DPaaS: Improving Decentralization by Removing Relays in Ethereum PBS

Chenyang Liu
Duke University
chenyang.liu@duke.edu

Ittai Abraham
Intel Lab
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 improves decentralization and scalability by offloading block construction to specialized builders. In practice, MEV-Boost implements PBS via a side-car protocol with trusted relays between proposers and builders, resulting in increased centralization as well as security (e.g., block stealing) and performance concerns. We propose Decentralized Proposer-as-a-Service (DPaaS), a deployable architecture that eliminates centralized relays while preserving backward compatibility with Ethereum's existing consensus layer. Our insight is that we can reduce centralized trust by distributing the combined roles of the proposer and relay to a set of Proposer Entities (PEs), each running in independent Trusted Execution Environments (TEEs). For compatibility, DPaaS presents itself to Ethereum as a single validator, leveraging threshold and aggregation properties of the BLS signature scheme used in Ethereum. At the same time, DPaaS protocols ensure fair exchange between builders and proposers even in the face of a small fraction of TEE failures or partial synchrony in networks. Our evaluation, deployed across four independent cloud hosts and driven by real-world traces, shows that DPaaS achieves $\leq$ 5 ms bid processing latency and 55.75 ms latency from the end of auction to block proposal – demonstrating that DPaaS can offer security and decentralization benefits while providing strong performance.

1. Introduction

Maximal Extractable Value (MEV) [1], [2] 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 resources (e.g., trading companies) [3], [4], [5], [6]. This has led to the *Proposer-Builder Separation (PBS)* [5] architecture proposed by 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 make the market for builders more open and competitive, and ensures that all validators have equal opportunity to capture MEV.

Unfortunately, it is hard to realize PBS in practice without introducing new forms of centralization. First, the implementation must ensure fair exchange [7]: proposers should receive available and valid blocks from builders, which include fee payment to the proposer, while builders should have the confidentiality of their block contents protected unless selected for the slot. Supporting such fair exchange requires the introduction of a trusted party, increasing centralization. Second, the implementation must deliver strong performance, given that there is a trend towards shorter slot times [8] and that MEV is very sensitive to latency [9], [10]. Any decentralized solution inherently adds latency overhead due to communication between entities. Third, the implementation should be readily (and incrementally) deployable. While potential PBS implementations could consider modifying the validator committee protocol for endorsing a block to improve decentralization, any changes to Ethereum's core protocol typically take years to be finalized [11], [12], [13]. Additionally, deployable implementations should consider maintaining compatibility with the broader ecosystem (e.g., bid cancellation for builders [14]).

Today, PBS is implemented via MEV-Boost [15], a sidecar protocol that introduces *relays* to serve as trusted auctioneers and escrows. The primary drawback is the inherent trust placed in relays by builders and proposers. There is no underlying mechanism to prevent exposing non-winning block payloads [16] to steal MEV or potentially running a biased auction. Additionally, relays are highly centralized, with over 90% of blocks in Ethereum routed through just five relays and 84% controlled by only three companies [17]. Moreover, relays incur performance overheads due to their *commit-and-reveal* scheme, which involves multiple rounds between the proposer and relay to complete the fair exchange. This leads to the proposer capturing less MEV due to ending the auction earlier [10], and could be amplified by shorter slots in the future [8].

Given the growing consensus that centralized, trusted relays should be eliminated, there have been several new proposed implementations of PBS [18], [19], [20], [21]. TEE-Boost [20] leverages Trusted Execution Environments (TEEs) for builders to prove the validity of their block to a proposer without revealing its contents. While TEE-Boost removes trust in relays for block validity, it does not address the tension between block availability and losing block's privacy in the fair exchange. Other proposals [18], [19] explore modifications to Ethereum's consensus layer, which makes deployment challenging. A proposal by Paradigm [19] extends TEE-Boost and addresses block availability through

the use of Silent Threshold Encryption [22] for the attester committee to decrypt the payload. However, it does not address potential bias in the auction and can pose issues with supporting the dynamic availability goals in Ethereum. Enshrined PBS (ePBS) [18] proposes staking builders to facilitate fair exchange. Although ePBS remains under active development with no scheduled hard fork, Ethereum's long-term roadmap envisions its eventual adoption. However, even in an ePBS world, sidecar protocols will matter, as proposers can still source blocks from both ePBS and external relays, which offer lower entry barriers for builders and often higher profitability [23].

Our insight is that we can remove centralized trust and improve performance by decentralizing the combined roles of the proposer and relay. Based on this insight, we introduce Decentralized Proposer-as-a-Service (DPaaS), a deployable architecture that leverages a distributed set of TEEs. Users enroll as validators through a given deployment that meets their security policy (e.g., different cloud providers or TEE implementations). When selected as a proposer, DPaaS handles all of the responsibilities, 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.

We need to address several challenges to realize DPaaS. First, to preserve full compatibility with Ethereum's existing consensus protocol, DPaaS must appear as a single entity (*i.e.*, tied to one public key) with respect to the rest of the Ethereum network, despite being distributed across multiple TEEs. Second, despite the presence of a small fraction of faulty TEEs [24] and/or partially synchronous networks [25], DPaaS must maintain security guarantees for ensuring fair exchange. Third, as a decentralized auctioneer, DPaaS should achieve performance comparable to existing centralized, relay-based systems, ensuring that its decentralized design remains practically deployable without compromising efficiency.

To summarize, our contributions include:

- We propose Decentralized Proposer-as-a-Service (DPaaS), a deployable implementation of PBS for Ethereum that eliminates reliance on centralized relays while preserving backward compatibility.
- We design a suite of efficient, fault-tolerant protocols that enable DPaaS to provide security guarantees while meeting strict latency requirements.
- We implement a prototype of DPaaS and deploy it across four hosts spread over different cloud providers and availability zones. Our evaluation, driven by real-world bid traces, demonstrates DPaaS (on average) processes bids in less than 5 ms and can propose blocks within 55.75 ms once the auction completes (leading to 3% more MEV).

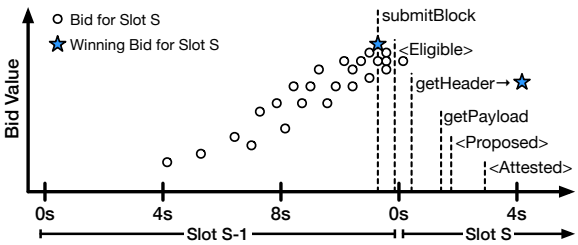

Figure 1: Example slot timeline with MEV-Boost.

2. Background

In this section, we present additional background on Ethereum and MEV-Boost [15], along with key motivations for design of DPaaS and building blocks we use.

2.1. Ethereum and MEV-Boost

Ethereum currently operates under a Proof-of-Stake (PoS) consensus protocol known as Gasper [26]. Participants become *validators* by staking 32 ETH [27]. 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.

Today, MEV-Boost implements PBS via a sidecar protocol that allows the proposer to outsource block construction and MEV extraction to a market of builders. To support the fair exchange between the builders and proposer, the third parties known as *relays* are widely used. Relays act as trusted 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 in MEV-Boost, with a focus on the winning bid (star). Builders start constructing blocks for slot S during slot $S-1$ and submit their blocks and bids to relays 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[1] (signed block header and beacon block body); 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.

Several factors make MEV-Boost a highly timing-sensitive sidecar protocol. First, as shown in [29], [10], bid value increases with time, incentivizing proposers to delay for cocommitting to higher bids. Second, the strict attestation deadline [30] means that excessive delay risks

1. Beacon Block is the block in Ethereum Beacon Chain which was merged with the original Ethereum proof-of-work chain in September 2022 and became today's Ethereum proof-of-stake chain [28].

a missed slot [10]. As a result, proposers favor minimal latency to end the auction as late as possible while still ensuring their block reaches enough attesters in time.

Remark To eliminate the need for trusted relays, we must find an alternative that provides both fair-exchange security guarantees and comparable performance with relays. TEE-based solutions are well suited for this compared to cryptographic approaches such as zero-knowledge proofs or multi-party computation, which have higher performance overhead.

2.2. Trusted Execution Environments (TEEs)

TEEs enable Secure Remote Execution (SRE) on a platform provided by an untrusted infrastructure provider via trusted hardware. TEE-based enclaves or confidential virtual machines (CVMs) can natively run on the CPU with hardware-assisted isolation which provides confidentiality and integrity protections for application code and data. Additionally, the application owner can determine the identity of the application running via remote attestation. Major cloud providers have adopted many TEE implementations, including: Intel SGX [31], Intel TDX [32], and AMD SEV [33].

However, there are several important considerations regarding the security guarantees. First, there are growing numbers of side-channel attacks against TEEs, particularly against confidentiality [34], [35], [36], [37], [38], [39], [40], [41]. While less common, it is possible for a break of confidentiality to lead to a break of integrity when attestation-related secrets are compromised [34], [42], [43]. Second, most TEEs do not provide availability guarantees due to (untrusted) privileged software.

Remarks We should consider a *security-in-depth* approach when using TEEs. A single TEE serving as a relay can still lead to a single point of failure (e.g., MEV stealing attacks) due to vulnerabilities present in the TEE ecosystem (e.g., side-channel attacks). Therefore, in DPaaS, we leverage a *group* of TEEs to mitigate cases in which a small fraction of them may be compromised by an adversary. Next, we will introduce relevant background on threshold cryptography schemes that supports such a distributed design.

2.3. Threshold Cryptography Schemes

Secret Sharing (SS) A (t, n)-secret sharing scheme [44], [45] enables a dealer to divide a secret among n players so that only groups of at least t players can reconstruct it. In Shamir's Secret Sharing [44], the dealer encodes the secret as a random polynomial and gives each player one share. Any subset of at least t honest players can later combine their shares to recover the original secret via Lagrange interpolation [46].

Threshold Signature Scheme (TSS) A (t, n)-threshold signature scheme (TSS) supports n signers where subsets of size $\geq t$ can produce a signature on a message m which can be verified using a single public key regardless of the subset. Boneh–Lynn–Shacham (BLS) scheme [47] is a TSS that is heavily used in Ethereum [48]. To support threshold signatures, a (t, n) secret sharing scheme is used to distribute shares to all n signers. Each signer computes a partial signature on the message, which the aggregator verifies and then interpolates to reconstruct the final signature (equivalent to using the original secret key).

Remarks While secret sharing and threshold signatures are useful, they require the different parties to agree on the same values; for instance, two parties trying to reconstruct different secrets will fail. In our work, this relates to parties having different views of the winner of the auction. To address disagreements that may arise, we require consensus between the TEEs in each slot.

2.4. Byzantine Broadcast/Consensus

Byzantine fault-tolerant broadcast [49], [50] ensures that all honest parties agree on a value broadcast by a designated leader, matching the leader's input if it is honest. While theoretical work emphasizes worst-case latency, practical systems prioritize good-case performance under a stable, non-faulty leader. Abraham *et al.* [51] classify good-case broadcast latency across synchrony models.

DPaaS adopts the $(5f-1)$-psync-VBB (Validated Byzantine Broadcast) protocol [51], [52], achieving two-round good-case latency and optimal performance for $n = 4$, $f = 1$. This configuration suits MEV-Boost's tight block proposal deadlines: although both $n \geq 3f+1$ and $n \geq 5f-1$ guarantee agreement under partial synchrony, the latter provides faster finalization with identical fault tolerance in the typical $f = 1$ setting.

3. Overview

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

3.1. Goals

We summarize the design goals below, focusing on a given slot in which DPaaS instance acts as the proposer:

1) **Bid Selection** The block winning the auction must correspond to the highest bid among the set of all non-canceled and eligible[2] bids.
2) **Block Validity** The winning block payload must embed the builder's bid as payment to the proposer and all honest validators can accept and vote for the block.
3) **Block Availability** DPaaS must propose a winning block.
4) **Failed Trade Privacy** DPaaS must not reveal transactions contained in any non-winning blocks' payloads.

2. The set of eligible bids is the subset of all submitted bids that has been validated and will be considered in the auction. This follows from the definition in relay implementations [53]. We will refine this definition later in the context of our proposed protocols.

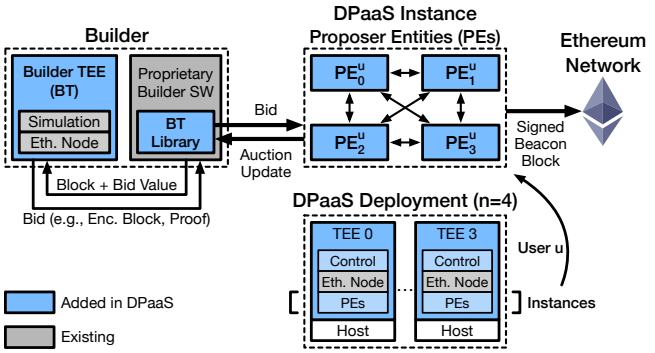

Figure 2: Overview of DPaaS architecture.

5) **Compatibility** DPaaS must operate solely as a sidecar, without requiring major modifications to the Ethereum consensus protocol changes to Ethereum's consensus protocol (*i.e.*, validator duties) or to the existing builder market (*i.e.*, how builders construct and submit blocks).
6) **Performance** DPaaS 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., length of wrapping up the auction).

In the current MEV-Boost architecture, goals 1-4 are only met when relays operate correctly in a neutral manner and are trusted by all participants; however, there are no mechanisms to enforce this. Our design of DPaaS meets these goals with a design that avoids unconditional trust.

3.2. 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 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; any principal can become an operator by setting up and running a DPaaS deployment. Each DPaaS deployment hosts an arbitrary number of DPaaS instances, each of which is made up of n Proposer Entities (PEs) executing 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 runs an Ethereum node internally, which handles block proposal after the auction finishes. The PEs in a given instance (e.g., PE_i^u for user u) jointly execute the roles of the validator on behalf of the user; our focus, in this paper, is when the validator becomes a proposer for a given slot. Therefore, the PEs are responsible for running an auction over blocks and bids submitted by builders, determining the winning block, and proposing a Signed Beacon Block to the attester committee within the slot deadline.

On the builder side, most of the software remains unchanged (e.g., receiving transactions, 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 and Ethereum execution node (e.g., Reth [54]). If successful, the BT creates a bid request containing sealed payload to the PEs, which serves to: 1) protect the confidentiality of the block execution payload, but 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").

3.3. Threat Model

There are three main classes of mutually distrusting principals in DPaaS: builders (construct and submit blocks to DPaaS), operators (set up and manage DPaaS deployments), and users (enroll as validators/proposers through DPaaS). The adversary wants to violate any of the security goals (1-4), where goals 1-3 relate to attacks against enrolled users and goal 4 relates to an attack against builders. We assume partially synchronous communication [25], indicating that there exists an unknown Global Synchronization Time (GST), after which communication delays within the network are bounded by Δ. All messages sent eventually arrive after GST. We assume loosely synchronized clocks (i.e., on the order of milliseconds), which is typical for cloud hosts [55].

Each builder hosts its own proprietary software (e.g., block construction algorithm) along with the DPaaS Builder TEE (BT) on a platform 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. Note that attacks against the confidentiality of the TEE platform that lead to compromises of integrity (e.g., leaking attestation keys [34], [42], [43]) are out of scope.

An operator sets up a DPaaS deployment that consists of n cloud-based TEEs (e.g., confidential VMs) that execute DPaaS software. When a user enrolls as a validator, the operator spawns a new DPaaS instance with n Proposer Entities (PEs) spread across the TEEs that act on behalf of the user. Since an operator administers the TEEs, they have the ability to unilaterally shut them down; however, we argue that monetary incentives via payments from enrolled users will prevent the operator from doing so, and thus we consider this as out of scope. The adversary may compromise confidentiality and integrity all software outside of the TEEs

for all n hosts; however, we assume this does not violate availability. Furthermore, the adversary may compromise up to f of the $n \geq 5f - 1$ hosts by either: 1) breaking integrity or confidentiality guarantees of the TEE platforms or software running inside of the TEE, or 2) breaking availability guarantees of PEs. Such restrictions on the adversary may be achieved by distributing the TEEs across different cloud providers in different availability zones and leveraging different TEE implementations (*e.g.*, Intel TDX, AMD SEV, and ARM CCA) to provide independence between compromised hosts. We will reflect on this later and discuss how builders and users can determine whether a given operator's instance meets their trust assumptions.

Note on Asymmetry As discussed above, we assume the adversary may violate the integrity of some PEs, up to a bound (f out of n), but not BTs. Most modern attacks against TEEs break confidentiality guarantees for code and data within the TEE; for instance, single-stepping attacks leverage fine-grained visibility into TEE execution to infer sensitive information [56]. Despite the above, we want to accurately capture the strongest possible threat model such that our DPaaS design still remains correct. Recent PBS proposals [19], [20], [57] also introduce TEEs at builders and highlight the fact that: 1) bugs in relays are more common than TEE integrity vulnerabilities, and 2) the scope of the such a violation is relatively small (i.e., missed slot) and can be detected. A stronger adversarial model on the PE side is beneficial because the scope of impact for attacks is larger (*i.e.*, no guarantee on fair exchange at all).

4. DPaaS Design - Offline Setup

In this section, we will describe how an operator can set up a DPaaS deployment (Sec. 4.1), and how a user can enroll as a validator/proposer in Ethereum through the given deployment (Sec. 4.2).

4.1. Bootstrapping a DPaaS deployment

When a principal wants to become a DPaaS operator by setting up and running a deployment, they are first responsible for determining n and configuring that many TEEs. The operator is responsible for determining how the TEEs are deployed to ensure that at most f may be compromised by the adversary (as per Sec. 3.3). This failure isolation may be achieved through a combination of using different cloud service providers (*e.g.*, AWS and Azure), different regions and availability zones within a provider (*e.g.*, US-East-1 and US-East-2 in Azure), and different TEE hardware implementations (*e.g.*, AMD SEV and Intel TDX). The operator configures the TEEs to load DPaaS software images consisting of the control, Ethereum node, and PE applications. The DPaaS provider must capture the configuration in a deployment manifest file which contains information on all of the endpoints in the instance, including: 1) their IP addresses and ports for the control application, 2) their placement information (i.e., provider, region, zone), and 3) the hardware and firmware information (*i.e.*, type of CPU, version of firmware). The provider will publish this manifest file publicly, which will be used by users referring to enroll in DPaaS.

4.2. Enrolling User in a DPaaS Instance

Given multiple DPaaS deployments from different operators, users can choose which deployment to enroll through based on: 1) whether the PE configuration meets their security policy, and 2) the price charged by the provider. While (2) is outside of our scope, we enable a user to check (1) by inspecting the manifest file and attesting the PEs through TEE remote attestation.

When enrolling a user, the operator instructs the control application on each TEE to spawn a new PE; all of these PEs together form the new DPaaS instance. The operator then creates an instance manifest file and provides it to the user, which contains information on the ports for PEs in their instance. The user establishes connections to all of these PEs and verifies that the instance complies with the user's security policy through remote attestation. Although different users could have different security policies, we conclude the following four properties as the basic requirement: (1) the PE runs on valid trusted hardware (expected CPU model and firmware), (2) the PE's image including PE software matches a known measurement of the correct implementation, (3) the PE is deployed by the correct cloud provider as claimed by the provider, and (4) the PE has a specific placement within the cloud provider's infrastructure (*e.g.*, region and availability zone) as claimed by the provider. At this point, the user has secure, authenticated channels to each PE.

Outsourcing of User's Validator Key In current Ethereum, each validator can possess two distinct BLS key pairs [58]: 1) a validator key pair $(sk_{\text{V}}, pk_{\text{V}})$ for validator operations (e.g., block proposal and attestation), and 2) a withdrawal key pair $(sk_{\text{W}}, pk_{\text{W}})$ for controlling access to funds. DPaaS leverages this separation by making PEs generate and manage validator private key sk_{V} for operational duties, while allowing users to generate and retain sole control over the withdrawal key (and thus their underlying funds).

DPaaS employs distributed key generation (DKG) protocol [59], [60] to generate the validator key without relying on any trusted third party. Each PE_i generates $(sk^i_{\text{V}}, pk^i_{\text{V}})$ independently. The PEs then exchange their public part pk^i_{V} with one another to compute the aggregate BLS public key [61] $pk_{\text{V}} = \sum_{i=0}^{n-1} pk^i_{\text{V}}$, which is subsequently provided to the user. However, to support threshold signatures (with $t < n$), each party generates a partial BLS key pair $(sk^i_{\text{p\_sign}}, pk^i_{\text{p\_sign}})$ [62] which enables interpolating the signature from any t PEs to that of a signature over the same message with sk_{V} (without any PEs knowing sk_{V}). Finally, each PE_i generates a public-secret key pair $(pk^i_{\text{asym}}, sk^i_{\text{asym}})$ independently, which builders will use to encrypt secret shares for specific PEs during the auction (see Sec. 5.3).

In this way, even though DPaaS is decentralized, the rest of the Ethereum network sees it as a single entity with the

$(pk_{\text{V}}, pk_{\text{W}})$ 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.

Configurable Proposal Time During and after enrolling, the user can specify a target block-proposal time t_p and a duration T, requiring DPaaS to end the auction at $t_e = t_p - T$. In practice, the user will configure T based on the latency of the post-auction process in DPaaS, which depends on the inter-PE latencies; we will later provide best and worst case bounds for T based on analysis of our runtime protocol. Note the user could also directly specify the t_e if desired.

4.3. Outcome

After a successful offline setup, users offload validator duties to n mutually attested PEs in a DPaaS instance. Each PE_i holds three key pairs: a BLS validator key pair $(sk^i_{\text{V}}, pk^i_{\text{V}})$ for multi-signatures, a BLS partial signing key pair $(sk^i_{\text{p\_sign}}, pk^i_{\text{p\_sign}})$ for threshold signatures, and an asymmetric key pair $(pk^i_{\text{asym}}, sk^i_{\text{asym}})$ for secret share encryption. All secret keys are strictly local to each PE_i and are never shared with other PEs or external entities (*e.g.*, user or operator). Finally, the user holds the withdrawal key pk_{W}, which allows them to unilaterally exit from their role as a validator at any time.

5. DPaaS Runtime Protocol

In this section, we will introduce the runtime design of DPaaS in which an enrolled validator acts as the proposer for a certain slot. We begin by demonstrating the notations (Sec. 5.1) and calling back the goals and formalize them into properties (Sec. 5.2). We will next elaborate on the runtime protocol, focusing on bidding during the auction (Sec. 5.3) first. Then we look at how DPaaS finalizes the auction up through the construction of the Signed Beacon Block (Sec. 5.4). We will leave detailed proofs for properties to Appendix A.

5.1. Preliminaries

We focus on a particular slot starting at $t = 0$ in which a DPaaS instance for a particular enrolled user is selected as the proposer. The DPaaS instance consists of a set of PE_i where $i \in [0, n-1]$. We denote the builders participating in the slot as $\mathcal{B}$, where each $\texttt{b} \in \mathcal{B}$ (identified by public key) submits one or more bids from the start of last slot ($t = -12$s) to this slot's attestation deadline ($t = 4$s). For each PE_i, we refer to the set of bids it receives from builders as R_i. For simplicity, we define each bid visible to every PE as a tuple $(\texttt{b}, \texttt{bv}, \texttt{seq}, \texttt{h}, \texttt{att}, \texttt{spl})$ which respectively refer to the builder, bid value (in R^*), sequence number (in $\mathbb{N}$), block hash, TEE attestation generated by BT, and sealed payload. We will refer to the subset of bids from builder b in any set of bids S as $S[b]$.

Note that before GST in a partial synchronous network, there is no guarantee on any message delivery (which includes bid submissions). This implies in the worst case $R_i = \emptyset$. Because of the inherent uncertainty before GST, DPaaS does not provide any properties concerning that. Instead, we provide properties when GST comes early enough before the end of the auction with the network latency bounded by a known Δ thereafter.

In contrast to current relays, DPaaS executes auctions in a distributed manner, which necessitates distinguishing between eligibility and cancellation local to a given PE_i versus globally across PEs.

Definition 5.1 (Locally Eligible)**.** *For PE_i, the set of locally eligible bids is:*

$$E_i = \big\{r \mid r \in R_i \wedge \text{PE}_i \textit{ has verified } r.\texttt{att} \wedge \forall s \in [0, r.seq), \exists r' \in E_i[r.\texttt{b}] : r'.seq = s\big\}$$

We use a recursive set definition here for simplicity, which captures the fact that the bids for any builder b in E_i (i.e., $E_i[b]$) must be continuous with respect to their sequence numbers, starting from 0.

Definition 5.2 (Globally Eligible)**.** *The set of globally eligible bids E is defined as:*

$$\check{E} \subseteq E \subseteq \hat{E} \textit{ where}$$
$$\check{E} = \{e \mid 2f + 1 \leq |\{i \in HonestPEs \mid e \in E_i\}|\}$$
$$\hat{E} = \{e \mid f + 1 \leq |\{i \in HonestPEs \mid e \in E_i\}|\}$$

There is no single definition for E, since global eligibility is defined across both honest and Byzantine PEs. The locally eligible sets for the (up to f) Byzantine PEs are *unknown*, but we can reason about the possible effects on the globally eligible set. The lower bound $\check{E}$ contains a bid if it is in at least $2f+1$ E_i's for honest PEs; no matter what the Byzantine PEs do, they should not be able to lead to a smaller globally eligible set. On the other hand, the upper bound $\hat{E}$ contains a bid if it is in at least $f + 1$ E_i's for honest PEs, which may only be realized if the Byzantine PEs receive that bid and behave honestly with respect to that bid. In any execution, E must sit between $\check{E}$ and $\hat{E}$.

Definition 5.3 (Globally Active)**.** *The set of globally active bids is:*

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

While E represents globally eligible bids, we want to only consider the most recent bid from each builder when selecting the winning bid. A contains at most one bid from each builder $b \in B$ with highest sequence number.

5.2. Formal Properties

We formalize the security goals 1-4 and part of goal 5 (see Sec. 3.1) as follows:

Goal 1. *Bid Selection: The winning bid w is the highest globally active bid.*

$$w = \arg\max_{e \in A}(e.\texttt{bv})$$

Goal 2. *Block Validity: For the winning bid w, the builder should pay at least w.bv to the enrolled user through a set of transactions in the reconstructed w.pl. Also, all the transactions inside w.pl can be successfully simulated and executed by honest Ethereum validators.*

Goal 3.1. *Block Availability: If some valid bids become eligible in all honest* PE*s, eventually, an honest threshold ($\geq f+1$) of* PE*s should be able to reconstruct* pl *from* spl *and jointly produce the signed beacon block.*

However, in practice, Definition 3.1 is usually too weak for a timely protocol for current Ethereum (*i.e.*, $t = 4$s attestation deadline), so we consider another definition.

Goal 3.2. *Timely Block Availability: If some valid bids become eligible in all honest* PE*s, before a cut-off t_p, an honest threshold ($\geq f+1$) of* PE*s should be able to reconstruct* pl *from* spl *and jointly produce the signed beacon block.*

Goal 4. *Failed Trade Privacy: For each slot, there exists at most one winning bid w among the set of submitted bids whose payload is released; no party learns the payload of any other bid in plaintext.*[3]

Goal 5. *Effective Cancellation: For each builder* b $\in \mathcal{B}$*, let e denote the most recent bid across all* PE*s before $t_e - \Delta_{b2p} - \Delta_{pr}$. $\forall e' \in E[b], e'.\texttt{seq} < e.\texttt{seq}$ is considered canceled and must not be the winner.*

While our compatibility goal is broad, we specifically formalize our support for bid cancellation, which the current builder ecosystem leverages heavily [6], [9], [14], [29]. We assume an upper bound of latency between builders and PEs (Δ_{b2p}) and PE bid processing latency (Δ_{pr}). Given a bid submitted by the builder sufficiently early with respect to t_e, we guarantee under synchrony (or GST $= 0$ in partial synchrony) that all earlier bids are canceled.

Goals Under (Partial) Synchrony Under partial synchrony with unbounded latency before GST, some of the goals are impossible to guarantee. Therefore, we categorize our goals based on the level of synchrony required. Specifically, Goal 2, Goal 3.1, and Goal 4 hold under partial synchrony or synchrony, while Goal 1, Goal 3.2, and Goal 5 hold only under synchrony (i.e., GST = 0).

In the following discussion, we outline how our protocol design meets these goals, leaving our proof sketches to Appendix A.

3. Note that we do allow parties to learn the size of the payload and the fact it was submitted. We consider hiding such information is out of scope of this research.

5.3. Bidding During Auction

The bidding phase for slot S (which starts at $t = 0$s) begins during slot $S-1$. 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 and gather the necessary data (*e.g.*, parent hash) before they can construct a valid block.

5.3.1. Bidding at Builder. DPaaS preserves the core builder workflow used today with the exception of two changes: 1) introducing the Builder TEE (BT) used to submit bids to DPaaS-enrolled proposers, and 2) aggregating auction updates across the PEs (e.g., new highest bid). The BT provides a proof via TEE attestation to convey the validity of the block to the PEs without the PEs having access to the block payload in the clear. Note that TEEs for block validation is not novel [20]; we adopt it because it suits our setting and goals. Using TEEs results in lower latencies than cryptographic proof mechanisms (e.g., zero-knowledge proofs), making it practical for today's time-critical and competitive builder market. Unlike existing proposals [20] and commercial prototypes [57], [63], our design excludes untrusted builder software from the Trusted Computing Base (TCB), significantly reducing the attack surface.

To validate the block, BT synchronizes with beacon chain through its Eth Node. For each epoch, the BT retrieves a monotonic beacon state from its node. For each slot, it additionally queries a public registry to determine whether the designated proposer uses DPaaS and, if so, to obtain the corresponding PE instance manifest and public keys.

Once the builder constructs a block and determines a bid value, it submits the block with bid value to the BT. The BT will first perform initial checks followed by simulation of the block payload, before finally preparing the bid to return to the builder (which it can submit to the PEs).

Checks The BT performs basic checks on the input from the builder to ensure structural and contextual validity. The BT confirms all data fields (*i.e.*, slot number, parent hash) contained in the block matches the expected beacon state by connecting to other peers in p2p network. If any of these checks fails, the BT returns an error.

Block Simulation If initial checks are successful, the BT simulates the transactions in the block based on the most recent state (*i.e.*, related accounts' balance, transaction's nonce) on Ethereum Virtual Machine (EVM) for this slot. The simulation sequentially executes the transactions in the block and computes the payment value to the proposer of this slot. If simulation is not successful or the payment value is less than the claimed bid value, the BT returns an error.

Data Preparation & Submission Finally, if prior steps are successful, the BT prepares the necessary data for constructing the bid. Following the definition in Sec. 5.1, the bid consists of $(\texttt{b}, bv, \texttt{seq}, h, \texttt{att}, \texttt{spl})$. Among these fields, the BT extracts $(\texttt{b}, bv, h)$ directly from the builder's block (after checks). The BT provides the seq by keeping

track of previous bids in the slot, starting from $\texttt{seq} = 0$. The BT constructs the $\texttt{spl}$:

$$(\texttt{esshr} : \{\texttt{E}_{\texttt{pk}^{\texttt{i}}_{\text{asym}}}(K_i) \mid \texttt{H}(K_i)\}_{i=0..n-1},\ \texttt{E}_K(\texttt{pl}),\ \texttt{R}(\texttt{pl}))$$

by first randomly generating a key K to encrypt the payload as $\texttt{E}_K(\texttt{pl})$. The BT uses a $(f+1, n)-$secret sharing scheme to construct shares K_i where $i \in [0, n)$. For each K_i, the BT computes the hash $\texttt{H}(K_i)$ and encrypts it for the PE_i $E_{pk^i_{\text{asym}}}(K_i)$; the hash will be used by other PEs to verify decrypted shares during reconstruction (see Sec. 5.4). At this point, the BT generates a TEE attestation supplying the hash over all other elements of the bid as the user data field: $\texttt{att} = \texttt{att}\langle H(\texttt{b}, bv, \texttt{seq}, h, \texttt{spl})\rangle$. The BT returns the bid to the builder software, which can send it to the PEs.

5.3.2. Bidding at DPaaS instance. Prior to the end-auction time t_e, each PE_i processes bids to include them in their locally eligible set E_i. Upon receiving a bid r, PE_i: 1) checks $r.\texttt{seq}$ to maintain contiguous ordering of sequence numbers starting from 0, and 2) verifies $r.\texttt{att}$ to ensure that a trusted measurement of the BT, running on valid trusted hardware, constructed the bid. If successful, PE_i inserts r into E_i.

5.3.3. Update Top Bid. Currently, the builder ecosystem relies heavily on current top bid updates from relays to adjust their strategies. DPaaS supports this by enabling builders to subscribe to an update stream from each PE. If PE_i receives an eligible bid from a builder that is the new highest bid in E_i, then PE_i publishes an update to all subscribers. Because DPaaS distinguishes between local and global eligibility, builders rely on the BT library to merge updates from PEs. The library aggregates them to ensure that at least $f+1$ PEs agree on the top bid.

5.4. Post-Auction

After the auction deadline t_e, the PEs need to participate in a protocol to determine the winning bid w from all of the locally eligible sets E_i (*finalization*), reconstruct the block payload from $w.\texttt{spl}$ (*reconstruction*), and finally output an instance-level signature for the Signed Beacon Block (*interpolation*).

5.4.1. Finalization. At t_e, all PEs may have different locally eligible sets, even among the honest PEs. This may happen for many reasons, including: 1) differences in builder-PE_i latencies, 2) builder not sending bid to all PEs.

We present the protocol that each honest PE_i executes in Algorithm 5.1. As input to the algorithm, each PE_i computes V_i, which is a compressed representation of E_i, as follows:

$$V_i = \bigcup_{\texttt{b} \in \mathcal{B}} \arg\max_{e \in E_i}(e.\texttt{seq}) \text{ mapped to } (e.\texttt{b}, e.bv, e.\texttt{seq}, e.h).$$

Since all eligible bids from builder b in E_i must have contiguous sequence numbers starting from 0, denoting the maximum sequence number in V_i is sufficient. Additionally, we do not need to send $e.\texttt{att}$ or $e.\texttt{spl}$, since other PEs must have this information for any bid in their locally eligible set. Each PE_i exchanges their V_i with all other PEs (Line 1); note that we use $\langle\cdot\rangle_i$ to denote a message signed by PE_i.

If PE_i receives n distinct V_js (Line 2) within Δ, it uses them to construct and broadcast $\mathcal{V}^n$; this serves as a $n-$Quorum Certificate ($n-$QC) over the globally eligible bid set. Otherwise, PE_i waits to receive at least $n-f$ distinct V_js (Line 5), which it uses to construct and broadcast $\mathcal{V}_i$; this serves as a $(n-f)$-QC. This may happen in cases where one PE crashes in the first round or there are some message delays beyond Δ. Afterwards, PE_i waits (Line 8) until it has either a n-QC or $\geq f+1$ $(n-f)$-QCs, which serves as the input to the psync-VBB protocol. By requiring at least $f+1$ of the $(n-f)$-QCs, we ensure that the input contains at least one $(n-f)$-QC from an honest PE. The PE_i executes the psync-VBB protocol, which ensures that all honest PEs commit to the same input. We need to define a notion of *external validity* to constrain the values that (faulty) parties may propose, whereby each PE_i checks whether the leader's proposal is consistent with the QC(s) collected by the PE_i.

Under partial synchrony, Δ is a known upper bound only after Global Stabilization Time (GST); before GST, the network may be arbitrarily asynchronous, so the leader may not have any input to the psync-VBB protocol before GST. Honest PEs keep waiting until they finish the two rounds. Therefore, the definition of the external validity added to the basic protocol provides two guarantees:

- An honest PE will only join to propose and vote for a proposal when it has externally valid input of either

Algorithm 5.1 Finalization when $n = 5f - 1$

Input: $V_i \neq \emptyset$, upper bound of network latency Δ after GST. **Output:** winning bid w

/* Exchange-R1 */

1: **Broadcast** $\langle \texttt{Ex\_R1}, V_i\rangle_i$ to peers

/* Exchange-R2 */

2: **if** within Δ received n distinct $\texttt{Ex\_R1}$ messages. **then**

3: $\quad \mathcal{V}^n \leftarrow \{\langle \texttt{Ex\_R1}, V_j\rangle_j\}$ for all $j = 0 \ldots n-1$

4: $\quad$ **Broadcast** $\langle \texttt{Ex\_R2\_Fast}, \mathcal{V}^n\rangle_i$;

5: **else**

6: $\quad$ **Wait** until receiving at least $n-f$ distinct signed $\texttt{Ex\_R1}$ messages. $\mathcal{V} \leftarrow \{\langle \texttt{Ex\_R1}, V_j\rangle_j\}$ for $n-f$ $j \in \{0 \ldots n-1\}$

7: $\quad$ **Broadcast** $\langle \texttt{Ex\_R2\_Normal}, \mathcal{V}\rangle_i$.

8: **Wait** until either (i) $f+1$ signed $\texttt{Ex\_R2\_Normal}$ messages or (ii) one signed $\texttt{Ex\_R2\_Fast}$ message is received, and **set** the input to the collected $f+1$ $\mathcal{V}$ (case (i)) or $\mathcal{V}^n$ (case (ii)).

/* Consensus (psync-VBB) */

9: Run the protocol $(5f-1)$-psync-VBB with input until commit on Val.

/* Post-Consensus */

10: $w \leftarrow \mathcal{FW}(\text{Val})$

11: **return** w

Algorithm 5.2 Finalizing Winning Bid $\mathcal{FW}(\text{Val})$

Input: Val. **Output:** w
1: **if** Val is n-QC **then**
2: $\quad \mathcal{V}_c = \text{Val}$
3: **else**
4: $\quad \text{procPE} \leftarrow n * [\text{false}]$
5: $\quad$ **for** $\mathcal{V}_i \in \text{Val}$ **do** ▷ *deterministically by index i*
6: $\quad\quad$ **for** $V_j \in \mathcal{V}_i$ **do**
7: $\quad\quad\quad$ **if** $\text{procPE}[j] == \text{false}$ **then**
8: $\quad\quad\quad\quad$ Insert V_j to $\mathcal{V}_c$; $\text{procPE}[j] = \text{true}$
9: $\mathcal{A} \leftarrow \emptyset$
10: **for all** $\mathtt{b} \in \mathcal{B}$ **do**
11: $\quad V[\mathtt{b}] = \emptyset$
12: $\quad$ **for all** $V_i \in \mathcal{V}_c$ **do**
13: $\quad\quad$ Insert $V_i[\mathtt{b}]$ to $V[b]$;
14: $\quad \mathcal{A}[b] \leftarrow \text{ord}^{\downarrow}_{2f+1}\big(V[b];\ e \mapsto e.\mathtt{seq}\big)$
15: $w \leftarrow \arg\max_{a \in \mathcal{A}}(a.\mathtt{bv})$
16: **return** w

n-QC or $\geq f+1$ $(n-f)$-QCs
- The protocol can only proceed for the first view in which a leader has proposes externally valid input and sufficiently many followers vote for the proposal.

After consensus (Line 10), PE_i executes $\mathcal{FW}(\text{Val})$ to determine the winning bid, which we present in Algorithm 5.2. PE_i derives the globally active bid set $\mathcal{A}$ from the compressed locally eligible sets V_i included in either the n-QC or $\geq f+1$ $(n-f)$-QCs. Since sequence numbers are gap-free and start from 0, an honest PE_i holding the k-th bid also holds all earlier bids. Therefore, by aggregating all of the sequence numbers for each builder across the V_i's, we determine the $(2f+1)$-th largest sequence number (i.e., $\text{ord}^{\downarrow}_{2f+1}$) which implies that at least $2f+1$ PEs have the bid in their locally eligible set. This is important, since we need to ensure that $f+1$ PEs can reconstruct the block payload (discussed next); since consensus and reconstruction are necessarily split, we need to consider the case in which up to f PEs vote for a bid but are not willing to reconstruct (hence $2f+1$ versus $f+1$). Finally, PE_i selects the bid in $\mathcal{A}$ which has the highest bid value as the winner.

5.4.2. Reconstruction. Given a winning bid w, PEs must now reconstruct the block payload from $w.\mathtt{spl}$. As shown in Algorithm 5.3, 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 the hash contained in $w.\mathtt{spl}$. Once PE_i receives valid shares from at least $f+1$ PEs, it can reconstruct the symmetric key K and decrypt the $\text{E}_K(\mathtt{pl})$ contained in $w.\mathtt{spl}$.

5.4.3. Interpolation. At this point, at least $f+1$ PEs have access to the plaintext block payload for the winning bid w and we need to produce the signed beacon block to propose to the Ethereum network. In Ethereum, the final signature of the beacon block should be over the root hash of all

Algorithm 5.3 Reconstruction & Interpolation for PE_i

Input: w, $\mathtt{cdata}$, sk^i_{asym}, $sk^i_{\text{p\_sign}}$. **Output:** $(\mathtt{pl}, \sigma)$.
1: $S = \emptyset$, $I = \emptyset$.
2: **procedure** Share_sshr(w)
3: $\quad K_i \leftarrow \text{D}_{\mathtt{sk}^i_{\text{asym}}}(w.\mathtt{spl}.esshr[i].\text{E}_{\mathtt{pk}^i_{\text{asym}}}(K_i))$
4: $\quad$ Broadcast K_i to peers; Insert K_i to S.
5: **procedure** Reconstruction(w)
6: $\quad$ Run Share_sshr(w);
7: $\quad$ **while** $|S| < f+1$ **do**
8: $\quad\quad$ Upon receiving a share K'_j from PE_j:
9: $\quad\quad$ **if** $\text{H}(w.\mathtt{spl}.\mathtt{esshr}.\text{H}(K_j)) = \text{H}(K'_j)$ **then**
10: $\quad\quad\quad$ Insert K'_j to S.
11: $\quad K \leftarrow \mathtt{Recovery}(S)$; $\mathtt{pl} \leftarrow \mathcal{D}_K(w.\mathtt{spl}.\text{E}_K(\mathtt{pl}))$
12: **procedure** Share_psig(w, r)
13: $\quad \sigma_i \leftarrow \text{Sign}_{sk^i_{\text{p\_sign}}}(r)$
14: $\quad$ Broadcast σ_i to peers; Insert σ_i to I
15: **procedure** Interpolation(w,$\mathtt{cdata}$)
16: $\quad r \leftarrow \text{R}(\text{R}(\mathtt{cdata}), \mathtt{w.spl.R(pl)})$
17: $\quad$ Run Share_psig(w, r);
18: $\quad$ **while** $|I| < f+1$ **do**
19: $\quad\quad$ Upon receiving a partial signature σ_j from PE_j:
20: $\quad\quad$ **if** $\text{Verify}_{pk^j_{sign}}(\sigma_j, r) == \text{True}$ **then**
21: $\quad\quad\quad$ Insert σ_j to S.
22: $\quad \sigma \leftarrow \mathtt{Interpolation}(I)$.

fields of the beacon block, including the payload $\mathtt{pl}$ and $\mathtt{cdata}$. Instead of waiting to reconstruct the payload $\mathtt{pl}$ from $w.\mathtt{spl}$ to compute $\text{R}(\mathtt{pl})$, we instead optimize this by including $\text{R}(\mathtt{pl})$ in plaintext as part of $w.\mathtt{spl}$. Therefore, we can execute the Reconstruction and Interpolation in parallel, which reduces the latency from end-auction to proposal.

Each PE_i generates its partial signature σ_i and broadcasts this to all other PEs. 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_E. Each PE_i propagates the Signed Beacon Block on the Ethereum network after interpolation, allowing the attester committee to vote on it.

6. Implementation

This section describes the implementation details of DPaaS, which consists of several components: the BT, a library for builder software to interface with the BT, and the management and PE components for DPaaS deployments. We use Tonic gRPC framework [64] based on asynchronous Rust runtime Tokio [65] to support communication between the BT and PEs, as well as among the PEs themselves. We target AMD SEV-SNP [33] TEE hardware and leverage VirTEE's [66] open-source libraries. We implemented our prototype in Rust with a total of 6,835 loc—5,597 for the PE and 1,238 for the BT. We use Reth [54] and Lighthouse [67] as the Ethereum Node, with Reth serving as an archive node for our historical bid traces.

6.1. Builder TEE (BT)

We implemented BT in our prototype as an API `validate_bid` for untrusted builder, receiving request with block in plaintext with bid value and outputting bid submission to DPaaS. The `validate_bid` mainly contains three parts upon receiving a request: block validation, data preparation for the bid submission to send to PEs, and TEE proof generation.

For block validation, we implemented a new RPC called `validate_block_dpaas` in an execution client Reth [54] co-located with BTs in the CVM on AMD SEV-SNP enabled CPU. The `validate_block_dpaas` should take (1) block execution payload with raw transactions (*i.e.*, bytes) and (2) bid value as the input from builders. In the `validate_block_dpaas`, we sequentially execute all transactions against the beacon state of this slot. Then we compute the actual fee paid to the proposer. Specifically, we derive the net value by measuring the balance change of the beneficiary account—that is, the address designated by the proposer to receive block rewards—or, if present, by detecting a final transaction in the block that explicitly transfers funds to the proposer. The actual fee can be compared with the bid value claimed. The RPC returns success only if (i) all transactions executed with zero failure; and (ii) actual proposer fee is no less than the bid value.

We implemented data preparation for `spl` based on the open-source Shamir Secret Share library [68] with its Rust binding [69] and asymmetric encryption crate age [70]. To generate common hash included in `spl`, we used SHA256 in the SHA-2 crate [71]; to generate the Merkle root of payload $\mathtt{R}(\mathtt{pl})$, we used native tree_hash crate [72] for Ethereum for compatibility. We implemented TEE proof generation based on the AMD SEV-SNP's attestation, where a guest CVM could request attestation from the underlying processor.

6.2. Proposer Entity (PE)

We implemented four gRPC services for PEs: Enrollment, Auction, Consensus, and Recovery.

Enrollment. We implemented an `enroll` API in PEs and a simple client for the user of DPaaS to request `enroll` API for enrollment in a DPaaS instance. The client will take the manifest file as the input, send requests to `enroll` API in all endpoints, and compare the response with public available information (*i.e.*, measurement of PEs and public keys of PEs'). To verify the placement information, each PE queries the metadata service [73], [74] for its provider, region, and availability zone during `enroll`, and includes this information in the attestation report back to the user.

Auction. For auction, we implemented an API `submitBlock` for builders to request and an API `UpdateTopBid` for anyone to request. The `submitBlock` can be requested with all data mentioned in Sec. 5.1. We implemented an in-memory cache to store the cryptographic keys used for verifying TEE attestations in `submitBlock` API. Whenever a PE detects a new builder based on its public key, it queries the attestation service of the builder's hardware provider to obtain the corresponding verification key (*e.g.*, the VCEK for AMD). This key is then cached in memory, enabling faster verification for subsequent attestations.

Consensus. To determine the winner, we implemented the base protocol $5f-1$-psync-VBB [51]. To provide external validity for input, we also implemented the first two rounds of exchanging. Authentication is enabled by the combination of RA-TLS [75] and BLS signature. RA-TLS is to ensure the secure channel between PEs, while BLS is used for checking the integrity of relayed messages (*i.e.*, signed $\mathcal{V}^n$ exchanged in `Ex_R2_fast`). We also implemented the function $\mathcal{FW}$ shown in Algorithm 5.2.

Recovery. To facilitate the fast recovery of winning block, we implemented two APIs: `shareShare`, `shareBlsSignature` for every PE. We utilize the same secret share crate [69] as BT's and blsttc library [76] to support threshold BLS signature. We further leverage asynchronous execution in the Rust Tokio runtime to run block reconstruction and BLS signature interpolation concurrently.

6.3. Limitations

Block Attester Currently, Ethereum validators must serve as both proposers and attesters. We focus primarily on the role of proposer, which involves significant protocol design to support the auction and block proposal. However, in practice, DPaaS will also need to handle the attester role. This extension is straightforward, since each PE can use its local Ethereum node to validate a Signed Beacon Block, perform a partial signature, and then interpolate the signature (similar to how the PEs produce a Signed Beacon Block when acting as the proposer).

TEE Variety We have implemented and evaluated our prototype for AMD SEV-SNP. The design naturally extends to other confidential VM (CVM)-based TEEs, such as Intel TDX [32] or ARM CCA [77], which share the same high-level abstraction. Enclave-based TEEs like Intel SGX [31] can also be supported with additional engineering effort, which we leave as future work.

Attested TEE Placement When selecting a DPaaS deployment, a user's policy may require that TEEs are spread across different availability zones. Currently, major cloud providers have endorsement keys [78] for TEE attestation tied to individual providers; however, these keys are global across all availability zones for a provider. In our implementation, we rely on services from providers (e.g., Instance Metadata Service for AWS [74]), which exist outside the TEEs, to learn this placement information. This limitation could be removed if providers move towards finer-granularity keys in the future. In the meantime, users can enforce their own deployment policy by selecting n distinct cloud providers as DPaaS deployments, avoiding dependence on untrusted metadata services.

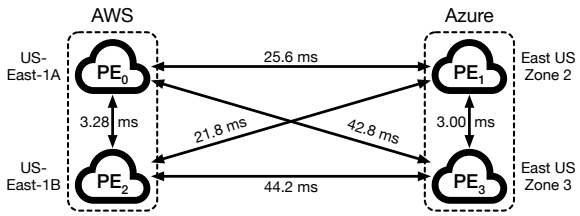

Figure 3: DPaaS deployment for our evaluation ($n = 4$), showing topology labeled by round-trip times (including signature signing and verification).

7. Evaluation

A key goal of DPaaS is to deliver performance comparable to existing relays. In this section, we evaluate DPaaS across four dimensions:

- BT **Overhead**: Performance penalty introduced in the builder due to BT. (Sec. 7.2)
- **Bid Processing Performance**: Latency between when a bid is received by DPaaS and when it becomes locally eligible. (Sec. 7.3)
- **Post-auction Performance**: Duration required for a winning bid to be finalized and incorporated into the final signed beacon block. (Sec. 7.4)
- **Macrobenchmark**: Evaluation of timing for bid count and MEV when processing historical bid data. (Sec. 7.5)

7.1. Dataset and Configuration

The dataset we used is the full bids from Ultra Sound Relay [79], which records the historical bids during 2023/09-2023/12. That's the newest public dataset as far as we know. After December 2023, Ultra Sound Relay stopped collecting full bid data due to privacy concerns. We randomly chose 40,000 slots during that time window for our analysis. Among the 40,000 slots, we compared them with the winning bids recorded on-chain [6] and got a winning-bid dataset with 28,107 winning bids. We also extended the winning-bid dataset to winning-builder dataset with 117,313 bids, which not only includes the winning bids but also includes all the winners' submission.

We deployed our DPaaS on $n = 4$ CVMs across 4 availability zones provided by Amazon AWS and Microsoft Azure. Fig. 3 shows the round trip times measured between different CVMs. Each CVM has 4 vCPUs and 16 GB of memory. We deployed a simulated BT in one CVM in AWS, with 8 vCPU and 32 GB memory. We choose $n = 4$ as the practical deployment configuration, given the limited number of provider availability zones supporting CVMs.

7.2. BT Overhead

BT could adds overhead to builder's bidding due to block validation, block package data preparation, and TEE attestation generation. To evaluate this overhead, we simulate builder submissions using the extended winner-bid

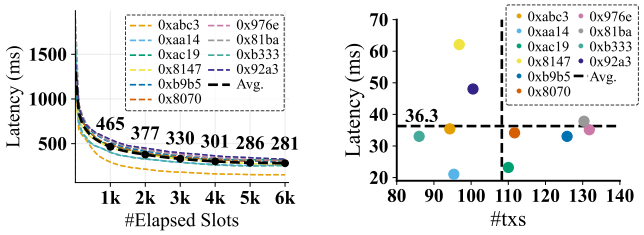

Figure 4: Analysis of validation latency across 10 builders, labeled by public key prefix, and 6,000 slots. (a) Latency for the first block in each slot. (b) Scatter plot of "warmed" latency for subsequent blocks in each slot versus number of transactions, averaging over all slots.

Block Validation	Data Preparation	TEE Attestation	Total
51.69 ± 20.72	9.16 ± 1.29	4.92 ± 0.17	65.80 ± 20.3

TABLE 1: BT latency (ms) breakdown.

dataset, which has on average 108 transactions and payload size of 57.19 KB. Table 1 summarizes the detailed latency breakdown observed in BT. Data preparation and TEE attestation generation are lightweight operations: the former is a function of the payload size, while the latter is constant. Block validation, taking 51.6 ms on average, is the most significant because it requires simulating the execution of all transactions in the block. We derive this average over 28,107 slots in a single BT; however, we observed significant variance, which we explore next.

Block Validation Details. We observe that the first block validation in each slot incurs substantially higher "cold-start" latency due to warming the in-memory EVM state cache. To quantify the builder-side impact, we sampled 10 active builders (i.e., bids in over 50% of slots) and collected their bids across 6,000 slots.

Fig. 4a shows the cold-start latency per slot: the first block a builder validates can exceed 1.5 s, but this cost drops quickly as the builder continues validating blocks. After 6,000 slots, the average cold-start latency falls to 281 ms. Importantly, these results reflect only the cold-start cost; all subsequent validations in the same slot benefit from a warmed cache. As shown in Fig. 4b, the average warmed validation latency across the 10 builders is just 36.3 ms.

This block-validation measurement shows that although cold-start latency is high, it can be effectively mitigated in two ways. First, running BT into auction for sufficient slots naturally amortizes the cost—after approximately 20 hours (6,000 slots), the overhead decreases by more than 78%. Second, a builder can proactively warm the in-memory cache each slot as it iteratively constructs the block.

7.3. Bid Processing Performance

We evaluate PE bid processing from the perspective of a single PE_i, from receiving a bid to placing it in the

Queue+ Deserialize	Int. Check	Verify Attestation		Overall
		TCB	Sig.	
1.48±0.03	0.84±0.54	0.12±0.07	2.25±1.91	4.83±1.26

TABLE 2: PE latency (ms) breakdown in unloaded setting.

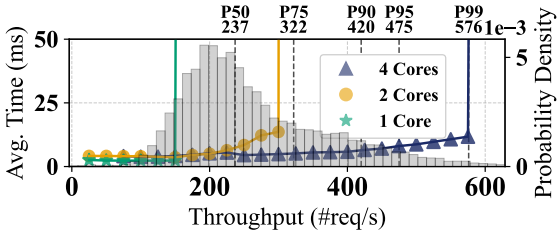

Figure 5: Analyzing saturation via average processing latency versus bid throughput while varying CPU cores. PDF of real-world peak throughput from full-bid dataset.

locally eligible set (i.e., E_i). We analyze the latency in an unloaded setting and then look at throughput saturation under load (as well as scalability with CPU resources). In both experiments, we use bids from the winning-bid dataset.

Unloaded. We set up a single builder that periodically submits bids in one-second intervals to a single PE_i. The results are in Table 2, broken down into key components. Average overall bid processing latency is 4.83ms, which is significantly lower than relays as a result of TEE attestations removing the need to simulate the block transactions [80]. Most of this comes from signature verification (*Sig.*) for the TEE attestation (2.25ms) as well as the RPC framework processing (*Queue+Deserialize*).*Int. Check* for checking the sanity and verifying integrity through TEE attestation and verifying the TCB (*TCB*) take little time.

Loaded. We deploy 20 builder threads on a single host, each periodically submitting bids to one PE_i to match target throughputs from 25 to 600 requests per second (step size 25). To assess DPaaS 's scalability, we vary the CPU allocation from 1 to 4 cores. For reference, we also plot the distribution of real-world bid arrival rates from the historical full-bid dataset. Bid submissions follow a steady-then-burst pattern: a moderate rate ($\leq$100 bids/s) for the first 9–10 seconds, followed by a sharp surge as builders race in the final seconds. Our analysis focuses on the peak-load region using 100 ms throughput windows.

We present the results in Figure 5. The saturation points for 1, 2, and 4 cores are 150, 300, and 575 (respectively). This means that, with just 4 cores, a PE can handle $\geq 99\%$ of historical windows within $\leq$ 10 ms per request. We emphasize that this represents a stress test using historical peak throughput under continuous submission. In practice, the load typically follows burst pattern at around $t{=}-4$ s to $t{=}0$ s . Interestingly, we observed that existing relays cannot sustain such bursty workloads under limited resources, motivating the design of optimistic relays [80] to mitigate critical-path latency. We will simulate real-world latency to DPaaS for comparison with relay in Sec. 7.5.

Case	Consensus	Recovery		Total
		Shares	Signing	
Case 1	41.44	13.85	14.31	55.75
Case 2	75.17	16.55	18.80	93.97
Case 3	153.92	16.21	18.41	172.33

TABLE 3: Average latency (ms) breakdown for finalization protocol. Shares and Signing take place in parallel, so Total only considers the highest in addition to Consensus.

7.4. Post-Auction Performance

After the auction is closed, the PEs need to agree on the winning bid, recover the plaintext block payload, and sign the block header. We generated eligible bid sets by randomly selecting 100 slots in 2023/12. We evaluate the latency in three different cases: 1) optimistic fast path, 2) fallback path with one follower crash, and 3) fallback path with first leader crash. We then input the 100 eligible bid sets to all n PEs under different cases. We explore all combinations of PEs for the initial leader and (if necessary) faulty party, running 100 trials per combination for each input bid set, to account for the differences in inter-PE RTTs (see Fig. 3). We set Δ to the 99th percentile of one-way latency measured over 10,000 samples between the PEs.

We present our results in Table 3. All measured values can be bounded by functions of Δ, though they are not directly comparable to it, since they are based on the actual (and unknown) latency $\delta < \Delta$. For instance, in some rounds PE can proceed without hearing from all n PEs, which means we don't need to wait for slower nodes.

In Case 1, the leader can construct $\mathcal{V}_Q$ having heard from all n PEs, leading to a three-round latency. In Case 2, due to a follower crash, the leader must use the fallback path and will wait for at most 2Δ prior to the VBB protocol; therefore, the upper bound on consensus in this case is 4Δ. In Case 3, the first leader crashed, so there will be a view change after a timeout of 4Δ. After the view change, it's guaranteed that new leader is honest and 2 more rounds are needed to commit. Therefore, the upper bound on consensus in this case is $4\Delta + 2\Delta + 2\Delta = 8\Delta$. Our implementation exploits the independence between recovering the secret from shares and signing the block header to parallelize these tasks; therefore, the total is equivalent to the consensus latency plus the maximum of either shares or signing latencies. Finally, we include a more-detailed, per-combination breakdown for the consensus latency in Appendix B.

7.5. 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 solely on the `ReceivedAt` timestamps in the dataset.

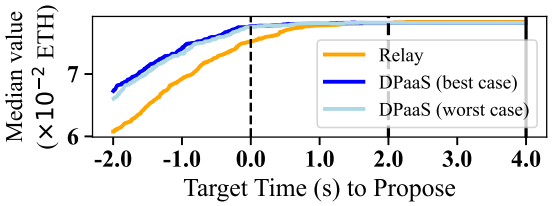

Figure 6: Median winning bid value based on different target proposal times (t_p) for DPaaS versus Relays.

Bid Processing Performance For each slot, we tracked the number of eligible and active (non-canceled eligible) bids throughout the slot for both DPaaS and the relay; for DPaaS, we used globally eligible and active bids to provide an equivalent comparison. We also kept track of upper bounds for each of these: the number of received bids at any time bounds the number of eligible bids, while the number of unique builders seen bounds the number of active bids. We found that DPaaS significantly improves both of these counts due to lightweight bid validation in the PEs; meanwhile, relays need to rely on optimistic mode (i.e., marking bids for select builders as eligible without validation) and selectively prioritizing other non-optimistic bids. As an example, when the receiving a median of 1,403 bids near the end of the auction in total, DPaaS attains 1,402 eligible bids, whereas the relay yields only 214. We refer readers to the detailed results in Appendix C.

Winner Bid Value However, higher volumn of eligible bids does not necessarily imply higher MEV values. Therefore, we 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 to determine t_e, we evaluated both the best case (4Δ) and worst case (9Δ since $f = 1$).

We present our results in Figure 6. At $t = 0$ s, which is the Ethereum-specified time for honest block proposal, DPaaS yields 0.0777 ETH (best case) and 0.0775 ETH (worst case), compared to 0.0752 ETH for the relay ($\geq 3\%$ improvements). Although both DPaaS and the relay converge after $t = 2$ s when no new bids arrive, such delayed proposals risk not receiving sufficient attestations [10] leading to a missed slot. The negligible gap between best-case and worst case latency bounds further shows DPaaS 's stable profit capture.

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.

8. Related Work

Replace Relay in MEV-Boost TEE-Boost [20] is the first proposal to eliminate the relay from the block-building pipeline, but it leaves several research challenges open. It handles only the block-validity side of fair exchange, while the tension between data availability and builder privacy remains unresolved. Based on TEE-Boost, a Paradign post [19] proposed to leverage a validator committee running threshold cryptography [22] to solve the conflicts between availability and privacy. But it is a complex change to the consensus layer for a mechanism that may or may not be adopted by the community. In contrast, DPaaS shows a backward compatible solution for current Ethereum and solves all fair exchange problems.

Enshrined PBS (ePBS) [18] has been proposed to internalize the existing sidecar mechanism (*i.e.*, MEV-Boost) directly into the protocol, thus removing the relays. However, ePBS has several limitations. First, ePBS requires builders to stake. This requirement disadvantages small builders, who are likely to incur negative profits if they must lock up 32 ETH. Second, ePBS may reduce throughput of auction by placing all interactions into the p2p network. As a result, proposers risk losing profitable opportunities during or near the end of the auction due to slower network communication. Third, ePBS restricts builder bidding strategies. Under ePBS, the auction operates as a single-bid open auction conducted through the peer-to-peer network, making bid cancellations and more sophisticated bidding strategies infeasible. Given these limitations, validators and small builders may still rely on MEV-Boost even in an ePBS world. DPaaS therefore remains complementary, providing a secure and efficient fair-exchange mechanism as a sidecar.

Decentralized Auction Franklin and Reiter [81] proposed the first cryptographic protocol for sealed-bid auction service for digital cash in fault tolerance manner. Besides that, SEAL [82], Riggs [83], Cicada [84], Cryptobazaar [85] are cryptographic protocols follows the lines of sealed-bid auction aiming at decentralized and auctioneer-free auction. Addax [86] was proposed as a sealed-bid auction service in online ad exchange.

In contrast, DPaaS is a distributed protocol that replaces trusted relays in PBS using lightweight cryptographic primitives. Our problem extends beyond running an open-bid, sealed-payload auction: DPaaS must remain backward-compatible with today's ecosystem so that it can serve as a practical drop-in replacement for centralized relays. Accordingly, DPaaS must offer builders efficient bidding with support for bid cancellation, while providing proposers with low post-auction latency and high-quality bid selection. However, previous works either tread bid cancellation as out of scope [81], [82], [84], [85], [86] or malicious behavior [83]. Moreover, DPaaS addresses the often-overlooked challenges of off-chain bidding and post-auction wrap-up in a partially synchronous network, whereas prior work operates entirely on-chain through smart contracts [82], [84].

9. Conclusion

Proposer-Builder Separation (PBS) is vital to Ethereum's goal of decentralization, yet the current MEV-Boost architecture introduces centralization and security risks through its reliance on trusted relays. In this paper, we presented Decentralized Proposer-as-a-Service (DPaaS), which distributes the combined roles of the proposer and relay across a set of Proposer Entities (PEs). We developed fault-tolerant protocols spanning the offline, auction, and post-auction phases, meeting fair-exchange goals while maintaining compatibility with Ethereum. Our evaluation demonstrates that DPaaS delivers strong performance, scalability, and ability to capture MEV in comparison to relays.

As part of future work, we plan to investigate the possibility of a solution that does not rely on the BT at the builder, which is the case for our work and several others [20], [57], [63], to meet the strict latency requirements.

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.

References

[1] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *IEEE Symposium on Security and Privacy*, 2020.

[2] Ethereum Foundation, "Maximal extractable value (MEV)," August 2025. [Online]. Available: http://ethereum.org/developers/docs/mev/

[3] V. Buterin, "Proposer/block builder separation-friendly fee market designs," June 2021. [Online]. Available: https://ethresear.ch/t/proposer-block-builder-separation-friendly-fee-market-designs/9725

[4] M. Bahrani, P. Garimidi, and T. Roughgarden, "Centralization in block-building and proposer-builder separation," in *Financial Cryptography and Data Security (FC)*, 2024.

[5] Ethereum Foundation, "Proposer-Builder Separation," https://ethereum.org/en/roadmap/pbs/, 2024, accessed: 2025-3-30.

[6] S. Yang, K. Nayak, and F. Zhang, "Decentralization of Ethereum's Builder Market," in *IEEE Symposium on Security and Privacy*, 2025.

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

[8] B. Adams and D. Feist, "EIP-7782: Reduce Block Latency," October 2024, ethereum Improvement Proposals, no. 7782. [Online]. Available: https://eips.ethereum.org/EIPS/eip-7782

[9] F. Wu, T. Thiery, S. Leonardos, and C. Ventre, "Strategic bidding wars in on-chain auctions," in *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2024.

[10] C. Schwarz-Schilling, F. Saleh, T. Thiery, J. Pan, N. Shah, and B. Monnot, "Time Is Money: Strategic Timing Games in Proof-Of-Stake Protocols," in *Conference on Advances in Financial Technologies, AFT*, 2023.

[11] V. Buterin, E. Conner, R. Dudley, M. Slipper, I. Norden, and A. Bakhta, "EIP-1559: Fee market change for ETH 1.0 chain," April 2019, ethereum Improvement Proposals, no. 1559. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1559

[12] M. Kalinin, D. Ryan, and V. Buterin, "EIP-3675: Upgrade consensus to Proof-of-Stake," July 2021, ethereum Improvement Proposals, no. 3675. [Online]. Available: https://eips.ethereum.org/EIPS/eip-3675

[13] M. Peterson, "Ethereum consensus shifts on EVM upgrade," https://blockworks.co/news/ethereum-consensus-evm-upgrade-fusaka-devs, 2025, accessed: 2025-10-20.

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

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

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

[17] T. Wahrstätter, "MEV-Boost Dashboard," https://mevboost.pics/, 2025, accessed: 2025-3-30.

[18] F. D'Amato, B. Monnot, M. Neuder, Potuz, and T. Tsao, "EIP-7732: Enshrined Proposer-Builder separation [DRAFT]," https://eips.ethereum.org/EIPS/eip-7732, June 2024, Ethereum Improvement Proposals.

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

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

[21] Flashbots, "Mev-boost risks and considerations," April 2025. [Online]. Available: https://docs.flashbots.net/flashbots-mev-boost/architecture-overview/risks

[22] S. Garg, D. Kolonelos, G.-V. Policharla, and M. Wang, "Threshold Encryption with Silent Setup," Cryptology ePrint Archive, Paper 2024/263, 2024. [Online]. Available: https://eprint.iacr.org/2024/263

[23] B. Vedartham, "Proposer Relay Interactions with ePBS," https://hackmd.io/@bchain/BJYoUYjIll, 2025, accessed: 2025-10-20.

[24] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[25] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[26] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and casper," *arXiv preprint arXiv:2003.03052*, 2020.

[27] Ethereum Foundation, "How to stake your ETH," https://ethereum.org/en/staking/, 2024, accessed: 2025-3-30.

[28] Ethereum, "The Beacon Chain," https://ethereum.org/roadmap/beacon-chain/, 2025, accessed: 2025-11-12.

[29] T. Thiery, "Empirical analysis of Builders' Behavioral Profiles (BBPs)," https://ethresear.ch/t/empirical-analysis-of-builders-behavioral-profiles-bbps/16327, Augest 2023.

[30] G. Konstantopoulos and M. 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.

[31] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, p. 86, 2016.

[32] Intel, "Intel trust domain extensions whitepaper," https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf, 2020, accessed: 2025-3-30.

[33] D. Kaplan, J. Powell, and T. Woller, "AMD SEV-SNP: Strengthening vm isolationwith integrity protection and more," *White paper, Advanced Micro Devices Inc*, 2020.

[34] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution," in *USENIX Security*, 2018.

[35] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *USENIX Workshop on Offensive Technologies, WOOT*, 2017.

[36] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proceedings of the 11th USENIX Conference on Offensive Technologies*, 2017.

[37] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. Lai, "SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution," in *IEEE European Symposium on Security and Privacy*, 2019.

[38] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX," in *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[39] W. Wang, M. Li, Y. Zhang, and Z. Lin, "Pwrleak: Exploiting power reporting interface for side-channel attacks on AMD SEV," in *Detection of Intrusions and Malware, and Vulnerability Assessment DIMVA*, 2023.

[40] B. Schlüter, S. Sridhara, A. Bertschi, and S. Shinde, "Wesee: Using malicious #vc interrupts to break AMD SEV-SNP," in *IEEE Symposium on Security and Privacy*, 2024.

[41] S. Gast, H. Weissteiner, R. L. Schröder, and D. Gruss, "Counterseveillance: Performance-counter attacks on AMD SEV-SNP," in *Network and Distributed System Security Symposium (NDSS)*, 2025.

[42] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, "SGAxe: How SGX fails in practice," https://sgaxe.com/, 2020.

[43] B. Schlüter and S. Shinde, "Rmpocalypse: How a catch-22 breaks amd sev-snp," in *ACM Conference on Computer and Communications Security (CCS)*, 2025.

[44] A. Shamir, "How to Share a Secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[45] G. R. Blakley, "Safeguarding cryptographic keys," in *International Workshop on Managing Requirements Knowledge, MARK*, 1979, pp. 313–318.

[46] J. von zur Gathen and J. Gerhard, "Fast polynomial evaluation and interpolation," in *Modern Computer Algebra*, 3rd ed. Cambridge, UK: Cambridge University Press, 2013, ch. 10.

[47] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," in *Advances in Cryptology - ASIACRYPT International Conference on the Theory and Application of Cryptology and Information Security*, 2001.

[48] J. Drake, "Pragmatic signature aggregation with BLS," https://ethresear.ch/t/pragmatic-signature-aggregation-with-bls/2105?u=benjamin ion, May 2018.

[49] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM J. Comput.*, vol. 12, no. 4, pp. 656–666, 1983.

[50] M. K. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart," in *ACM Conference on Computer and Communications Security (CCS)*, 1994.

[51] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case Latency of Byzantine Broadcast: a Complete Categorization," in *ACM Symposium on Principles of Distributed Computing (PODC)*, 2021.

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

[53] Flashbots, "MEV-Boost Relay," October 2025. [Online]. Available: https://github.com/flashbots/mev-boost-relay

[54] Paradigm, "Reth," August 2025. [Online]. Available: https://reth.rs/

[55] AWS, "Precision clock and time synchronization on your EC2 instance." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/set-time.html

[56] J. V. Bulck, F. Piessens, and R. Strackx, "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP*, 2017.

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

[58] Ethereum, "Keys in proof-of-stake ethereum," https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/keys/, 2025, accessed: 2025-07-01.

[59] T. 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.

[60] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *IEEE Symposium on Security and Privacy*, 2017.

[61] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," Cryptology ePrint Archive, Paper 2002/175, 2002. [Online]. Available: https://eprint.iacr.org/2002/175

[62] A. 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.

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

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

[65] Tokio, "Tokio - An asynchronous Rust runtime," October 2025. [Online]. Available: https://tokio.rs/

[66] VirTEE, "A Community for building Virt-based TEE." [Online]. Available: https://virtee.io/

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

[68] A. Sprenkels, "Shamir secret sharing library," October 2025. [Online]. Available: https://github.com/dsprenkels/sss

[69] ——, "Shamir secret sharing in Rust," October 2025. [Online]. Available: https://github.com/dsprenkels/sss-rs

[70] J. Grigg, "rage: Rust implementation of age," October 2025. [Online]. Available: https://github.com/str4d/rage

[71] Rust, "Crate sha2," October 2025. [Online]. Available: https://docs.rs/sha2/latest/sha2

[72] ——, "Crate tree_hash," October 2025. [Online]. Available: https://docs.rs/tree_hash/latest/tree_hash

[73] M. Azure, "Azure Instance Metadata Service," October 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/instance-metadata-service?tabs=windows

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

[75] Confidential Computing Consortium, "Unifying Remote Attestation Protocol Implementations," https://confidentialcomputing.io/2023/03/06/unifying-remote-attestation-protocol-implementations/, 2023.

[76] MaidSafe, "BLST Threshold Crypto (blsttc)," October 2025. [Online]. Available: https://github.com/maidsafe

[77] ARM, "Arm Confidential Compute Architecture," https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, 2025, accessed: 2025-3-30.

[78] AMD, "Versioned Chip Endorsement Key (VCEK) Certificate and KDS Interface Specification," https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/57230.pdf.

[79] Ultra Sound, "Ultra Sound Relay." [Online]. Available: https://relay.ultrasound.money/

[80] A. Chiplunkar and M. Neuder, "Optimistic relays and where to find them," https://frontier.tech/optimistic-relays-and-where-to-find-them, 2025, accessed: 2025-9-28.

[81] M. K. Franklin and M. K. Reiter, "The design and implementation of a secure auction service," in *IEEE Symposium on Security and Privacy*, 1995.

[82] S. Bag, F. Hao, S. F. Shahandashti, and I. G. Ray, "SEAL: Sealed-bid auction without auctioneers," Cryptology ePrint Archive, Paper 2019/1332, 2019. [Online]. Available: https://eprint.iacr.org/2019/1332

[83] N. Tyagi, A. Arun, C. Freitag, R. S. Wahby, J. Bonneau, and D. Mazières, "Riggs: Decentralized sealed-bid auctions," in *ACM Conference on Computer and Communications Security (CCS)*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds., 2023.

[84] N. Glaeser, I. A. Seres, M. Zhu, and J. Bonneau, "Cicada: A framework for private non-interactive on-chain auctions and voting," Cryptology ePrint Archive, Paper 2023/1473, 2023. [Online]. Available: https://eprint.iacr.org/2023/1473

[85] A. Novakovic, A. Kavousi, K. Gurkan, and P. Jovanovic, "Cryptobazaar: Private sealed-bid auctions at scale," *IACR Cryptol. ePrint Arch.*, p. 1410, 2024. [Online]. Available: https://eprint.iacr.org/2024/1410

[86] K. Zhong, Y. Ma, Y. Mao, and S. Angel, "Addax: A fast, private, and accountable ad exchange infrastructure," in *Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

Appendix A. Detailed Proof of Properties

First, we introduce the correctness of $(5f-1)$-psync-VBB protocol as three lemmas, which have been proved in [51], [52].

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

Lemma 2. ***Termination****: After GST, every* PE *eventually commits and terminates.*

Lemma 3. ***Good-case latency****: When the network is synchronous and the leader is honest, the proposal of the leader will be committed within 2 rounds.*

Theorem 1. *After GST, all honest* PE*s eventually agree on the same winning bid* w.

Proof sketch. Since each honest PE 's input to the $(5f-1)$-psync-VBB in Algorithm 5.1 is externally valid, the committed Val is also valid, because invalid proposal will not be voted for. By Lemma 1 and Lemma 2, all honest parties eventually commit the same externally valid and non-$\perp$ value. As $\mathcal{FW}$ is deterministic, all honest PEs derive the same winner w.

Theorem 2. *Failed Trade Privacy (Goal 4) and Block Validity (Goal 2) can always hold.*

Proof sketch. According to the Theorem 1, at least $n-f$ honest PEs will decide on the same winning block and only share the block's secret share. The faulty PEs (up to f) can never recover the block other than the winning block because of the hiding property of secret share. Therefore, Failed Trade Privacy holds. Due to the trust of the integrity of BT, the block has been simulated in the BT and proved by the TEE proof. Therefore, Block Validity holds.

Theorem 3. *Under synchrony, the Effective Cancellation property (Goal 5) always holds.*

Proof sketch. Since the builder $\mathfrak{b}$–PE network latency is bounded by Δ_{b2p} under synchrony and the bid processing latency can be bounded by Δ_{pr} , any latest bid from builder b sent before $t-\Delta_{b2p}-\Delta_{pr}$ to all PEs is guaranteed to appear in all honest PEs' locally eligible bid set. We use k_{max} to denote the sequence number of the most latest bid sent from builder $\mathfrak{b}$ before $t-\Delta_{b2p}-\Delta_{pr}$. According to the definition of active bid data set, in an honest PE_i, $\forall e \in E_i[b]$, if $e.\texttt{seq} < k_{max}, e \notin V_i[b]$.

If the leader's proposal is the $\mathcal{V}^n$ in the fast path. The k_{max}-th bid can make earlier bids not appear in at least $n-f$ PEs' V_i, as $n-f \geq 2f+1$, they can only agree on some bid $e : e.\texttt{seq} \geq k_{max}$ instead of the older bids. Otherwise, in non-fast path, at least $f+1$ PEs ' $\mathcal{V}$ must be included, ensuring that at least one is from an honest PE. Under synchrony, one honest PE has collected bid tuples V_i from all honest PE (at least $n-f$). Since $n-f \geq 2f+1$, if the k_{max}-th bid appears at at least $n-f$'s PE's view, it's guaranteed that they can only agree on some bid $e : e.\texttt{seq} \geq k_{max}$ instead of the older bids.

Consequently, the Effective Cancellation gives builders the clear guarantee that under synchrony those builders who submits their bids by broadcast before $t-\Delta_{b2p}-\Delta_{pr}$ can always ensure the finalization of PEs will only consider their most recent bid, but not the stale one that should be canceled. If the builder still submit bid with sequence number larger than k_{max} after $t-\Delta_{b2p}-\Delta_{pr}$, ensuring those bid will be considered and cancel prior ones is out of scope of the property, but only best-effort.

Theorem 4. *Under synchrony, the protocol Algorithm 5.1 guarantees Bid Selection (Goal 1).*

Proof sketch. At runtime, each PE_i exchanges their V_i to all other PEs. For an honest PE_i, the V_i it sends corresponds to their locally eligible set E_i, with V_i serving as a compressed representation given that all eligible bids from a builder b must have contiguous sequence numbers starting from 0. Byzantine PEs may arbitrarily deviate in terms of the V_i sent to different PEs.

After the initial exchanges, each PE_i has an input, either $\mathcal{V}^n$ or $\{\mathcal{V}_1, ...\mathcal{V}_{f+1}\}$, to the psync-VBB protocol; from Lemma 1 and Lemma 2, we know that all honest PEs will eventually agree on the same input (i.e., Val). Afterwards, in Algorithm 5.2, each PE_i computes $\mathcal{V}_c$, which it uses to directly compute the globally active set $\mathcal{A}$ by selecting the $(2f+1)$-th largest sequence number across all $V_i \in \mathcal{V}_c$ for each builder b. From $\mathcal{A}$, we can define the

corresponding globally eligible set $\mathcal{E}$, which contains bids of sequence number 0 through $\mathcal{A}[b].seq$ for each builder b (i.e., inverse of Definition 5.3). We now need to show for both cases of inputs that $\check{E} \subseteq \mathcal{E} \subseteq \hat{E}$.

In the first case, where Val $= \mathcal{V}^n$ from the fast path, then $\mathcal{V}_c$ contains V_i for all n PEs. $\check{E} \subseteq \mathcal{E}$ holds since all honest PEs contribute their V_i's and $n - f \geq 2f + 1$ with $n = 5f - 1$. $\mathcal{E} \subseteq \hat{E}$ holds since we know a bid $e \in \mathcal{E}$ must have appeared in at least $2f + 1$ V_i's, meaning that at least $f + 1$ honest PEs had it in their locally eligible set.

In the second case, where Val $= \{\mathcal{V}_1, ...\mathcal{V}_{f+1}\}$, we know that at least one $\mathcal{V}_i$ comes from an honest PE_i since at most f may be Byzantine. Under synchrony, the $\mathcal{V}_i$ must contain at least $n - f$ V_j's from the honest PEs because PE_i must have heard from all other honest PEs; this in turn means that $\mathcal{V}_c$ contains at least these V_j's. Both $\check{E} \subseteq \mathcal{E}$ and $\mathcal{E} \subseteq \hat{E}$ hold following logic from the first case now.

Finally, each PE_i determines the winning bid w by computing the exact function in the Bid Selection definition, using the set $\mathcal{A}$.

Latency For the remaining goal, we will look into the finalization and block recovery. We only consider synchronous network with upper bound Δ as the latency between honest PEs' communication. According to Lemma 3, $(5f - 1)$-psync-VBB can achieve 2-rounds good-case (the first leader is honest and GST$= 0$) latency. Considering the two rounds of exchange, at most 4 rounds are needed to commit the final winner; if all PEs are honest, this can be reduced to 3 rounds because of taking fast-path.

However, if the first designated leader is Byzantine, timeout is needed to view-change and rotate the leader. Note under synchrony, all honest PEs can have valid input to the $(5f - 1)$-psync-VBB after at most two rounds. Therefore, as long as an honest leader proposed after views, 2 rounds are left for the good-case latency from [51]. Therefore, the expected latency is: $2\Delta + 4\Delta \cdot f + 2\Delta = 4(1 + f)\Delta$.

Theorem 5. *Block Availability (Goal 3.1) always holds. Timely Block Availability (Goal 3.2) holds under synchrony at $t_p \geq t_e + 4\Delta + 4\Delta \cdot f + \delta_{rec}$, where δ_{rec} is actual latency for recovering Signed Beacon Block.*

Proof sketch. With the presumption that BT is trusted for integrity, the block is encrypted under K and K is secret shared correctly. Moreover, every PE can verify the decrypted shares' validity using correctly pre-computed hash (generated by BTs) of the shares. We only need to prove that there exists sufficient many honest PE can participate in the block reconstruction eventually after GST. In Algorithm 5.2, we set the threshold of counting a bid into set $\mathcal{A}$ for winner as $2f+1$. There exists a $(f + 1)$-subset of the $2f+1$ PEs choosing the bid must be honest and have the package to recover the block, thus ensuring Block Availability.

For Timely Block Availability, under synchrony, the finalization latency is bounded by $4(1 + f)\Delta$. Once the winner is committed, the PEs jointly reconstruct the Signed Beacon Block using Algorithm 5.3, with reconstruction latency δ_{rec}. The $\delta_{\text{rec}} \leq \Delta$ for most cases as we observed because the high concurrency of reconstruction of payload and interpolation of the signature. Thus, by setting the final block revelation deadline to $T + \delta_{\text{rec}}$ after the auction ends, timely block availability is guaranteed.

Leader ID \ Crashed ID	None	0	1	2	3
0	41.38±3.13	155.91±14.42	71.42±9.17	87.29±3.22	74.45±9.98
1	40.81±5.26	66.89±4.13	148.38±12.51	66.32±2.31	88.31±5.30
2	42.88±4.56	87.36±4.49	69.13±8.22	165.13±18.03	71.31±4.63
3	40.69±5.88	65.87±4.09	85.25±5.81	67.13±2.66	146.31±15.78

TABLE 4: Average latency (ms) ± std.dev for 4-psync-VBB under different combination of leader and crashed party choice when $\Delta = 23$ ms. **None** means no faulty PEs. All in millisecond (ms). Case 1: Every PE is honest; Case 2: First view follower crashed; Case 3: First view leader crashed, time out needed.

Appendix B. Post-Auction Performance: Consensus Details

As shown in Table 4, in case 1 referred to "fast path", the first leader can form $\mathcal{V}^n$ and input it to the VBB protocol and will then enjoy good-case latency, resulting in 3 rounds in total. In case 2, one of the followers in the first view crashed at the start, which can drive the protocol away from fast path. Therefore, first leader will wait for at most two Δ at the start then enter the VBB protocol with a normal-path proposal, whose upper bound is 4Δ. In case 3, the first leader crashed therefore a view change after a timeout of 4Δ and the second view can make the other honest PEs commit in good-case latency, 8Δ in total.

Appendix C. Macrobenchmark: Bid Count

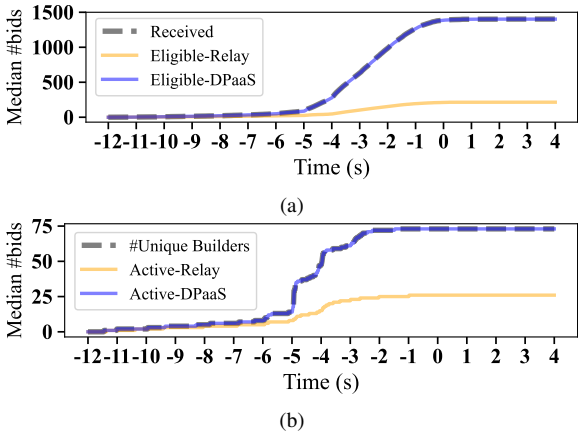

Figure 7: Median number of eligible (a) and active (b) bids throughout the auction timeline for a slot, comparing DPaaS and Relay against upper bounds (received bids and unique builders).

During the auction, both the count of eligible bids (#EB(t)) and active bids (#AB(t)) matters. For #EB(t), the closer it matches the number of received bids at time t, and for #AB(t), the closer it matches the number of active builders at time t, the more efficient the bid processing is. We measured #EB(t) and #AB(t) for both the relay (using historical bid traces) and DPaaS (by simulating the auction with the same traces). As shown in Fig. 7, the Ultra Sound Relay in December 2023 exhibited significantly lower #AB(t) and #EB(t) throughout the slot compared to DPaaS, which removes bid validation from the critical path. This discrepancy arises because Ultra Sound Relay had adopted the optimistic-v1 [80] mode since September 2023. In this mode, the relay fully simulates only bids from builders outside the allowlist (those without stake on the relay), while merely performing lightweight checks for allowlisted builders and marking their bids as eligible. Consequently, many bids remain queued for delayed simulation, and over 50 builders' bids are effectively ignored at certain times t.

In contrast, DPaaS substantially improves both #AB(t) and #EB(t), even in a decentralized setting where eligibility and activity require a threshold of at least $2f+1$ parties. This demonstrates that by integrating TEE-Boost into DPaaS, the scalability challenges faced by relays under heavy bid loads can be addressed—without resorting to optimistic modes or requiring builder to stake on the relay.