of $(\mathtt{b}, bv, \mathtt{seq}, h, \mathtt{att}, \mathtt{spl})$. Among these fields, the BT extracts $(\mathtt{b}, bv, h)$ directly from the builder's block (after checks). The BT provides the `seq` by keeping track of previous bids in the slot, starting from $\mathtt{seq} = 0$. The BT constructs the `spl` where $\mathtt{pl} = (\mathtt{header}, \mathtt{txs})$ for all $i = 0..n-1$:

$$(\mathtt{esshr} : \{\mathtt{E}_{\mathtt{pk}^{\mathtt{i}}_{\mathtt{asym}}}(K_i) \mid \mathtt{H}(K_i)\},\ \mathtt{header},\ \mathtt{E}_K(\mathtt{txs}),\ \mathtt{R}(\mathtt{pl}))$$

by first randomly generating a key K to encrypt the transactions as $\mathtt{E}_K(\mathtt{txs})$. The BT uses an $(f+1, n)-$secret sharing scheme to construct shares K_i where $i \in [0, n)$. For each K_i, the BT computes the hash $\mathtt{H}(K_i)$ and encrypts it for the PE_i $E_{pk^i_{\mathrm{asym}}}(K_i)$; the hash will be used by other PEs to verify decrypted shares during reconstruction (see Sec. 6.3). The `header`, including metadata (*e.g.*, `parent_hash`, `gas_limit`) of the block, will also be submitted. At this point, the BT generates a TEE attestation supplying the hash over all other elements of the bid including the block header as the user data field: $\mathtt{att} = \mathtt{att}\langle H(\mathtt{b}, bv, \mathtt{seq}, h, \mathtt{spl})\rangle$. The BT returns the bid to the builder software, which can send it to the PEs.

5.2 PE Side

Before the end of the auction, each PE_i listens for bids from builders. Upon receiving a bid r, it performs several checks: 1) verifies the $r.\mathtt{att}$ and signature over the carried user data; 2) sanity checks: $r.\mathtt{seq}$ adheres to contiguous ordering from 0 for the builder and unverified information in the `header` from the BT against the current chain state (e.g., `parent_hash`). If checks pass, PE_i marks r as eligible. To support top bid updates, as in the current builder ecosystem, builders can subscribe to a stream published by each PE_i and aggregate them (with a $f+1$ threshold).

6 DPaaS Design - Online Auction Finalization

In this section, we discuss the post-auction phase for DPaaS, focusing on PE coordination to determine the auction winner, reconstruct the winning block payload, and produce a signed beacon block to submit to the attesters. We first abstract the crux of decentralized auctioneer as a Crab-BB problem, and present a generalized protocol $\Pi_{\mathsf{Crab\text{-}BB}}$ to address it. We then formalize PBS auction semantics and present our runtime protocol Π_{DPaaS}, which builds atop $\Pi_{\mathsf{Crab\text{-}BB}}$.

6.1 Censorship-resistant and Availability-backed Byzantine Broadcast (Crab-BB)

The key challenges in the online phase of DPaaS lie in finalizing the winning block, which all honest PEs must agree on

to jointly reconstruct, sign, and propose. Builders submit bids continuously to all PEs, which should be considered until the auction deadline; however, due to varying network latency and clocks (or even malicious builders), each PE may observe a different set of bids. More importantly, Byzantine PEs may try to censor certain bids, which motivates the need for the protocol to be *censorship-resistant*: any bid considered eligible by all honest PEs must be considered during winner finalization. Additionally, Byzantine PEs may vote for a winning block but not be willing to participate in reconstruction, which motivates the need for the protocol to be *availability-backed*: there are sufficient honest PEs to reconstruct the block even when f PEs are compromised. Finally, given slot deadlines and MEV opportunities, this process must be as efficient as possible (i.e., minimal rounds of communication).

To address these challenges, we introduce censorship-resistant, availability-backed Byzantine Broadcast (Crab-BB):

Definition 1. *In the censorship-resistant, availability-backed Byzantine Broadcast (Crab-BB) problem, there are n replicas $P_0, \ldots, P_{n-1}$, of which one is designated as the leader P_ℓ. Furthermore, there are m clients $c_0, \ldots, c_{m-1}$. Each client c_j can have an input set U_j. Each replica P_i collects a set E_i, commits on an evidence value* Val*, and then produces an output set W. The following properties pertain to a given protocol Π with input preparation* Compress *and output interpretation* Interp *for the Crab-BB problem.*

- ***Agreement.*** *No two non-faulty replicas P_i disagree on the output subset W from Π.*
- ***Non-triviality.*** $\text{Val} \neq \perp$*, and must satisfy external validity.*
- ***Termination.*** *All correct replicas eventually decide.*
- ***Timely Termination.*** *After* GST*, all correct replicas decide within a bounded time.*
- ***Censorship-Resistant.*** *If* $\text{GST} = 0$*, then for all clients c_j that delivered their input $u \in U_j$ to all honest replicas (i.e., $u \in E_i$ for all honest replicas i), it holds that $u \dot{\in} W$.*
- ***Availability-backed.*** *For every $u \dot{\in} W$ output from Π, there must exist $f+1$ honest P_i with $u \in E_i$.*

This general definition uses a new relationship $\dot{\in}$, which we term *implicitly in*. We write $u \dot{\in} V$ to mean that the summary V represents object u. This representation may be explicit (e.g., set membership) or implicit (e.g., application-specific compression rule). Beyond standard BB, Crab-BB's censorship-resistance and availability-backed guarantees depend on the way replicas process their input set E_i and committed evidence Val. We therefore parameterize Crab-BB by two deterministic functions, Compress and Interp, which respectively prepare replica inputs and interpret the agreed evidence. The next subsection (Sec. 6.1.1) formalizes the requirements these functions must satisfy.

Input: Input set E_i; two functions: Interp, Compress.
Output: Interpreted output W.

S1. Input Preparation. $V_i \leftarrow \text{Compress}(E_i)$
S2. Exchange-1. Broadcast $\langle \texttt{ex-1}, V_i \rangle_i$.
S3. Exchange-2. Upon $T_e + \Delta$:
  a. If have received $> n-f$ distinct validly signed `ex-1` messages, let SQ_i denote the set of all such messages, broadcast $\langle \texttt{ex-fast-2}, SQ_i \rangle_i$.
  b. Otherwise, wait until it has received $n-f$ distinct validly signed `ex-1` messages. Let WQ_i denote the set of all such messages, and broadcast $\langle \texttt{ex-normal-2}, WQ_i \rangle_i$.
S4. Entering $\Pi_{vbb}(\mathcal{F})$. Wait until whichever of the following conditions is satisfied first:
  a. received $f+1$ distinct validly signed `ex-normal-2` messages; or
  b. received one validly signed `ex-fast-2` message.
  Enter Π_{vbb} with the corresponding evidence as input then executes Π_{vbb} until it commits a value Val.
S5. Output Interpretation. Apply a deterministic interpretation function $W \leftarrow \text{Interp}(\text{Val})$

Figure 3: $\Pi_{\text{Crab-BB}}$ with configurable input and output interpretation Interp for P_i. $\langle\rangle_i$ means a message signed by P_i.

6.1.1 Generalized Crab-BB Protocol

We present our $\Pi_{\text{Crab-BB}}$ in Fig. 3 consisting of 5 steps. Step 1 is a deterministic function for summarizing what each replica received. Step 2-4 form the consensus layer, which invokes a Validated Byzantine Broadcast (VBB) protocol, denoted by Π_{vbb}, as a subroutine. The external validity of Π_{vbb} is defined by Algorithm 3 in Appendix B. In our setting, we use the $(5f-1)$-psync-VBB protocol [52], achieving two-round good-case latency at the minimum resilience threshold[1]. Step 5 is the output interpretation layer implemented to get the final agreed output set in a deterministic way. The protocol starts with the input set E_i to each replica. To begin with, we require Compress to be representation-preserving: for every local input set E_i and every object $u \in E_i$, $u \dot{\in} \text{Compress}(E_i)$.

Then each replica broadcasts a signed `ex-1` message containing its local set E_i (Step 2). During the subsequent Δ time, replicas collect `ex-1` messages from their peers. If a replica receives $> n-f$ distinct validly signed `ex-1` messages, it forms a strong quorum certificate, denoted by SQ (Step 3a). Otherwise, it continues waiting until it receives $n-f$ distinct validly signed `ex-1` messages, at which point it forms a weak quorum certificate, denoted by WQ (Step 3b). In either case, once a replica forms a certificate, it signs and broadcasts it.

Next, replicas wait until they obtain either (i) one SQ or (ii) $f+1$ distinct WQs. An honest replica enters Π_{vbb} after receiving one of these two valid inputs, and uses the corresponding evidence as its input to the broadcast protocol

[1] We note that for $n \leq 5f-2$, using two rounds is impossible and one additional round is necessary [52].

(Step 4). Assuming correctness of Π_{vbb}, all honest replicas eventually commit the same valid output, which must be either a single SQ or a set of $f+1$ distinct WQ values proposed by some replica. Given a committed evidence value Val, let $\mathcal{C}(\text{Val})$ be the collection of summaries extracted from Val. The threshold-supported object set $\mathsf{Supp}_{2f+1}(\text{Val})$ is:

$$\left\{u \;\middle|\; |\{V_j \in \mathcal{C}(\text{Val}) : u \overset{\bullet}{\in} V_j\}| \geq 2f+1\right\}.$$

The interpretation function (Step 5) outputs a canonical summary of this set: $\mathsf{Interp}(\text{Val}) = \mathsf{Compress}(\mathsf{Supp}_{2f+1}(\text{Val}))$.

6.1.2 Correctness

We describe the intuition here and defer the full proof of $\Pi_{\mathsf{Crab\text{-}BB}}$ to Appendix B. Agreement follows from Π_{vbb}: all honest replicas commit the same Val and apply the same deterministic interpretation rule. Termination follows because, after GST, the exchange phase completes within 2Δ, after which all honest replicas enter Π_{vbb}. Non-triviality follows from the external validity of Π_{vbb}, which only allows values carrying valid evidence. Censorship resistance follows from the $2f+1$ interpretation threshold. Consider an input u received by all honest replicas, so $u \overset{\bullet}{\in} V_i$ for every honest i. If a SQ is committed, it contains at least $n-f+1$ summaries, of which at least $n-2f+1 = 3f$ are honest; hence u is represented by at least $2f+1$ summaries and is included in W. If only WQs are available, external validity requires $f+1$ such certificates, including at least one from an honest replica; that certificate contains at least $n-f = 4f-1 \geq 2f+1$ summaries representing u, again ensuring $u \overset{\bullet}{\in} W$. Finally, backed availability follows from the same threshold: any input represented by $2f+1$ replicas is backed by at least $2f+1-f = f+1$ honest replicas.

6.2 Application to DPaaS

We firstly define bid eligibility in DPaaS. In status-quo MEV-Boost, a relay marks a bid as eligible once its block is validated. In contrast, DPaaS runs auctions in a decentralized setting, which necessitates distinguishing between eligibility local to a given PE_i versus globally across PEs. We follow the definition of bid in Sec. 5. For each PE_i, we refer to the set of bids it receives from builders as R_i. For any set of bids S, we refer to the subset from builder $\mathtt{b}$ as $S[\mathtt{b}]$.

Locally, for each builder, PEs treat the highest-sequence eligible bid as active; thus, bid cancellation [62] is supported by canceling lower-sequence stale bids. If a builder submits malformed sequence numbers, those bids fail the eligibility checks and are excluded from winner selection. Globally, eligibility and activeness is with respect to $\mathcal{E}_i$ for honest PEs. Bids seen by all honest PEs are guaranteed globally eligible, while others may also qualify with support from Byzantine PEs. However, every bid considered globally eligible must be supported by at least $f+1$ honest PEs, and any bid locally eligible at all honest PEs is globally eligible.

Definition 2. *In the DPaaS instance with protocol Π_{DPaaS}, we can define the following eligibility concerning bids:*

- ***Local Eligibility.*** *Given received bid set R_i, the set of PE_i's locally eligible bids $\mathcal{E}_i$ is:*

$$\mathcal{E}_i = \big\{r \,|\, r \in R_i \wedge \mathsf{PE}_i \text{ has verified } r.\mathtt{att} \wedge \forall s \in [0, r.seq), \exists r' \in \mathcal{E}_i[r.\mathtt{b}] : r'.seq = s\big\}$$

- ***Local Active.*** *Given locally eligible bid set $\mathcal{E}_i$, the set of PE_i's locally active (non-canceled) bids $\mathcal{V}_i$ is:*

$$\mathcal{V}_i = \bigcup_{\mathtt{b}\in\mathcal{B}} \arg\max_{e\in\mathcal{E}_i[\mathtt{b}]}(e.\mathtt{seq})$$

- ***Global Eligibility.*** *Given all PE_i's locally eligible bids $\mathcal{E}_i$ and honest PEs' set $\mathcal{HP}$, we define globally eligible bid set $\mathcal{E}$ as $\check{\mathcal{E}} \subseteq \mathcal{E} \subseteq \hat{\mathcal{E}}$ where*

$$\check{\mathcal{E}} = \{e \mid n-f \leq \big|\{i \in \mathcal{HP} \mid e \in \mathcal{E}_i\}\big|\}$$
$$\hat{\mathcal{E}} = \{e \mid f+1 \leq \big|\{i \in \mathcal{HP} \mid e \in \mathcal{E}_i\}\big|\}$$

- ***Global Active.*** *The set of globally active bids is the bid with highest sequence number with respect to every builder.*

$$\mathcal{A} = \bigcup_{\mathtt{b}\in\mathcal{B}} \arg\max_{e\in\mathcal{E}[\mathtt{b}]}(e.\mathtt{seq})$$

We formalize goals **G1** through **G4** and the bid-cancellation aspect of **G6** (see Sec. 2.2) through the following composite security definition.

Definition 3. *A protocol Π satisfies DPaaS security goals if, for every slot and every f-bounded adversary, the following hold under partial synchrony with auction deadline T_e and proposal deadline T_p. After GST, builder-PE latency is bounded by Δ_{b2p}, and bid processing latency in PE is bounded by Δ_{pr}.*

3.1 ***Bid Selection Correctness*** *(G1). If $T_e \geq$ GST, then the winning bid w is the globally active bid with the highest bid value: $w = \arg\max_{e\in\mathcal{A}} e.\mathtt{bv}$*

3.2 ***Block Validity*** *(G2). For the winning bid w, the reconstructed payload $w.\mathtt{pl}$ pays at least $w.\mathtt{bv}$ to the enrolled user, and honest Ethereum validators accept the payload as valid.*

3.3 ***Block Availability*** *(G3). If at least one valid bid becomes eligible at all honest PEs, then PEs jointly reconstruct and produce the corresponding signed beacon block.*

3.4 ***Timely Block Availability*** *(G3). If at least one valid bid becomes eligible at all honest PEs before the cutoff T_p, then PEs jointly reconstruct and produce the corresponding signed beacon block before T_p.*

3.5 ***Failed Trade Privacy*** *(G4). For each slot, for every non-winning bid, no information beyond payload size and submission timing is released to any party.*

Input: `slot`, T_e **Output:** `SignedBeaconBlock`

P1. Bid Collection.
- **a.** PE_i starts accepting bids one slot before `slot`.
- **b.** Until T_e, accept bids from BTs and insert each received input into $\mathcal{E}_i$ if $\text{TEEVERIFY}(r.\texttt{att}) = \text{TRUE}$ and $\text{SANITYCHECK}(r) = \text{TRUE}$.

P2. Winner Finalization.
- **a.** Agree on $\mathcal{A} \leftarrow \Pi_{\mathsf{Crab\text{-}BB}}(\mathcal{E}_i, \mathsf{AppCompress}, \mathsf{AppInterp})$.
- **b.** Find the winner $w \leftarrow \arg\max_{a \in \mathcal{A}}(a.\texttt{bv})$.

P3. Reconstruction and Signing.
- **a.** Reconstruct and sign `SignedBeaconBlock` $\leftarrow \Pi_{rec}(w)$.

Figure 4: DPaaS's runtime protocol Π_{DPaaS}.

3.6 ***Cancellation Effectiveness*** *(**G6**). For each builder* $\texttt{b} \in \mathcal{B}$*, let* e *be the most recent bid from* `b` *observed after* GST *and before* $T_e - \Delta_{b2p} - \Delta_{pr}$*. Then every earlier bid* $e' \in \mathcal{E}[\texttt{b}]$ *with* $e'.\texttt{seq} < e.\texttt{seq}$ *is considered canceled and cannot be selected as the winner.*

Intuitively, a DPaaS protocol must provide the following guarantees. First, the auction phase admits only valid bids; we enforce this using BTs that validate bids before acceptance by PEs. Second, the winner finalization ensures outcome agreement for all honest PEs; this is provided by $\Pi_{\mathsf{Crab\text{-}BB}}$. Third, the outcome must be correct under auction semantics: the winner w is the highest bid in the globally active set $\mathcal{A}$, the latest subset of globally eligible bids defined by $\check{\mathcal{E}}$ and $\hat{\mathcal{E}}$. This last guarantee is not implied by agreement alone, because it requires the agreed output set to satisfy the correctness of the auction: bids that should be included by censorship resistance must remain represented, while bids in the output must be backed by enough replicas for availability. These are precisely the censorship-resistance and backed-availability guarantees provided by $\Pi_{\mathsf{Crab\text{-}BB}}$.

Based on the above, we provide Π_{DPaaS} in Fig. 4 consisting of 3 phases. P1 captures the online bid collection (as discussed in Sec. 5), while we discuss P2 and P3 next.

6.3 Post-Auction - P2. & P3.

After the auction deadline T_e, the PEs need to jointly determine the winning bid w from all of the locally eligible sets $\mathcal{E}_i$ (*finalization*), reconstruct the block payload from $w.\texttt{spl}$ (*reconstruction*), and finally output an instance-level signature for the Signed Beacon Block (*sign*). We describe the high level idea and present the detailed protocol in Appendix A.

Finalization. We instantiate $\Pi_{\mathsf{Crab\text{-}BB}}$ with the semantics that real builders as clients submit a stream of bids with continuous sequence number. Therefore, $\mathcal{E}_i[b]$ at honest PEs only contain contiguous sequence numbers from 0, each $\mathcal{V}_i$ need only to keep the highest-sequence bid, which implicitly captures earlier ones. We apply a summarization function AppCompress (Algorithm 2 in Appendix A) to only include the active bid for each builder. We also adapt the post-consensus function AppInterp (Algorithm 2) to operate on consensus output, ensuring that it preserves implicit inclusion.

Reconstruction. Given a winning bid w, PEs must now reconstruct the block payload by decrypting the `txs` from $w.\texttt{spl}$. As shown in Fig. 9 in Appendix A, each PE_i starts by decrypting its own share using its private key and broadcasts the share to all PEs. When receiving a share, PE_i checks the validity of the share by comparing the hash of the K_j it receives with the hash contained in $w.\texttt{spl}$. Once PE_i receives valid shares from at least $f+1$ PEs, it can reconstruct the symmetric key K and decrypt the $\mathtt{E}_K(\texttt{txs})$ contained in $w.\texttt{spl}$ to get `pl`.

Sign. At least $f+1$ PEs obtain plaintext payload for the winning bid w and must produce a signed beacon block. Since the beacon block needs to be signed over all block fields, including `pl` and `cdata`, we include $\mathtt{R}(\texttt{pl})$ in plaintext within $w.\texttt{spl}$ instead of reconstructing `pl` first. This enables parallel reconstruction and interpolation, reducing latency. Each PE_i generates a partial signature σ_i and broadcasts it. After collecting $f+1$ valid partial signatures, PE_i uses Lagrange interpolation to compute the full signature (verifiable under pk_V) and propagates the Signed Beacon Block.

7 Implementation

Our prototype implementation consists of several core components: BT and DPaaS library for builders, and the Control and PE applications for deployments. We implement the components in Rust, using Tonic [51] to support RPC-based communication. For TEE components, we provide implementations for both AMD SEV-SNP [55] and Intel TDX [53]. We use Reth [67] and Lighthouse [71] as the Ethereum Node for PEs. We validate DPaaS end-to-end functionality through a private testnet supported by ethereum-package [37].

Builder TEE (BT) For stateful BT, we use the simulation crate from Reth [67] and store chain state on a dm-integrity [86] volume with a HMAC-SHA256 key derived from SNP firmware. For stateless BT, we use crates from stateless [68] for handling execution witnesses. We implement data preparation for the `spl` based on the open-source Shamir Secret Share library rust bindings [80] and asymmetric encryption crate age [50]. For `spl`, we generate the common hash via crate [74] and the Merkle root of the payload $\mathtt{R}(\texttt{pl})$ via the tree_hash crate [75].

Proposer Entity (PE) We implemented four services for PEs that capture different phases of DPaaS: Bootstrap, Auction, Consensus, Recovery. For Bootstrap, we use RA-TLS for constructing secure channels to PEs. Due to the lack of Rust crate at the time of our implementation, we created the implementation for AMD SEV-SNP and Intel TDX that build on top of the Rust bindings for OpenSSL [73]. For Auction, we cache keys fetched for verifying TEE attestations (e.g., VCEK for AMD SEV-SNP) to enable faster verification for

subsequent bids from the same builder. For Consensus, we implement $\Pi_{\text{Crab-BB}}$ with our application-specific compress and interp functions (see Algorithm 2 in Appendix A) for both $n = 5f - 1$ (Fig. 3 is the general protocol, Appendix E is its optimization when $f \geq 2$) and $n = 3f + 1$ setting (by adding one more round of vote in Π_{vbb} to meet the lower bound [2]). Additionally, we use BLS signatures for messages relayed between PEs. For Block Recovery, we use the blsttc library [61] for threshold BLS signatures.

Limitations Current TEE attestations identify the cloud provider but not the availability zone, since provider endorsement keys are shared across zones [5]. Our prototype therefore obtains zone information from provider metadata services (e.g., AWS IMDS [4]), which are outside the TEE and thus cannot cryptographically prove fault isolation across zones. Users requiring stronger isolation can instead select DPaaS deployments across n distinct cloud providers, whose identities are attested, avoiding reliance on zone metadata. Future data-center-level attestation [72] can provide the finer-grained placement guarantees directly.

8 Evaluation

In this section, we evaluate DPaaS across four dimensions: BT performance overhead (Sec. 8.2), PE bid processing performance (Sec. 8.3), PE post-auction finalization performance (Sec. 8.4), and macro-benchmark evaluation for MEV over historical bid traces (Sec. 8.5).

8.1 Dataset and Configuration

We use the full bid dataset from Ultra Sound Relay [88], which records the historical bids during 2023/09-2023/12. To the best of our knowledge, this is the most recent publicly available dataset of its kind and remains representative of today's market in bid value, per-slot bid throughput, and builder participation [43]. We also use the canonical block dataset from ethPandaOps [36], which records consecutive 6k on-chain block from block number 25,662,613 to 25,668,612.

We deployed DPaaS on 4 CVMs across 4 availability zones provided by Amazon AWS and Microsoft Azure. Fig. 5 shows the round trip times measured between different CVMs. Each CVM has 4 vCPUs and 16 GB of memory. We deployed a BT in one CVM in AWS, with 8 vCPU and 32 GB memory. For our $n = 9$ experiments, we also deployed 9 AMD SEV-SNP CVMs, each with 4 vCPUs and 16 GB memory, on a single server with an AMD EPYC 9354.

8.2 BT Overhead

To evaluate the BT overhead for builder bidding, we simulate the builder submissions to stateful and stateless BTs by running consecutively for the 6,000 slots in the canonical block

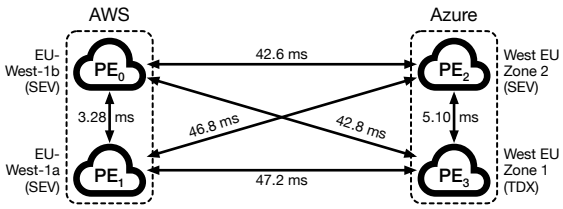

Figure 5: DPaaS deployment topology for our evaluation ($n = 4$), labeled by ping RTTs.

	Block #s	Builder	BT.Val	BT.Prep	TEE	Total
Stateful	1		2889	1.312	7.132	2897
	1 – 1000		428.1	1.320	7.254	436.5
	Replay		75.12	1.326	7.557	84.00
Stateless	1	836.1	134.2	1.312	7.522	979.1
	1 – 1000	612.3	145.8	1.370	7.631	767.1
	Replay	270.1	156.1	1.389	7.534	435.1

Table 1: Latency breakdown for BT overhead (ms).

dataset. For the stateless BT, the builder prefetches trie nodes from the transactions while assembling the block to reduce execution witness generation latency. We evaluate for the first block (1), average of the first thousand blocks (1-1000), and an average of five resubmissions for every block (Replay).

Table 1 presents the detailed latency breakdown. Data preparation and TEE attestation generation are lightweight operations (around 1.3 ms and 7.5 ms respectively): the former is a function of the payload size, while the latter is constant. Stateful validation incurs higher cold-start latency (1st) due to needing to fetch relevant state from disk, while stateless validation includes all such state in the execution witness. However, as the in-memory EVM state cache warms up (1000th), the stateful validation reduces end-to-end latency relative to stateless validation (436.5 ms versus 767.1 ms). We observe that execution witness generation, captured by builder-side latency (Builder), remains the dominant cost for stateless validation. We also see this in the case where a builder resubmits the same block (Replay), which is meant to explore the bound on performance improvement when the builder makes minor changes to the block; while stateful decreases significantly down to 75.12 ms, stateless still incurs a total of 426.2 ms for generation and verification. We provide additional results in Appendix D.2.

8.3 Bid Processing Performance

We evaluate a single PE_i from receiving a bid to inserting it into the locally eligible set. The BT client and PE_i run on two cloud CVMs, with the client replaying valid bids at 100–700 bids/s. Figure 6 shows mean latency increases from 1.43 ms at 100 bids/s to 5.91 ms at 600 bids/s. Attestation and integrity verification remain stable at approximately 0.77–0.90 ms and

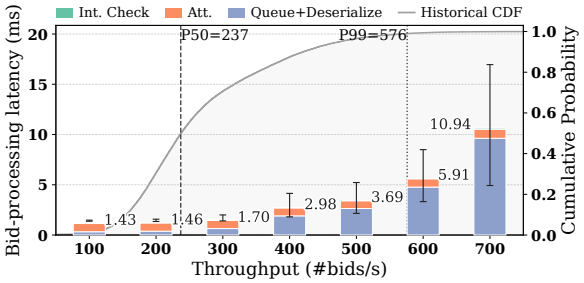

Figure 6: PE bid processing latency versus throughput. Background CDF shows historical throughput observed by relays.

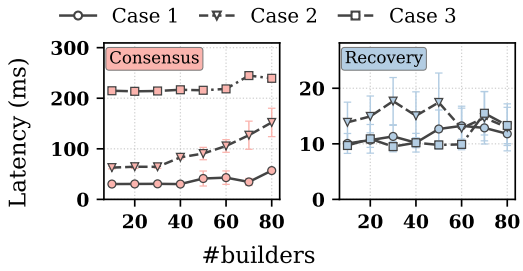

Figure 7: The consensus and block recovery latency (ms) under cloud deployment ($n = 4$).

0.09–0.12 ms, respectively. The increase therefore comes primarily from Queue+Deserialize, which grows from 0.33 ms to 4.77 ms. Since 600 bids/s exceeds the historical P99, a single PE handles at least 99% of historical peak windows with mean latency below 6 ms. At 700 bids/s, latency reaches 10.94 ms with greater variance, indicating queue buildup.

8.4 Post-Auction Performance

We randomly select 10 slots in 2023/12, input the eligible bid sets to all n PEs, and measure the consensus and block recovery latency. We consider three different scenarios: optimistic fast path (Case 1), only follower crash (Case 2), and first leader crash (Case 3). We set $\Delta = 30$ ms based on halving the 99th-percentile inter-PE RTT over 10,00 samples and vary the builders in each PE's $\mathcal{V}_i$ from 10 to 80. We cap at 80 since it is the 99th percentile in our dataset; recent data [43] shows 50 builders exist during June 1–7, 2026, and the builder market is shrinking [91].

We evaluate $n = 4$ in real cloud deployment setting. To assess larger deployments, we evaluate $n = 7, 9$ in an emulated WAN testbed. Appendix E.2 shows the resulting trade-offs between committee size, communication rounds, and latency. For each point, we record 5 runs and compute the average. We do not evaluate $n > 9$: DVT deployments [30], the closest operational analogue, sit at $f = 1$ or 2 in practice [59, 64].

Cloud Deployment. The results are in Fig. 7. For consensus latency (from the end of the auction to $2f + 1$ PEs output a winning bid), in Case 1, the leader constructs SQ after receiving messages from all four PEs, completing consensus in three rounds. Consensus latency remains low as the number of builders increases, reaching only 56.9 ms at 80 builders. In Case 2, a follower crash forces the leader onto the fallback path, which waits up to 2Δ before entering Π_{vbb} and requires $f + 1$ WQ messages. Consequently, latency becomes more sensitive to the amount of builder metadata exchanged, increasing to 152.1 ms at 80 builders. In Case 3, the initial leader crashes, triggering a view change after timeout; the new leader is then honest and commits within two additional rounds, yielding 239.4 ms at 80 builders.

The recovery overhead remains comparatively small across all cases—about 11.8–13.3 ms. Since secret-share recovery and block signing are independent, our implementation executes them in parallel. In the best case (Case 1), the post-auction latency is only $56.9 + 11.8 = 68.7$ ms.

8.5 Macrobenchmark

We randomly selected 3,000 slots in 2023/12 and replayed all the bids during those slots to our DPaaS instance. Due to limited data availability regarding round-trip times between relays and builders in the builder market, we replayed bids based on the `ReceivedAt` timestamps in the dataset. Because stateless witness generation requires a historical block tip in 2023/12 which is unavailable today, for a fair comparison, we have to artificially add the warmed-up latency shown in Table 1 (84 ms for stateful mode and 435.1 ms for stateless mode) when replaying the bids

We then analyzed the winning bid values in both DPaaS and the relay under different target block-proposal times: for DPaaS we used the user-configured T_p, while for the relay we consider the time `getHeader` with an offset of +880 ms to account for the average time before the `getPayload` (measured across 5,063 slots from the Ultra Sound Relay dataset). To match DPaaS 's use case, where users specify an upper bound on post-auction latency relative to T_p, we evaluated both the best case (Case 1) and worst case (Case 3).

We present our results in Figure 8. At $t = 0$ s, which is the Ethereum-specified time for honest block proposal, DPaaS yields 0.0777 ETH (best case) and 0.0773 ETH (worst case), compared to 0.0752 ETH for the relay (2.8%-3.3% improvements). In contrast, if DPaaS operates under the stateless mode, the earning is only 0.0762 ETH (best case) and 0.0754 ETH (worst case). Although both DPaaS and the relay converge after $t = 2$ s when no new bids arrive, such delayed proposals risk not receiving sufficient attestations [78] leading to a missed slot. Due to the variability of actual network latency in practice relative to our setting of Δ, it is possible that DPaaS may infrequently miss the target proposal time (which would not be captured in the plot). In this experiment, DPaaS proposed the block ahead of T_p for all 3,000 slots.

Notes. We do not separately evaluate multiple validators shar-

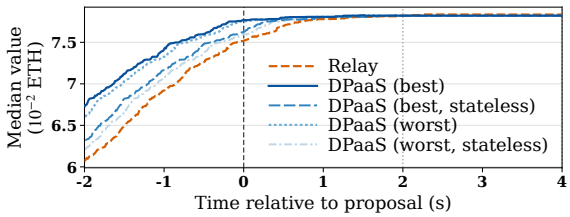

Figure 8: Median winning bid value based on different target proposal times (T_p) for DPaaS versus Relays.

ing a deployment because their primary signing duties are naturally staggered by Ethereum's slot schedule. For a proposer in slot s, DPaaS's latency-critical auction work is concentrated near the end of slot s and completes around the start of s, whereas ordinary attesters for $s-1$ sign around 4 s into that slot and attesters for s sign around 4 s after the proposal [57]. Thus, proposer finalization does not overlap with the dominant attestation-signing load.

9 Discussion

Costs/Incentives. DPaaS's guarantees are not free: builders must run a BT and users must pay for a DPaaS instance. However, DPaaS largely does not introduce new costs; it *makes explicit and reallocates costs that relays already incur*. For *Monetary cost*, the compute and storage used by the BT replace resources already provisioned by relays for block validation: our 8-vCPU CVM costs 0.36 USD/h, with an additional 0.12 USD/h for stateful storage. Since BT need not run continuously, its effective cost can be substantially lower. A DPaaS deployment costs 0.66 USD/h and can be amortized across multiple users. Its relative cost depends on the ETH price, staking APR, and stake under management. At the 2.6% staking APR observed when this paper was written [31], for example, a validator with 32 ETH and an ETH price of 2,456 USD (at the time the paper was written) earns roughly 2,043 USD/year; three such validators' earning can exceed the 5,782 USD annual cost of one DPaaS deployment. Higher ETH prices or larger balances further reduce the relative overhead. Thus, the cost can be amortized with only a small number of validators, well within the multi-validator deployments supported today (5,850 Ethereum clients supporting over 800k validators [35]). For *Operational cost*, DPaaS requires operators to maintain these services, but relays already perform the same always-on infrastructure management. The difference is that these costs are currently hidden behind a free trusted intermediary: relays charge no fees, and Blocknative exited the relay market in 2023 citing economic unviability [17].

In fact, current relay leads directly to an incentive problem. A trustworthy intermediary with recurring costs but no direct revenue must either be subsidized as a public good or recover value through adjacent roles, creating incentives for vertical integration with builders and searchers [9, 65]. DPaaS instead prices the trust it provides: builders and validators that benefit from attested, threshold-replicated execution directly fund its operation. Thus, DPaaS just makes that cost explicit and gives the trusted service a sustainable incentive model. As an additional builder benefit, losing blocks remain secret-shared and are never reconstructed, eliminating failed-trade leakage by construction rather than relying on a relay's promise. We thus envision a positive incentive loop: builders with valuable MEV prefer DPaaS proposers for stronger privacy, increasing proposer revenue and, in turn, adoption of DPaaS.

Centralization. We distinguish design-level trust from deployment-level centralization. At the design level, DPaaS replaces relay trust with a threshold assumption over PEs: one relay failure will not lead to the loss of all guarantees. Each instance can publish its PE composition and attestation metadata, enabling verification across operators, regions, and TEE vendors, further reducing opacity and single points of failure. At the deployment level, centralization depends on adoption (e.g., validator concentration or dominant PE operators) and must be evaluated empirically after adoption.

10 Related Work

We review other related work beyond PBS proposals (Sec. 2).

Censorship-Resistant Byzantine Broadcast. There are two concurrent works that formalize definitions similar to Crab-BB. First, Abraham *et al.* [1, 3] formalize censorship-resistant Byzantine broadcast (CRBB), a closely related problem that seeks to prevent consensus from ignoring sufficiently disseminated client inputs. However, CRBB does not solve the backed-availability required by DPaaS. They present a lower bound on the latency required to obtain this property; our protocol matches this lower bound while also obtaining backed availability. Another concurrent work, Multiple Concurrent Proposers (MCP) [46], also targets Crab-BB while matching the CRBB lower bound. However, MCP is designed for larger committee with Byzantine-fraction smaller than $1/5$. Their security guarantees can't hold under the most practical setting $n = 4, f = 1$ in DPaaS. We added one round in $\Pi_{\text{Crab-BB}}$ and fill this gap. Moreover, we show it can be optimized when $f > 1$ or $n > 4f$ to meet CRBB's lower bound in Appendix E.

Decentralized Auction. Previous protocols [7, 44, 48, 63, 87, 95] focus on decentralizing sealed-bid auction. In contrast, we focus on an open-bid, sealed-payload decentralized auction while supporting bid-cancellation. However, previous works either treat bid cancellation as out of scope [7, 44, 48, 63, 95] or malicious behavior [87].

11 Conclusion

Proposer-Builder Separation is central to Ethereum's decentralization, yet MEV-Boost relies on trusted relays that in-

troduce centralization and security risks. We presented Decentralized Proposer-as-a-Service, which distributes proposer and relay functionality across Proposer Entities. Our evaluation shows that DPaaS achieves low latency while preserving competitive MEV capture.

Acknowledgements

This work is supported by Flashbots MEV Fellowship Grants. We are grateful to Ultra Sound Relay and Sen Yang for sharing the full-bid dataset and providing access to relay-timestamp information. We also appreciate the valuable discussions with Jonathan Passerat-Palmbach and Quintus Kilbourn.

Ethical Considerations

Our work studies infrastructure for Ethereum block production, a live DeFi system. We designed our evaluation to avoid interaction with the Ethereum mainnet. Our end-to-end experiments run on an emulated setting, while experiments reflecting real workloads replay historical data rather than submitting bids or blocks to production relays or Ethereum mainnet. Consequently, our experiments do not influence real block selection, validator rewards, builder auctions, or user transactions, and do not place real validator funds at risk.

Our evaluation uses publicly available datasets: historical bid data released by Ultra Sound Relay and canonical Ethereum blocks from ethPandaOps. These datasets contain protocol-level information such as builder public keys, bids, timestamps, and public blockchain data. We use these records only to characterize workloads and replay protocol behavior; we do not attempt to deanonymize participants or infer information about individual Ethereum users. Where builder identifiers are useful for analysis, we use public protocol identifiers or abbreviated public-key prefixes and otherwise report aggregate results.

DPaaS may nevertheless have broader economic consequences. In particular, its lower post-auction latency can allow proposers to accept later bids and thereby increase MEV revenue. MEV itself can have negative externalities for Ethereum users, and our goal is not to develop new MEV extraction or transaction-ordering strategies. DPaaS operates after builders construct blocks and instead addresses how mutually distrustful builders and proposers exchange those blocks. Our intended benefit is to reduce reliance on trusted relays, prevent individual faulty PEs from censoring eligible bids, protect non-winning builder payloads, and improve availability. The censorship-resistance property studied in this paper is specifically bid-level censorship resistance among DPaaS PEs; it should not be interpreted as a policy position or guarantee concerning transaction-level censorship or content.

Finally, DPaaS should not be interpreted as eliminating trust or deployment risk. Its guarantees rely on the stated fault threshold, TEE integrity assumptions, and sufficient independence among PE deployments. Correlated TEE vulnerabilities, cloud-provider failures, incorrect placement information, or faults exceeding the assumed threshold could lead to privacy, availability, or financial consequences. Our prototype is intended as a research demonstration rather than production-ready validator infrastructure, and deployment on Ethereum mainnet would require additional implementation hardening, security auditing, and careful validation of the underlying TEE and infrastructure assumptions. We disclose these assumptions and limitations so that the decentralization and performance benefits reported in this paper are not interpreted as unconditional security guarantees.

Open Science

To promote transparency and reproducibility, the source code for our prototype and all benchmarks used in our experimental evaluation are publicly available at https://anonymous.4open.science/r/DPaaS_Proj-CBD2/. The complete evaluation dataset exceeds the storage limits of our anonymous repository. We therefore provide a representative sample for review and will archive the full dataset on a persistent public repository after de-anonymization.

References

[1] Ittai Abraham, Yuval Efron, and Ling Ren. The latency cost of censorship resistance. *IACR Cryptol. ePrint Arch.*, 2025:2136, 2025. URL: https://eprint.iacr.org/2025/2136.

[2] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case Latency of Byzantine Broadcast: a Complete Categorization. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2021.

[3] Abraham, Ittai and Efron, Yuval and Ren, Ling. Latency cost of censorship resistance. https://decentralizedthoughts.github.io/2026-04-23-latency-of-censorship-resistance/, 2026. accessed: 2026-8-26.

[4] Amazon AWS. Use instance metadata to manage your EC2 instance, October 2025. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html.

[5] AMD. Versioned Chip Endorsement Key (VCEK) Certificate and KDS Interface Specification. https://docs.amd.com/api/khub/documents/dWGhwYpo1Wv51rJN4d~47g/content. accessed: 2026-8-26.

[6] AWS. Precision clock and time synchronization on your EC2 instance. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/set-time.html.

[7] Samiran Bag, Feng Hao, Siamak F. Shahandashti, and Indranil G. Ray. SEAL: Sealed-bid auction without auctioneers. Cryptology ePrint Archive, Paper 2019/1332, 2019. URL: https://eprint.iacr.org/2019/1332.

[8] Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. Centralization in block-building and proposer-builder separation. In *Financial Cryptography and Data Security (FC)*, 2024.

[9] Blocknative. Lessons earned one year post-merge: Unraveling pbs and its effect on mev, block building, and the ethereum network. https://hackernoon.com/ethereum-merge-one-year-later-what-weve-learned-about-block-building-mev-boost-and-relays, 2023. Accessed: 2026-08-24.

[10] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography - PKC, 6th International Workshop on Theory and Practice in Public Key Cryptography*, 2003.

[11] Vitalik Buterin. Proposer/block builder separation-friendly fee market designs. https://ethresear.ch/t/proposer-block-builder-separation-friendly-fee-market-designs/9725, June 2021.

[12] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. EIP-1559: Fee market change for ETH 1.0 chain, April 2019. Ethereum Improvement Proposals, no. 1559. URL: https://eips.ethereum.org/EIPS/eip-1559.

[13] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and casper. *arXiv preprint arXiv:2003.03052*, 2020.

[14] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *IEEE European Symposium on Security and Privacy*, 2019.

[15] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[16] Jalen Chuang, Alex Seto, Nicolas Berrios, Stephan van Schaik, Christina Garman, and Daniel Genkin. Tee.fail: Breaking trusted execution environments via DDR5 memory bus interposition. In *IEEE Symposium on Security and Privacy*, 2026.

[17] CoidDesk. Blocknative to suspend mev-boost relay after economics fail to 'materialize'. https://www.coindesk.com/tech/2023/09/26/blocknative-to-suspend-mev-boost-relay-after-economics-fail-to-materialize, 2023. accessed: 2026-8-26.

[18] Victor Costan and Srinivas Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016.

[19] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE Symposium on Security and Privacy*, 2020.

[20] Francesco D'Amato, Nico Flaig, Barnabé Monnot, Michael Neuder, Potuz, Justin Traglia, and Terence Tsao. EIP-7732: Enshrined Proposer-Builder separation. https://eips.ethereum.org/EIPS/eip-7732, June 2024. Ethereum Improvement Proposals.

[21] Jesse De Meulemeester, David Oswald, Ingrid Verbauwhede, and Jo Van Bulck. Battering RAM: Low-cost interposer attacks on confidential computing via dynamic memory aliasings. In *IEEE Symposium on Security and Privacy*, 2026.

[22] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[23] EspressoSystems. Fair Exchange in Proposer-Builder Separation, October 2023. URL: https://hackmd.io/@EspressoSystems/fair-exchange-in-proposer-builder-separation.

[24] Ethereum. Simpleserialize (ssz). https://github.com/ethereum/consensus-specs/blob/v1.3.0/ssz/simple-serialize.md, 2021. accessed: 2025-07-01.

[25] Ethereum. Keys in proof-of-stake ethereum. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/keys/, 2025. accessed: 2026-8-26.

[26] Ethereum Foundation. How to stake your ETH. https://ethereum.org/en/staking/, 2024. accessed: 2026-8-26.

[27] Ethereum Foundation. Proposer-Builder Separation. https://ethereum.org/en/roadmap/pbs/, 2024. accessed: 2026-8-26.

[28] Ethereum Foundation. Maximal extractable value (MEV), August 2025. URL: http://ethereum.org/developers/docs/mev/.

[29] Ethereum Foundation. The Beacon Chain. https://ethereum.org/roadmap/beacon-chain/, 2025. accessed: 2026-8-26.

[30] Ethereum Foundation. Distributed validator technology. https://ethereum.org/staking/dvt/, 2026.

[31] Ethereum Foundation. Earn rewards while securing ethereum. https://ethereum.org/staking/, 2026.

[32] Ethereum Foundation. Ethereum protocol upgrade process. https://github.com/ethereum/pm/blob/master/processes/protocol-upgrade.md, 2026. Ethereum Project Management repository.

[33] Ethereum Foundation. Scaling ethereum through zero knowledge proofs. https://zkevm.ethereum.foundation/, 2026. accessed: 2026-8-26.

[34] Ethereum Foundation. Statelessness, state expiry and history expiry, June 2026. URL: https://ethereum.org/roadmap/statelessness/.

[35] Ethernodes. Ethereum Mainnet Statistics. https://www.ethernodes.org/, July 2026.

[36] ethPandaOps. Canonical Execution Block. https://ethpandaops.io/data/xatu/datasets/canonical_execution_/?table=canonical_execution_block, 2026.

[37] ethPandaOps. Ethereum package. https://github.com/ethpandaops/ethereum-package, 2026. accessed: 2026-05-06.

[38] Flashbots. Post mortem: April 3rd. URL: https://collective.flashbots.net/t/post-mortem-april-3rd-2023-mev-boost-relay-incident-and-related-timing-issue/1540.

[39] Flashbots. Introducing BuilderNet. https://buildernet.org/blog/introducing-buildernet, 2024.

[40] Flashbots. MEV-Boost Overview. https://docs.flashbots.net/flashbots-mev-boost/introduction, 2024.

[41] Flashbots. TEE-Boost. https://collective.flashbots.net/t/tee-boost/3741, August 2024.

[42] Flashbots. MEV-Boost Relay, October 2025. URL: https://github.com/flashbots/mev-boost-relay.

[43] Flashbots. Relayscan.io Bid Archive. https://bidarchive.relayscan.io/ethereum/mainnet/2026-06/index.html, 2026.

[44] Matthew K. Franklin and Michael K. Reiter. The design and implementation of a secure auction service. In *IEEE Symposium on Security and Privacy*, 1995.

[45] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold Encryption with Silent Setup. Cryptology ePrint Archive, Paper 2024/263, 2024. URL: https://eprint.iacr.org/2024/263.

[46] Pranav Garimidi, Joachim Neu, and Max Resnick. Multiple concurrent proposers: Why and how. Cryptology ePrint Archive, Paper 2025/1772, 2025. URL: https://eprint.iacr.org/2025/1772.

[47] Stefan Gast, Hannes Weissteiner, Robin Leander Schröder, and Daniel Gruss. Counterseveillance: Performance-counter attacks on AMD SEV-SNP. In *Network and Distributed System Security Symposium (NDSS)*, 2025.

[48] Noemi Glaeser, István András Seres, Michael Zhu, and Joseph Bonneau. Cicada: A framework for private non-interactive on-chain auctions and voting. Cryptology ePrint Archive, Paper 2023/1473, 2023. URL: https://eprint.iacr.org/2023/1473.

[49] Sophia Gold. Shipping an l1 zkevm #1: Realtime proving. https://blog.ethereum.org/2025/07/10/realtime-proving, 2025.

[50] Jack Grigg. rage: Rust implementation of age, October 2025. URL: https://github.com/str4d/rage.

[51] Hyperium. Tonic. https://github.com/hyperium/tonic, 2022.

[52] I. Abraham, K. Nayak, L. Ren, and Z. Xiang. Brief note: Fast authenticated byzantine consensus. *arXiv preprint arXiv:2102.07932*, 2021.

[53] Intel. Intel trust domain extensions whitepaper. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf, 2020. accessed: 2026-8-26.

[54] Mikhail Kalinin, Danny Ryan, and Vitalik Buterin. EIP-3675: Upgrade consensus to Proof-of-Stake, July 2021. Ethereum Improvement Proposals, no. 3675. URL: https://eips.ethereum.org/EIPS/eip-3675.

[55] David Kaplan, Jeremy Powell, and Tom Woller. AMD SEV-SNP: Strengthening vm isolationwith integrity protection and more. *White paper, Advanced Micro Devices Inc*, 2020.

[56] Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. Solana alpenglow consensus. https://drive.google.com/file/d/1Rlr3PdHsBmPahOInP6-Pl0bMzdayltdV.

[57] Georgios Konstantopoulos and Mike Neuder. Time, slots, and the ordering of events in Ethereum Proof-of-Stake. https://www.paradigm.xyz/2023/04/mev-boost-ethereum-consensus, April 2023.

[58] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia-Che Tsai, and Raluca Ada Popa. An off-chip attack on hardware enclaves via the memory bus. In *USENIX Security*, 2020.

[59] Lido. A year with simple DVT: Strengthening Ethereum staking through diversity and resilience. https://docs.lido.fi/run-on-lido/csm/node-setup/DVT-setup/obol/, 2025.

[60] Angela Lu. Searching in TDX. https://collective.flashbots.net/t/searching-in-tdx/3902, 2024.

[61] MaidSafe. BLST Threshold Crypto (blsttc), October 2025. URL: https://github.com/maidsafe.

[62] mikeneuder. Bid cancellations considered harmful. https://ethresear.ch/t/bid-cancellations-considered-harmful/15500, 2023.

[63] Andrija Novakovic, Alireza Kavousi, Kobi Gurkan, and Philipp Jovanovic. Cryptobazaar: Private sealed-bid auctions at scale. *IACR Cryptol. ePrint Arch.*, page 1410, 2024. URL: https://eprint.iacr.org/2024/1410.

[64] Obol Labs. Create a distributed validator with a group. https://docs.obol.org/docs/start/quickstart_group, 2026. Accessed: 2026-08-23.

[65] Burak Öz, Danning Sui, Thomas Thiery, and Florian Matthes. Who wins ethereum block building auctions and why? In *6th Conference on Advances in Financial Technologies (AFT 2024)*, volume 316 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.AFT.2024.22.

[66] Paradigm. Removing the Relays. https://www.paradigm.xyz/2024/10/removing-the-relays, October 2024.

[67] Paradigm. Reth, August 2025. URL: https://reth.rs/.

[68] Paradigm. stateless, August 2026. URL: https://github.com/paradigmxyz/stateless.

[69] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *Advances in Cryptology - EUROCRYPT, Workshop on the Theory and Application of of Cryptographic Techniques*, 1991.

[70] Macauley Peterson. Ethereum consensus shifts on EVM upgrade. https://blockworks.co/news/ethereum-consensus-evm-upgrade-fusaka-devs, 2025. accessed: 2026-8-26.

[71] Sigma Prime. Lighthouse: Ethereum consensus client. https://github.com/sigp/lighthouse, 2025. accessed: 2025-07-01.

[72] Filip Rezabek, Moe Mahhouk, Andrew Miller, Quintus Kilbourn, Georg Carle, and Jonathan Passerat-Palmbach. Proof of cloud: Data center execution assurance for confidential vms. *arXiv preprint arXiv:2510.12469*, 2025.

[73] Rust. Crate openssl. https://docs.rs/openssl/latest/openssl/, 2025. accessed: 2025-07-01.

[74] Rust. Crate sha2, October 2025. URL: https://docs.rs/sha2/latest/sha2.

[75] Rust. Crate tree_hash, October 2025. URL: https://docs.rs/tree_hash/latest/tree_hash.

[76] Benedict Schlüter, Supraja Sridhara, Andrin Bertschi, and Shweta Shinde. Wesee: Using malicious #vc interrupts to break AMD SEV-SNP. In *IEEE Symposium on Security and Privacy*, 2024.

[77] Benedict Schlüter, Christoph Wech, and Shweta Shinde. Heracles: Chosen plaintext attack on amd sev-snp. In *ACM Conference on Computer and Communications Security (CCS)*, 2025.

[78] Caspar Schwarz-Schilling, Fahad Saleh, Thomas Thiery, Jennifer Pan, Nihar Shah, and Barnabé Monnot. Time Is Money: Strategic Timing Games in Proof-Of-Stake Protocols. In *Conference on Advances in Financial Technologies, AFT*, 2023.

[79] Alexander Seto, Oytun Kuday Duran, Samy Amer, Jalen Chuang, Stephan van Schaik, Daniel Genkin, and Christina Garman. Wiretap: Breaking server SGX via DRAM bus interposition. In *ACM Conference on Computer and Communications Security (CCS)*, 2025.

[80] Amber Sprenkels. Shamir secret sharing in Rust, October 2025. URL: https://github.com/dsprenkels/sss-rs.

[81] SSV Network. Cluster creation. https://docs.ssv.network/learn/network-overview/clusters/cluster-creation/, 2026. Accessed: 2026-08-23.

[82] Alex Stokes, Ansgar Dietrichs, Danny Ryan, Martin Holst Swende, and lightclient. EIP-4788: Beacon block root in the EVM, Febrary 2022. Ethereum Improvement Proposals, no. 4788. URL: https://eips.ethereum.org/EIPS/eip-4788.

[83] Succinct. SP1 Hypercube Achieves Real Time Proving with 16 GPUs. https://blog.succinct.xyz/real-time-proving-16-gpus/, 2025.

[84] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, 2017.

[85] terence. Builder Bidding Behaviors in ePBS. https://ethresear.ch/t/builder-bidding-behaviors-in-epbs/20129, July 2024.

[86] The Linux Kernel. dm-integrity. https://docs.kernel.org/admin-guide/device-mapper/dm-integrity.html, 2026. accessed: 2026-8-26.

[87] Nirvan Tyagi, Arasu Arun, Cody Freitag, Riad S. Wahby, Joseph Bonneau, and David Mazières. Riggs: Decentralized sealed-bid auctions. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM Conference on Computer and Communications Security (CCS)*, 2023.

[88] Ultra Sound. Ultra Sound Relay. URL: https://relay.ultrasound.money/.

[89] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice. https://sgaxe.com/, 2020.

[90] Bharath Vedartham. Proposer Relay Interactions with ePBS. https://hackmd.io/@bchain/BJYoUYjIll, 2025. accessed: 2026-8-26.

[91] Toni Wahrstätter. MEV-Boost Dashboard. https://mevboost.pics/, 2025. accessed: 2026-8-26.

[92] Fei Wu, Thomas Thiery, Stefanos Leonardos, and Carmine Ventre. Strategic bidding wars in on-chain auctions. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2024.

[93] Sen Yang, Kartik Nayak, and Fan Zhang. Decentralization of Ethereum's Builder Market. In *IEEE Symposium on Security and Privacy*, 2025.

[94] Luca Zanolini. Why Ethereum Needs a Dynamically Available Protocol. https://ethresear.ch/t/why-ethereum-needs-a-dynamically-available-protocol/24418, Mar 2026.

[95] Ke Zhong, Yiping Ma, Yifeng Mao, and Sebastian Angel. Addax: A fast, private, and accountable ad exchange infrastructure. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

A Detailed sub-protocol and algorithm for Π_{DPaaS}

The AppCompress and AppInterp is shown in Algorithm 2, which are the functions instantiated by DPaaS semantic for the input Compress and Interp to $\Pi_{\mathsf{Crab\text{-}BB}}$. AppCompress summarizes the $\mathcal{V}_i'$ from $\mathcal{E}_i$ by extracting the most recent bid for each builder (Line 1,2). AppInterp takes the input Val then outputs a set of bids. If Val is a strong quorum (SQ), just extract all the summarization $\mathcal{V}_i'$ into a set $\mathcal{C}$ (Line 4,5); otherwise, if Val is set of a weak quorum (WQs), it extracts all summarization $\mathcal{V}_i'$ from unique PEs into a set $\mathcal{C}$ (Line 6-11). For every builder, a candidate bid multi-set will be formed (Line 14) by collecting all bids appeared explicitly in the summarizations $\mathcal{V}_i'$ of $\mathcal{C}$. Then based on the multi-set just formed, the function selects each builder's $(2f+1)$-th bid in decreasing order of sequence number (i.e., bid with the largest sequence number supported by at least $2f+1$ PEs) and aggregates those bids into the global active set–$\mathcal{A}$ (Line 15). We will show in Appendix C, this preserves the definition of implicit membership.

The protocol for Reconstruction and Interpolation phase is shown in Fig. 9. Each PE_i starts by decrypting its own share using its private key and broadcasts the share to all PEs (Step 1). When receiving a share, PE_i checks the validity of the share by comparing the hash of the K_j it receives the hash contained in $w.\texttt{spl}$ (Step 2). Once PE_i receives valid shares from at least $f+1$ PEs, it can reconstruct the symmetric key K and decrypt the encrypted transactions contained in $w.\texttt{spl}$. At last PE_i is able to recover the full payload (Step 3).

In Ethereum, the final signature of the beacon block should be over the root hash of all fields of the beacon block, including the payload `pl` and `cdata`. Instead of waiting to reconstruct the full payload with transactions in plaintext from $w.\texttt{spl}$ to compute $\mathtt{R}(\mathtt{pl})$, we instead optimize this by including $\mathtt{R}(\mathtt{pl})$ in plaintext as part of $w.\texttt{spl}$. After winner finalization, the root hash r of whole winning block can be directly computed given the root hash of the two necessary fields: consensus data and payload (Step 4). Therefore, we can execute the Reconstruction (Step 1-3) and Interpolation (Step 4-7) in parallel given the independency of the two procedure, which reduces the latency from end-auction to proposal.

For signature interpolation, each PE_i generates its partial signature σ_i and broadcasts this to all other PEs (Step 5). Meanwhile, when receiving any signatures from other PEs, the PE_i also verifies it against their public BLS partial public key generated during offline (Step 6). After receiving $f+1$ valid partial signatures, PE_i uses Lagrange interpolation to compute the full signature that can be verified by the aggregate public key pk_V (Step 7).

Input: winning bid w, consensus data `cdata`,
Output: `SignedBeaconBlock`

1. Decrypt the local payload-key share
$$K_i \leftarrow \mathcal{D}_{sk^i_{\text{asym}}}\big(w.\texttt{spl.esshr}[i]\big),$$
broadcast $\langle \texttt{share}, K_i\rangle_i$, and insert K_i into S.
2. Upon receiving $\langle \texttt{share}, K'_j\rangle_j$ from PE_j, insert K'_j into S only if $H(K'_j) = w.\texttt{spl.esshr}[j].h$.
3. Once $|S| \geq f+1$, recover the symmetric key and decrypt the winning block's transactions:
$$K \leftarrow \textsc{RecoverKey}(S).$$
$$\texttt{pl} \leftarrow \{w.\texttt{spl.header} \mid \mathcal{D}_K(w.\texttt{spl.E}_K(\texttt{txs}))\}.$$
4. Compute the signing root using the consensus data and the payload root committed in the sealed payload:
$$r \leftarrow \mathcal{R}\big(\mathcal{R}(\texttt{cdata}), w.\texttt{spl}.\mathcal{R}(\texttt{pl})\big).$$
5. Produce a partial BLS signature $\sigma_i \leftarrow \textsc{Sign}_{sk^i_{\text{p_sign}}}(r)$. Then broadcast $\langle \texttt{psig}, \sigma_i\rangle_i$ and insert σ_i into I.
6. Upon receiving $\langle \texttt{psig}, \sigma_j\rangle_j$ from PE_j, insert σ_j into I only if
$$\textsc{Verify}_{pk^j_{\text{p_sign}}}(\sigma_j, r) = \texttt{true}.$$
7. Once $|I| \geq f+1$, interpolate the final BLS signature $\sigma \leftarrow \textsc{AggregateSig}(I)$, and output $\texttt{SignedBeaconBlock} = (\texttt{pl}, \sigma)$.

Figure 9: Protocol Π_{rec} for reconstructing the winning payload and producing the threshold BLS signature.

B Detailed Proof of Properties for $\Pi_{\mathsf{Crab\text{-}BB}}$

First, we introduce the correctness of Π_{vbb}, which can be instantiated by $(5f-1)$-psync-VBB protocol as four lemmas, which have been proved [2, 52].

Lemma 1. ***Agreement****: If an honest replica commits* Val*, no honest replica commits any* $\text{Val}' \neq \text{Val}$ *in* Π_{vbb}.

Lemma 2. ***Termination****: Every replica eventually commits and terminates in* Π_{vbb}.

Lemma 3. ***Validity****: In* Π_{vbb}*, if the designated broadcaster replica is honest and* $\text{GST} = 0$*, then all honest replicas commit the broadcaster's value; otherwise the value* v *committed by any honest replica satisfies* $\mathcal{F}(v) = \texttt{true}$ *(*v *is externally valid).*

Lemma 4. ***Good-case latency****: When* $\text{GST} = 0$ *and the leader is honest, the proposal of the leader will be committed within 2 rounds in* Π_{vbb}.

Then we can prove that the $\Pi_{\mathsf{Crab\text{-}BB}}$ solves the Crab-BB problem as defined in Definition 1 given all the lemmas

Algorithm 2 DPaaS Compress and Interp Functions

function AppCompress($\mathcal{E}_i$)
  return $\mathcal{V}_i = \bigcup_{\mathtt{b}\in\mathcal{B}} \arg\max_{e\in\mathcal{E}_i[\mathtt{b}]}(e.\mathtt{seq})$
function AppInterp(Val)
  if Val is SQ **then**
    $\mathcal{C} \leftarrow \{\text{Val}\}$
  else
    $\mathcal{C} \leftarrow \emptyset$
    procPE $\leftarrow n * [\mathtt{false}]$
    for $WQ_i \in$ Val, $\mathcal{V}_j \in WQ_i$ **do** ▷ *i ordered deterministicly*
      if procPE$[j] == \mathtt{false}$ **then**
        $\mathcal{C} = \mathcal{C} \cup \{\mathcal{V}_j\}$; procPE$[j] = \mathtt{true}$
  $\mathcal{A} \leftarrow \emptyset$
  for all $\mathtt{b} \in \mathcal{B}$ **do**
    cand[b] $\leftarrow [\mathcal{V}_i[\mathtt{b}] \mid \mathcal{V}_i \in \mathcal{C}, \mathtt{b} \in \mathrm{dom}(\mathcal{V}_i)]$ ▷ *multi-set*
    $\mathcal{A}[b] \leftarrow \mathrm{ord}^{\downarrow}_{2f+1}(\text{cand[b]}; e \mapsto e.\mathtt{seq})$
  return $\mathcal{A}$

Algorithm 3 External validity used by Π_{vbb}

Input: Val. **Output:** `True` or `False`
function $\mathcal{F}$(Val)
  if Val $= \perp$ **then**
    return `False`
  if all relayed message signatures in Val verified **then**
    if $n = 4$ **then**
      if Val contains either one SQ or $f+1$ WQ **then**
        return `True`
    else if $n > 4$ **then**
      if Val contains one WQ **then**
        return `True`
  return `False`

guarantees the correctness of Π_{vbb}, . We start from basic Agreement and Termination (Theorem 1), then Non-triviality (Theorem 2), at last censorship-resistance and availability-backed (Theorem 3).

Theorem 1. *If Π_{vbb} achieves Agreement and Termination, $\Pi_{\text{Crab-BB}}$ (Fig. 3) achieves Agreement and Termination as well: if one honest replica outputs W, no other honest replicas decide on output as $W' \neq W$; and all honest replicas eventually decide a value.*

Proof. For Agreement, by Lemma 1, honest replicas do not disagree on the output of evidence value Val from Π_{vbb}. After applying the same deterministic function Interp, honest replicas do not disagree on the output W. For Termination, after GST, Steps 2 and 3 in Fig. 3 can be completed by 2 rounds at most, which requires 2Δ time after which all honest replicas enter Π_{vbb}. By Lemma 2, all honest replicas decide and terminate. □

Theorem 2. *Assume Π_{vbb} is correct. Then $\Pi_{\text{Crab-BB}}$ satisfies Non-triviality against any f-bounded adversary when $n \geq 5f-1$ with external validity function $\mathcal{F}$ defined as in Algorithm 3.*

Proof. Observe that Π_{vbb} requires parties to input externally valid values as defined in Algorithm 3. Moreover, Steps 2 and 3 ensure that honest parties enter Π_{vbb} only after receiving either one SQ or $f+1$ WQ. Any Val outside of this scope will result in $\mathcal{F}(\text{Val}) = \mathtt{false}$ and never become committed by Π_{vbb}.

□

Theorem 3. *Assume Π_{vbb} is correct. Then $\Pi_{\text{Crab-BB}}$ with deterministic output interpretation satisfies censorship-resistance and availability-backed against any f-bounded adversary when $n = 5f-1$.*

Proof. We first consider censorship resistance. Assume GST $= 0$, and let u be an input broadcast by client c_j. Suppose that every honest replica P_i includes u in its local input set, i.e., $u \in E_i$. Then, by Theorem 2, the output must have one of two forms: either it is certified by a SQ, or it is certified by $f+1$ WQ values.

In case (i), inside Val $= SQ$, the number of distinct input set $|SQ| \geq n-f+1 = 4f$. Among the SQ, there should exist $4f-f = 3f$ honest replicas' input set E_i. That implies, if an input u is made to all honest replicas, it must appear in at least $3f \geq 2f+1$ sets in SQ, by the rule in Step 5 of Fig. 3, the client input $u \overset{\bullet}{\in} W$.

In case (ii), after GST, within Δ, all honest replicas' `ex-1` messages can be delivered. Therefore all honest replica's should include at least $n-f$ distinct validly signed `ex-1` messages to their either `ex-fast-2` or `ex-normal-2` messages to be broadcast at $T = \Delta$. When the output is Val $= WQ$, that implies the leader driving commit proposes a WQ which contain $f+1$ distinct validly signed `ex-normal-2` messages. Among these `ex-normal-2` messages, there must be one `ex-normal-2` from an honest replica, which must include all honest replica's `ex-1` messages carrying their input set. Then the argument follows case (i), if a client input u appear in all honest replicas' input set, it must exist in every honest replica's `ex-1` messages. By $n-f \geq 2f+1$ and the rule in Step 5 of Fig. 3, the client input $u \overset{\bullet}{\in} W$.

Then we focus on availability-backed. Once a client input u appears in the output (*i.e.*, $u \overset{\bullet}{\in} W$), by the rule in Step 5 of Fig. 3, $u \in \mathsf{Supp}_{2f+1}(\text{Val})$ which implies there must exist at least $2f+1$ replicas' compressed input set implicitly containing such input u, among which at most f replicas are Byzantine. Therefore, at least $f+1$ honest replicas can back such input implicitly containing such input u to exchange and explicitly backing such input u, the claim is proved. □

C Detailed Proof of Properties in application-level for Π_{DPaaS}

In this section, we provide additional details on how DPaaS satisfies the security goals in Definition 3.

We first consider *Block Validity* (**G2**, Def. 3.2). For a winning block to be valid, it must satisfy two conditions: (1) it contains the promised payment to the proposer, and (2) it is valid with respect to the canonical Ethereum state. Because we trust the integrity of the BT, it verifies both conditions before accepting and submitting a bid.

Consider first a stateful BT, which validates a block against its locally stored pre-state. An adversarial host may attempt to either tamper with this state or prevent the BT from synchronizing to the latest Ethereum state. The former violates the integrity protection of the BT's storage and is therefore detected. In the latter case, the BT may successfully validate a block against a stale pre-state; however, the corresponding bid will be rejected by honest PEs, whose Ethereum nodes remain synchronized with the canonical chain and verify the parent state associated with the bid. Consequently, a malformed block or a block validated against a stale state cannot enter the globally eligible set and therefore cannot win the auction.

The argument is simpler in stateless mode. Because the BT does not maintain persistent Ethereum state, an adversarial host cannot corrupt a locally stored pre-state. Instead, the BT validates the block using the authenticated state information supplied with the block, while honest PEs still ensure that the referenced parent state corresponds to the canonical chain. Thus, the same Block Validity guarantee holds without relying on persistent storage integrity.

For the remaining security goals, we organize the analysis around four components:(1) the f-bounded adversary among the heterogeneous TEEs; (2) the threshold BLS-signature and threshold secret share scheme are correct; (3) the correctness of $\Pi_{\mathsf{Crab\text{-}BB}}$; (4) the input compression and output interpretation accurately falls into the implicit inclusion. As (1) is the basic assumption justified in our threat model Sec. 3.2, (2) has been proved in the cryptographic literature, (3) has been proved in Appendix B, we prove mainly (4) to conclude the correctness of Π_{DPaaS}.

Theorem 4. *Failed Trade Privacy (**G4**, Def. 3.5) always holds in the protocol Π_{DPaaS}.*

Proof. According to Theorem 1, at least $n-f$ honest PEs will decide on the same winning bid w and only share the winning block's secret share. For all other bids, no honest PE will release their secret share; moreover, since the reconstruction threshold is $f+1$, the $\leq f$ faulty PEs cannot recover these bids. □

Remark: Theorem 4 guarantees that all honest replicas will help recover the only winning bid; the losing bids cannot be recovered. In contrast, any centralized escrow like MEV-Boost relay does not have such a guarantee.

For rest of properties, we need to first define the semantic of bidding in the PBS setting and then prove the input/output wrapper in Π_{DPaaS} (*i.e.*, Algorithm 2) is correct.

Definition 4. *Bid Order: For each builder $\mathtt{b} \in \mathcal{B}$, let $E_i[\mathtt{b}]$ denote the set of bids from $\mathtt{b}$ contained in replica PE_i's local input set E_i. For two bids $e, e' \in E_i[\mathtt{b}]$, we define the builder-local order*

$$e \preceq_{\mathtt{b}} e' \quad \Longleftrightarrow \quad e.\mathtt{seq} \leq e'.\mathtt{seq}.$$

Among bids from the same builder, the bid with the larger sequence number is considered more recent.

Definition 5. *DPaaS representation relation: Let V be an active-bid summary. For a bid e, we write*

$$e \overset{\bullet}{\in} V$$

if and only if V contains a bid $\hat{e}$ from the same builder such that $e \preceq_{\mathtt{b}} \hat{e}$:

$$e \overset{\bullet}{\in} V \quad \Longleftrightarrow \quad \exists \hat{e} \in V : \hat{e}.\mathtt{b} = e.\mathtt{b} \wedge e \preceq_{\mathtt{b}} \hat{e}.$$

Thus, an active-bid summary explicitly represents the bids it stores and implicitly represents all earlier bids from the same builder.

Lemma 5. *Compression soundness: For every local eligible bid set $\mathcal{E}_i$, let*

$$\mathcal{V}_i = \mathsf{AppCompress}(\mathcal{E}_i).$$

Then for every bid e,

$$e \overset{\bullet}{\in} \mathcal{V}_i \quad \Longleftrightarrow \quad e \in \mathcal{E}_i.$$

Proof. Fix a builder $\mathtt{b}$. If $\mathcal{E}_i[\mathtt{b}] = \emptyset$, then $\mathsf{AppCompress}(\mathcal{E}_i)$ contains no bid from $\mathtt{b}$. Therefore, by Definition 5, no bid from $\mathtt{b}$ is represented by $\mathcal{V}_i$. Hence, for every bid e from $\mathtt{b}$, both $e \overset{\bullet}{\in} \mathcal{V}_i$ and $e \in \mathcal{E}_i$ are false.

Otherwise, let

$$e^\star = \underset{e \in \mathcal{E}_i[\mathtt{b}]}{\arg\max}\, e.\mathtt{seq}.$$

By definition of $\mathsf{AppCompress}$, the summary $\mathcal{V}_i$ contains exactly $e^\star$ for builder $\mathtt{b}$.

We first show the forward direction. Suppose $e \overset{\bullet}{\in} \mathcal{V}_i$. By Definition 5, there exists $\hat{e} \in \mathcal{V}_i$ such that $\hat{e}.\mathtt{b} = e.\mathtt{b}$ and $e \preceq_{\mathtt{b}} \hat{e}$. Since $\mathcal{V}_i$ contains only $e^\star$ for builder $\mathtt{b}$, we have $\hat{e} = e^\star$. Hence $e \preceq_{\mathtt{b}} e^\star$. By the definition of local eligibility, eligible bids from the same builder have contiguous sequence numbers up to the locally active bid. Therefore, every bid preceding $e^\star$ is also in $\mathcal{E}_i$, and so $e \in \mathcal{E}_i$.

We now show the reverse direction. Suppose $e \in \mathcal{E}_i[\mathsf{b}]$. Since $e^\star$ is the maximal-sequence bid in $\mathcal{E}_i[\mathsf{b}]$, we have $e \preceq_{\mathsf{b}} e^\star$. Since $e^\star \in \mathcal{V}_i$, by Definition 5, $e \overset{\bullet}{\in} \mathcal{V}_i$.

The argument holds independently for every builder, so for every bid e:

$$e \overset{\bullet}{\in} \mathcal{V}_i \quad \Longleftrightarrow \quad e \in \mathcal{E}_i.$$

□

Definition 6. *Threshold-supported eligible set: For a committed FBB value* Val*, let* $\mathcal{C}(\mathrm{Val})$ *be the collection of summaries extracted from* Val*. We define*

$$\mathsf{Supp}_{2f+1}(\mathrm{Val}) = \left\{ e \;\middle|\; \left| \{ \mathcal{V}_j \in \mathcal{C}(\mathrm{Val}) : e \overset{\bullet}{\in} \mathcal{V}_j \} \right| \geq 2f+1 \right\}.$$

as the threshold-supported eligible set.

Lemma 6. *Interpretation soundness: For every committed FBB evidence value* Val*,*

$$\mathsf{AppInterp}(\mathrm{Val}) = \mathsf{AppCompress}(S(\mathrm{Val})).$$

Equivalently, $\mathsf{AppInterp}(\mathrm{Val})$ *is the canonical active-bid summary of all bids represented by at least* $2f+1$ *summaries in* $\mathcal{C}(\mathrm{Val})$*.*

Proof. Let $\mathcal{A} = \mathsf{AppInterp}(\mathrm{Val})$ be the output of Algorithm 2. We prove that $\mathcal{A} = \mathsf{AppCompress}(S(\mathrm{Val}))$ builder by builder.

Fix a builder b. Let T_{b} be the multiset of active bids from b appearing in the summaries of $\mathcal{C}(\mathrm{Val})$. Order T_{b} in non-increasing sequence number:

$$e_1, e_2, \ldots \quad \text{where} \quad e_1.\mathsf{seq} \geq e_2.\mathsf{seq} \geq \cdots.$$

If $|T_{\mathsf{b}}| < 2f+1$, then no bid from b can be represented in at least $2f+1$ summaries. Hence $S(\mathrm{Val})[\mathsf{b}] = \emptyset$, and both $\mathsf{AppCompress}(S(\mathrm{Val}))$ and Algorithm 2 output no active bid for b.

Otherwise, let $e^\star = e_{2f+1}$ be the $(2f+1)$-th bid in this order. By Definition 5, a bid e from builder b is represented by a summary containing active bid $\hat{e}$ exactly when $e \preceq_{\mathsf{b}} \hat{e}$. Therefore, e is represented by at least $2f+1$ summaries if and only if at least $2f+1$ bids in T_{b} have sequence number at least $e.\mathsf{seq}$.

It follows that the maximum-sequence bid in $S(\mathrm{Val})[\mathsf{b}]$ is exactly $e^\star$. Indeed, $e^\star$ is represented by the first $2f+1$ summaries in the order, so $e^\star \in S(\mathrm{Val})$. Any bid e with $e^\star \prec_{\mathsf{b}} e$ has sequence number larger than $e^\star$, and is therefore represented by fewer than $2f+1$ summaries; hence $e \notin S(\mathrm{Val})$.

Thus, for builder b, $\mathsf{AppCompress}(S(\mathrm{Val}))$ keeps exactly $e^\star$. This is also exactly the bid selected by Algorithm 2. Since the argument holds for every builder independently, we conclude

$$\mathsf{AppInterp}(\mathrm{Val}) = \mathsf{AppCompress}(S(\mathrm{Val})).$$

□

Lemma 7. Π_{DPaaS} *satisfies Censorship-Resistance: for any builder* b*, if its bid* e *satisfies, for all honest* PE_i*,* $e \in \mathcal{E}_i$*, after executing* $\Pi_{\textit{Crab-BB}}$ *in* Π_{DPaaS} *with compressed input and output interpretation defined in Algorithm 2,* e *should satisfy* $e \overset{\bullet}{\in} \mathcal{A}$*.*

Proof. The soundness Lemma 5 and Lemma 6 shows Π_{DPaaS} is in fact an instantiation of $\Pi_{\mathsf{Crab\text{-}BB}}$. By Theorem 3 showing that $\Pi_{\mathsf{Crab\text{-}BB}}$ is censorship-resistant, we know the Π_{DPaaS} is also censorship-resistant. □

Lemma 8. Π_{DPaaS} *satisfies Backed-Availability: for any builder* b*, if its bid* $e \overset{\bullet}{\in} \mathcal{A}$*, then there must exist* $f+1$ *honest* PE_i *backing* $e \in \mathcal{E}_i$*.*

Proof. The soundness shows Π_{DPaaS} is in fact an instantiation of $\Pi_{\mathsf{Crab\text{-}BB}}$. By Theorem 3 showing that $\Pi_{\mathsf{Crab\text{-}BB}}$ satisfies backed-availability, we know the Π_{DPaaS} satisfies backed-availability. □

Theorem 5. *If* $\mathrm{GST} < T_e - \Delta_{b2p} - \Delta_{pr}$*, Cancellation Effectiveness (**G6**, Def. 3.6) always holds in the protocol* Π_{DPaaS}*.*

Proof. Fix any builder b. Let e be the most recent bid from b observed after GST and before $T_e - \Delta_{b2p} - \Delta_{pr}$. By the post-GST latency bound Δ_{b2p} and processing bound Δ_{pr}, bid e is delivered to and processed by every honest PE before T_e. Hence $e \in \mathcal{E}_i$ for every honest PE_i. By Censorship-Resistance of Π_{DPaaS} (Lemma 7), we have $e \overset{\bullet}{\in} \mathcal{A}$. By the DPaaS representation relation, this implies that $\mathcal{A}$ explicitly contains some bid $\hat{e}$ from the same builder with $e \preceq_{\mathsf{b}} \hat{e}$. Therefore, for every earlier bid $e' \in \mathcal{E}[\mathsf{b}]$ such that $e' \prec_{\mathsf{b}} e$, we have $e' \prec_{\mathsf{b}} \hat{e}$.

Winner finalization selects only explicit active bids in $\mathcal{A}$. Therefore e' cannot be selected as the winner. Since this holds for every earlier bid e' and every builder b, Bid Cancellation Effectiveness holds. □

Remark on Theorem 5. Consequently, the Bid Cancellation Effectiveness guarantees a builder that after GST, those builders who submit their bids to all PEs before $t - \Delta_{b2p} - \Delta_{pr}$, can always ensure that PEs will only consider their most recent bid and not the stale one that should be canceled. Such a guarantee is missing with existing relays since cancellation is best-effort and thus a relay could effectively censor bids. However, if the builder still submits new bids after $t - \Delta_{b2p} - \Delta_{pr}$ or before GST, the cancelation may only happen on a best-effort basis.

Theorem 6. *After* GST*, protocol* Π_{DPaaS} *guarantees Bid Selection (**G1**, Def. 3.1).*

Proof. Let $\mathcal{A}$ be the globally active bid summary output by Π_{DPaaS}, and let w be the bid selected by the winner-finalization rule.

First, we prove selection of the highest globally eligible bid. Let e be a bid such that, for every honest replica PE_i, we

have $e \in \mathcal{E}_i$. By Censorship-Resistance of Π_{DPaaS} (Lemma 7), after executing $\Pi_{\mathsf{Crab\text{-}BB}}$ with DPaaS compression and interpretation functions, bid e is represented in the final active summary:

$$e \overset{\bullet}{\in} \mathcal{A}.$$

By the DPaaS representation relation, this means that $\mathcal{A}$ contains an explicit active bid $\hat{e}$ from the same builder such that

$$e \preceq_{\mathsf{b}} \hat{e}.$$

Thus, the builder of e has an active representative in $\mathcal{A}$, and any earlier bid from the same builder is considered canceled.

Now suppose that e has the highest bid value among all globally active bids at the end of the auction. By interpretation soundness, $\mathcal{A}$ is exactly the canonical active summary produced from the protocol-induced globally eligible set. Therefore, the winner-finalization rule selects the explicit bid with the largest bid value in $\mathcal{A}$:

$$w = \arg\max_{a \in \mathcal{A}} a.\mathsf{bv}.$$

Since e is represented by the active bid for its builder and has the highest value among globally active bids, the selected winner is exactly that highest globally active bid.

Second, we prove backed availability of the winner. Since w is selected from $\mathcal{A}$, we have

$$w \overset{\bullet}{\in} \mathcal{A}.$$

By Backed-Availability of Π_{DPaaS} (Lemma 8), there exists at least $f+1$ honest replicas PE_i such that

$$w \in \mathcal{E}_i.$$

Therefore, the winning bid is backed by at least $f+1$ honest PEs. Since honest PEs retain the corresponding bid data needed for reconstruction, the winner can be reconstructed.

Hence, after GST, Π_{DPaaS} both selects the highest globally active bid and ensures that any selected winner is backed by enough honest replicas to be reconstructed. □

Remark on Theorem 6. The guarantee ensures two aspects. First, it grants the builder clear guarantee that if it reaches all honest PEs and the builder did not cancel it, it must be considered as a candidate for winner. Second, it grants the proposer that the winner's block data must be backed by at least $f+1$ honest PEs. This theorem directly implies the correctness of the DPaaS protocol.

Theorem 7. *Block Availability (**G2**, Def. 3.3) always holds in* Π_{DPaaS}.

Proof. By Theorem 1, $\Pi_{\mathsf{Crab\text{-}BB}}$ ensures all PEs terminate eventually. By Theorem 6, if any bid w is selected as the winner, w should be within the globally active bid set and must be backed by $f+1$ honest PEs. As the threshold for both secret share and BLS signature is $f+1$ in Π_{rec}, the $f+1$ honest PEs will output the Signed Beacon Block. □

Theorem 8. *If* Π_{vbb} *can achieve 2-rounds good-case latency,* Π_{DPaaS} *can achieve the good-case latency of* 5 *rounds to determine the winner, and can achieve the optimistic (all* PE*s are honest) good-case latency of* 4 *rounds. (**G7**)*

Proof. Since DPaaS uses $\Pi_{\mathsf{Crab\text{-}BB}}$ as a primitive $\mathsf{T}_{gc}(\Pi_{\mathsf{DPaaS}}) = \mathsf{T}_{gc}(\Pi_{\mathsf{Crab\text{-}BB}})$. We instantiated Π_{vbb} by $(5f-1)$-psync-VBB which can achieve 2-rounds good-case latency by Lemma 4, so $\mathsf{T}_{gc}(\Pi_{vbb}) = 2$. Considering the two rounds of exchange $\tau_{ex} = 2$ added in $\Pi_{\mathsf{Crab\text{-}BB}}$, at most 4 rounds are needed to commit the final winner. In total, including one round for bid submission as in previous work [3], the good-case latency is $\mathsf{T}_{gc}(\Pi_{\mathsf{DPaaS}}) = 5$.

Under optimistic good-case, if a valid message `ex-fast-2` exists in the network, every honest PE can get prepared with one signed `ex-fast-2` message. This only takes 1 round ($\tau_{ex} = 1$) for all honest PEs to complete then 2 rounds after entering Π_{vbb}. Therefore, the optimistic good-case latency is $\mathsf{T}_{gc}(\Pi_{\mathsf{DPaaS}}) = 4$. □

Remark. This result matches the lower bound shown in [3] when the Byzantine fraction is larger than $\frac{1}{5}$ (when $n = 4$), but has one extra round for good-case latency when n is larger. We leave more discussion to Appendix E.

Theorem 9. *Timely Block Availability (**G3**, Def. 3.4) holds after* GST *before* T_p.

It's easy to show Theorem 9 holds given Theorem 7 after GST.

Remark. Besides the theorem, it is worth exploring how to set such T_p, because setting T_p too conservative ($T_p = 4$s) may incur a missed slot; setting T_p too aggressive may end the auction too early thus missing MEV opportunity.

By Theorem 8, we know that the good-case post-auction latency can be $5-1=4$ rounds plus reconstruction latency and optimistic good-case post-auction latency can be $4-1=3$ rounds plus reconstruction latency by excluding the bidding phase.

In both cases, Δ for the first round of exchange is necessary. In optimistic good-case, all honest PEs can directly enter Π_{vbb} without waiting for one more round. In other good-case, δ is added to collect the $f+1$ $\mathcal{V}$. Due to the responsiveness of Π_{vbb}, Π_{vbb} takes only 2δ to finish in the good case. At reconstruction, as reconstruction and interpolation can be parallelized to one round, we denote it as δ_{rec}. To sum them up, the optimistic good-case (best case) post-auction will cost $\Delta + 2\delta + \delta_{rec}$ and normal good-case will cost $\Delta + 3\delta + \delta_{rec}$.

But if the first leader entering Π_{vbb} is a faulty leader, Π_{vbb} cannot hit good-case if the faulty leader is silent or proposes some invalid proposal. Thus, Π_{vbb} may not terminate until the first honest leader appears. If the leaders are rotated round-robin, and the Π_{vbb}'s view-length is 4Δ (as in $(5f-1)$-psync-VBB), this may take $4f\Delta + 2\delta$ to terminate. Therefore, the worst-case post-auction will cost $\Delta + 4f\Delta + 3\delta + \delta_{rec}$. When

$n = 4, f = 1$, this actually gives us the lowest worst-case latency $5\Delta + 3\delta + \delta_{rec}$.

D Stateless validation

D.1 Comparison with stateful mode.

Authenticated pre-state in stateless mode. The BT's inputs are the block and an execution witness supplied by the builder's untrusted node: the account- and storage-trie node preimages the block touches, the bytecode it invokes, and a contiguous range of ancestor headers. The BT walks the headers back from the candidate block, adopts the immediate parent's `stateRoot` as the expected pre-state root, and reveals into a sparse trie only nodes reachable by hash from that root, requiring the recomputed root to match. Untouched subtrees remain opaque hashes, which is what lets the sparse trie commit to the same root as the full one. After execution, the BT applies the state changes, recomputes the post-state root, and requires it to match the block header along with the other execution commitments.

Safety. Two properties ensures stateless validation's safety. First, the construction fails closed: every value the EVM reads is either proved to lie under the parent `stateRoot` or proved absent, and a path entering an omitted node errors rather than returning a default. A malicious builder cannot turn an omitted nonzero value into zero; it can only cause its own bid to be rejected. Second, the BT does not trust any claimed execution result—it re-executes and checks the post-state root itself. Together: given a parent root, collision resistance of Keccak-256, and a correct BT, a block the stateless BT accepts reads the same pre-state values and yields the same post-state root that full-state validation would compute. Even though the untrusted builder could send a stale witness from previous slots and pass the stateless validation, it can't pass the check of the majority of PEs on the parent root as we mentioned in Sec. 5, thus no impact on the Block Validity if the threat model holds.

Trade-offs. We conclude the trade-offs in Table 2. Stateless validation removes the state database, the storage stack, and the integrity layer from the trust boundary, replacing them with trie verification built into BT, and makes correctness independent of how the host stores or versions data. It also strengthens what a bid attests, since the BT checks the post-state root rather than only that execution succeeds. The latency cost moves onto the builder's bidding path: witness generation can cost extra 0.27 seconds on average even if the builder enables prefetch, and the witness is substantially larger than the block itself (28.7 ×). The stateless BT's overhead is also 1.85 × as the stateful BT.

	Stateful	**Stateless**
Pre-state source	Full state DB	Witness
Trust	BT +storage integrity	BT integrity
Persistent state	Required, synced	None
Data size	159.7 KB	4.47 MB
Builder overhead	None	0.62 s
BT overhead	84 ms	156 ms
Economic cost	0.36 + 0.12 USD/h	0.36 USD/h

Table 2: Comparison of BT validation modes.

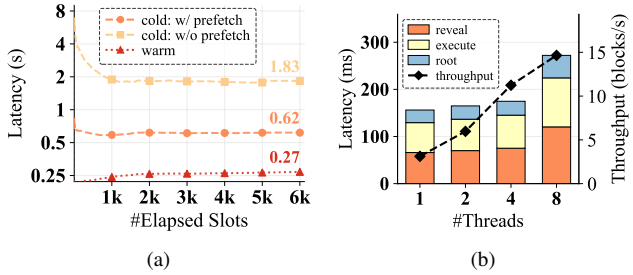

Figure 10: Evaluation of stateless validation on the canonical block dataset. (a) Witness generation latency for builders. (b) Witness verification break-down latency and throughput for BT.

D.2 Detailed evaluation results.

Cold-start latency merits closer attention, since this overhead falls on the builder. Figures based the same experiment as Sec. 8.2 shows more detail of stateless mode.

Fig. 10a reports latency over 6k elapsed slots under cold-start and warmed-up settings. With prefetching, cold-start latency stabilizes at around 0.62 s, compared with 1.83 s without prefetching, while warmed-up validation remains around 0.27 s. All of them reaches stable state before 1k slots where their in-memory cache is almost filled. Fig. 10b breaks witness verification into reveal (the pre-state), execution, and state-root computation, and shows that increasing the number of threads improves throughput from roughly 3 to 15 blocks/s. This parallelism increases per-block latency, however, from about 155 ms with one thread to 270 ms with eight threads.

Moreover, we instrumented major page faults to isolate the bottleneck (Fig. 11a). Fault counts track the cold-start latency curve of Fig. 11a across all 6,000 slots, and prefetching trie nodes into memory ahead of access cuts steady-state faults from 309 to 59 per validation, an 81% reduction; warm runs incur none. Cold-start cost is therefore I/O-bound rather than compute-bound, and the builder's lever is to populate the in-memory cache while it constructs the block, off the bidding path.

Witness verification, by contrast, is CPU-bound and scales with the number of trie nodes the block touches. Fig. 11b shows this relationship is linear at 16.0μs per node (R^2 =

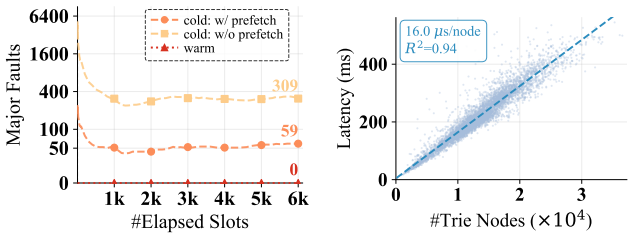

Figure 11: Evaluation of stateless validation on the canonical block dataset. (a) Number of major faults for builders when generating witnesses. (b) Witness verification latency and number of trie nodes.

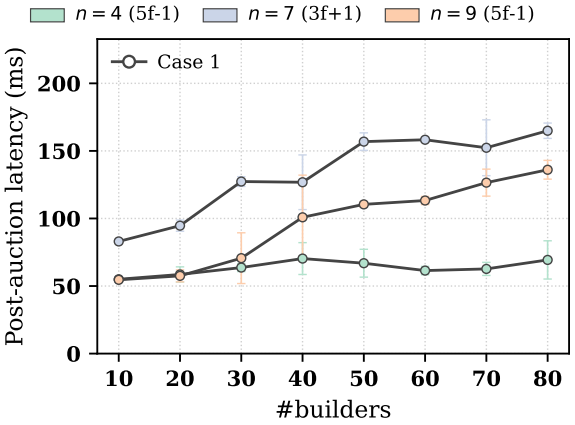

Figure 12: Under emulated WAN setting ($n = 4, 7, 9$), the post-auction latency (ms, consensus + block recovery) with number of builders of Case 2&3.

0.94) over the observed range up to 3.5×10^4 nodes. Verification cost is thus predictable from witness size alone, and unlike cold-start cost it does not amortize across bids.

E More results on $\Pi_{\text{Crab-BB}}$

E.1 Meet lower bound with more parties

In this section, we touch on a more interesting result for $\Pi_{\text{Crab-BB}}$. We can show several ways to reduce one round to achieve consensus and argue the protocol shown in Fig. 3 meets the best trade-off between latency and fault tolerance.

From the foundational work [1], we know that: "any protocol for censorship-resistance in partial synchrony secure against more than $\frac{1}{5}$-fraction of Byzantine faults requires 5 rounds". According to Theorem 8, $\Pi_{\text{Crab-BB}}$ matches this lower bound when $n = 4$ under $n = 5f - 1$. Moreover, any protocol under $n = 3f + 1$, though tolerate more faults, naturally matches this lower bound.

However, we can match this lower bound given larger n (*i.e.*, $f \geq 2$ when $n = 5f - 1$). The solution is to remove the **Exchange-2** step in the Fig. 3 and set the external validity of Π_{vbb} to just one WQ. It's trivial to show the Agreement, Termination, and Non-triviality. We can show this is sufficient for censorship-resistance and backed availability defined in Definition 1.

For censorship-resistance, the size of WQ is at least $n - f = 4f - 1$. For any input u made to all honest replicas, u is guaranteed to be implicitly in at least $4f - 1 - f = 3f - 1$ compressed input set. As $f \geq 2$, $3f - 1 \geq 2f + 1$, which implies such input u must be included to output interpretation subsequently. For availability-backed, the argument directly follows Theorem 3 as we didn't change the output interpretation in this optimization.

The reduced round can also be validated by our empirical analysis:

On the other hand, intuitively, we can show why fewer rounds is impossible for $f = 1$. First, the builder submission should be treated as one round and under good-case, the Π_{vbb} still need to take 2 rounds to finish. The key is that the two exchanging round can't be reduced to one because a simple WQ is not sufficient for censorship-resistance due to the number of honest replicas' input in WQ is less than the threshold of inclusion in output interpretation because $n - f - f < 2f + 1$ when $f = 1$. When the leader is faulty, it is able to insert stale bids (with smaller sequence numbers) to exclude honest replicas' input, and thus censoring certain bids.

Moreover, adding one more replica to $n = 4$ to $n = 5$ then assuming at most 1 can be faulty can also reduce one round to match the 4 round good-case latency. The key idea is still to make the WQ contain sufficient shares ($2f + 1$) from honest replicas.

E.2 Detailed evaluation results under emulated WAN setting

To accommodate deployments requiring $f = 2$, we also evaluate $n = 7$ for the $n = 3f + 1$ and $n = 9$ for the $n = 5f - 1$, colocating instances on a single host and emulating the WAN setting with `netem` rather than renting additional CVMs. We extend the measured $n = 4$ topology round-robin: each additional PE is assigned to the next cloud provider's region in rotation, and inter-PE links are configured with the RTTs measured for the corresponding provider pair in Fig. 5.

Fig. 12 reports latency of Case 1, under the same $5f - 1$ configuration, $n = 4$ attains the lowest latency and remains nearly flat across builder counts, whereas $n = 9$ starts comparably low but degrades as builders increase ($2\times$ latency compared with $n = 4$ when under high load): the SQ each PEs exchanges during $\Pi_{\text{Crab-BB}}$ grows with both the builder count and n, reaching 42.7 KB at $n = 4$ versus 114.1 KB at $n = 9$ for 80 builders. The $n = 7$ ($3f + 1$) configuration is consistently 50-70 ms slower than both $5f - 1$ configurations despite its smaller committee—reflecting the extra communi-

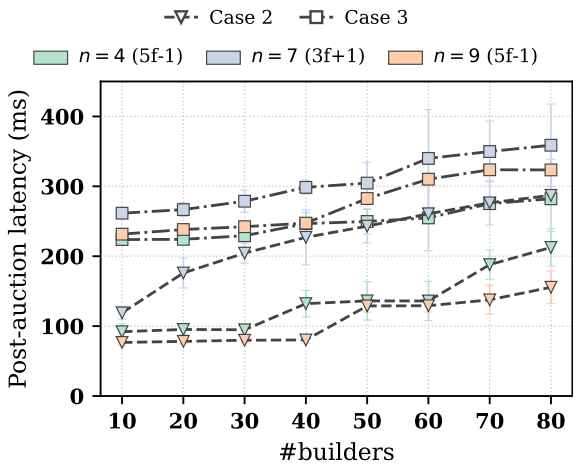

Figure 13: Under emulated WAN setting ($n = 4, 7, 9$), the post-auction latency (ms, consensus + block recovery) with number of builders of Case 2&3.

cation round that $3f + 1$ requires.

Fig. 13 reports latency of Case 2&3. In Case 2, where the first leader is honest, $n = 9$ attains the lowest latency, below even $n = 4$, as when $n = 9$ the parties can enter the Π_{vbb} in a single exchange round and a WQ of size $n - f$ suffices for solving Crab-BB problem (as we showed in Appendix E.1). Case 3 additionally triggers a view change, costing 4Δ timeout, but latency stays below 310 ms for both $5f - 1$ configurations. The $n = 7$ ($3f + 1$) configuration remains slower than both $5f - 1$ configurations, again reflecting its extra communication round.

The evaluation shows that, although $n = 3f + 1$ provides higher fault tolerance for a fixed committee size, it requires an additional communication round to implement Π_{vbb} [2], making it less attractive for PBS's latency-sensitive setting. In contrast, $n = 5f - 1$ achieves lower latency at the cost of a larger committee. Though adding more parties from $n = 4$ to $n = 9$ can reduce one round theoretically, but under our configuration, such benefit can only occur obviously under Case 2 when some PEs are faulty. In other cases, the $\Pi_{\text{Crab-BB}}$ requires a PE to disseminate larger messages and thus worsen the latency.

In practice, $n = 4$ already removes the relay's single point of failure while providing a favorable balance between performance and fault tolerance. Moreover, the limited diversity of availability zones/TEE implementations in practice makes it impractical to instantiate too many PEs without weakening the underlying trust assumptions. Therefore, the choice of $n = 4$ and $f = 1$ reflects this practical design point to get the best of the two worlds: it is the smallest deployment that tolerates the highest fraction of faulty TEE while achieving optimal latency practically in optimistic path.

Listing 1 Key structures and interfaces for SC_R

```
contract DPaaSRegistry {
    struct PEInfo {
        string endpoint;
        bytes  blsPubkey;
        bytes  asymPubkey;
        string provider;
        string zone;
        bytes  blsSig;
    }
    struct Deployment {
        address  validator;  // ETH withdrawal
        ↪ address of the validator
        PEInfo[] pes;
        bytes32  manifestHash;
        bytes    aggregatedPubkey;
        bytes    aggregatedSigOverManifest;
    }
    mapping(bytes32 => Deployment) public deployments;
    function register(
        bytes     calldata validatorBLSPubkey,
        address            validatorAddress,
        PEInfo[]  calldata pes,
        bytes32            manifestHash,
        bytes     calldata aggregatedPubkey,
        bytes     calldata aggregatedSigOverManifest,
        bytes     calldata sig,
        bytes     beacon_proof
    )
    function deregister(
        bytes     calldata validatorBLSPubkey,
        bytes     calldata sig,
    )
    function getPEManifest(bytes calldata
    ↪ validatorBLSPubkey)
        external view
        returns (bool found, PEInfo[] memory pes)
}
```

F Smart Contracts

In this section, we provide a high-level overview for the smart contracts that we develop and leverage in DPaaS, including the data structures and function interfaces.

Registry Contract. Listing 1 shows basic structure of DPaaS deployment's metadata and interface for SC_R.

We require both validator deposits and DPaaS registration to be performed by the user controlling the withdrawal key $(sk_{\text{W}}, pk_{\text{W}})$. Accordingly, in the registration contract SC_R. The `register` verifies an ECDSA signature included in the call data. The contract will check (1) if the ECDSA signature is from a valid validator; (2) if the validator's withdrawal address is tied to the validator public key in the beacon state. The contract also provides the `deregister` call for the user. Given the (2) is checked in the `register`, the `deregister` only checks if the ECDSA signature is from a valid validator and if the withdrawal credential matches the address of validator in the deployment.

To change the registered deployment metadata, the user must first deregister and then submit a new registration. This

Listing 2 Key structures and interfaces for SC_E

```
contract OperatorClaim {
    // EIP-4788 Beacon Roots Contract
    address constant BEACON_ROOTS =
    ↪ 0x000F3df6D732807Ef1319fB7B8bB8522d0Beac02;
    struct Escrow {
        address operator;
        bytes32 validatorPubkeyHash;
        uint256 reward;
        bool    claimed;
    }
    mapping(bytes32 => Escrow) public escrows;
    function claim(
        bytes32   escrowId,
        uint64    timestamp,
        bytes32[] calldata pubkeyProof
    ) external {
        /// EIP-4788: fetch the beacon block root
        ↪ committed at `ts`.
        bytes32 beaconRoot = _beaconRoot(timestamp);
        require(
        /// Using SSZ.
        _verifyValidatorPubkey(beaconRoot,
        ↪ escrow.validatorPubkeyHash, pubkeyProof),
            "bad pubkey proof"
        );
        ...
        escrow.claimed = true;
        (bool ok, ) = msg.sender.call{value:
        ↪ escrow.reward}("");
        ...
    }
}
```

prevents unauthorized or silent modification of the public registry state.

For builder side verification, the builder can get the manifest file for any DPaaS validator from SC_R. Note that such read operation doesn't cost gas as they can be done in the builder's local synced node because of the `view`. If a validator is not enrolled through DPaaS, then the builder will get an empty record from the storage. We consider a fake registration (*i.e.*, a non-DPaaS validator register itself as a DPaaS validator) as a behavior that only hurts the validator itself, as this require gas cost but will not pass the builder verification.

Escrow Contract. Listing 2 shows a simplified version of SC_E and its claim procedure. After a DPaaS proposer successfully proposes a block through the operator's deployment, the operator may claim the corresponding reward. The operator address is fixed at contract deployment, so only the registered operator can submit claims. To claim, the operator provides the block timestamp and a SSZ proof [24] tied to the proposer's public key. The contract verifies the corresponding beacon root using EIP-4788 [82] and enforces that each proposed block can be claimed only once.