Secure Vickrey Auctions for Online Advertising

Archit Bhatnagar Yunming Xiao[†] Ang Chen Amrita Roy Chowdhury
University of Michigan [†]*The Chinese University of Hong Kong, Shenzhen*

Abstract

Online advertising is an essential part of the web ecosystem. When a user's browser lands on a webpage, advertisers bid for the ad space. An auction algorithm (e.g., Vickrey/second-price auction) is executed to determine the *winner* and the *price*; ideally, only this information is revealed, and everything else (i.e., the losing bids and the bidder identities) is kept private. However, achieving these privacy goals under a malicious security model, while operating under stringent performance requirements for online advertising, is challenging.

Obsidian enables secure Vickrey auctions for online advertising with three ideas: a new Multiparty Computation (MPC)-friendly encoding scheme that decouples bid values from bidders' identities; a novel use of function secret sharing to shift the cost of encoding validation to an offline phase; and a lightweight ring signature scheme to anonymously verify bidders. Obsidian outperforms generic MPC and homomorphic encryption approaches by orders of magnitude. Moreover, it also surpasses the state-of-the-art system, Addax, which is tailored to ad auctions under a weaker (covert) threat model and leaks more information.

1 Introduction

Online web advertising must meet privacy goals while satisfying stringent performance requirements at scale. Every page load triggers a real-time protocol that handles sensitive financial and personal data across a distributed set of parties, spanning browsers, publishers, ad exchanges, and advertisers, which must complete in hundreds of milliseconds. Online ads generate over $260 billion annually in the U.S. alone [43], making it the economic engine of the modern web, and powering the business models of major platforms, such as Google and Facebook. Real-time ad auctions are the backbone of this ecosystem. When a user visits a publisher's site, their browser initiates an auction via an ad exchange. The exchange shares user metadata with advertisers, collects bids, runs the auction, and delivers the winning ad—all within a fraction of a second.

Given the scale and financial stakes of online advertising, its privacy concerns are increasingly gaining attention:

Bid Confidentiality. A major risk from the bidder's (i.e., advertiser) perspective is the exposure of *bid values* [1], which can reveal sensitive business strategies. Advertisers' bid amounts often reflect proprietary information, such as targeting logic, campaign priorities, or real-time business decisions—data that can be exploited by a malicious ad-exchange. This risk is exacerbated in Vickrey auctions, which incentivize bidders to disclose their true valuations. These concerns are not merely theoretical: Google has been accused of leveraging insider access to past bid data to gain unfair advantages when its subsidiaries participated in auctions [44]; additional reports allege that Google pressured Facebook to abandon "header bidding"—a more open and decentralized bidding mechanism—by offering privileged data advantages that skewed the auction in Facebook's favor [2]. Google has also been found to "tune" the auction outcome to raise advertising revenues [13].

Bidder Anonymity. Another privacy concern of ad auctions is the exposure of *bidder identities*, which can inadvertently leak information about the user. Since advertisers are invited to participate in an auction based on a user's personal data—such as browsing history, demographics, and inferred interests—the very presence of a bidder reveals sensitive information about the user. For example, if a user is shown ads for experimental cancer treatments, one could reasonably infer that someone in their household is likely affected by the disease. Hence, bidder identities serve as a proxy for user profiling, and their concealment is key to better user privacy.

In response to these concerns [2, 28, 60], there has been a concerted push toward privacy-preserving ad auctions from both industry [9, 10, 11] and academia [65, 71, 72]. Google's recent FLEDGE/Protected Audience API (Section 2) sandboxes the entire auction within the browser, ensuring that sensitive data never leaves users' devices. However, this does not provide formal cryptographic guarantees and has been shown to be vulnerable to practical attacks [29]; placing the entire trust onto the browser also creates concerns, since a malicious browser could extract bid values/patterns and manipulate auction outcomes arbitrarily. The academic community has also studied cryptographic protocols for secure auctions in the context of secure multiparty computation (MPC), but most existing solutions are designed for generic auction settings and assume architectural features (e.g., direct communication among bidders or trusted third-party auctioneers) that are incompatible with today's web advertising infrastructure. The most relevant work to our setting is Addax [72], which specifically targets online ad auctions—it provides real-time performance but is limited to first-price auctions and only provides covert security; it does not address bidder anonymity.

Obsidian closes this gap by enabling secure Vickrey auctions with *both bid confidentiality and bidder anonymity*—even in the *malicious* threat model. It also delivers *real-time performance* while integrating seamlessly with today's ad infrastructure. In Obsidian, each bidder's value is secret-shared

[1] This corresponds to the setting of sealed-bid auctions.

between two parties—the browser and the ad exchange—which then jointly execute an MPC protocol for the auction [2].

A conceptual challenge in secure Vickrey auctions is the *decoupling* of bidder identities I and bid values V. In first-price auctions, the values and identities are coupled together, and once we sort the tuples $[V,I]$ by bid values, the first entry $[v_1,i_1]$ determines *both* the winner and their price. However, in Vickrey auctions, this entry only allows us to identify the winner's identity i_1; we still need to look at $[v_2,i_2]$ to determine the second-highest bid. Hence, the auction needs to maneuver carefully to avoid unwanted leakage—namely, v_1, which would reveal the winner's original bid, and i_2, which would reveal the closest contender (linking them to their bid v_2). Breaking the coupling between I and V to execute the auctions, and then securely linking them back to determine the winner and price, therefore, is key in devising a solution.

Obsidian introduces three new ideas. First, we propose a new Vickrey auction algorithm based on an encoding scheme that decouples bidder identities from their bid values. This algorithm is MPC-friendly, relying *only* on linear operations, amenable to efficient implementation under secret-sharing-based MPC. Second, while the above algorithm makes the actual computation of the auction more efficient, it introduces a new challenge in the malicious threat model—ensuring that submitted encodings are well-formed (i.e., syntactically valid). To resolve this, Obsidian's MPC protocol leverages a novel application of function secret sharing to shift the cost of validating encodings entirely to an offline phase, therefore achieving malicious security almost for free in the online phase. Finally, achieving bidder anonymity becomes more complex under the malicious threat model, as we must also ensure that all participants are legitimate advertisers. Without this, an unauthorized party could flood the auction with fake bids. Obsidian addresses this by proposing a lightweight ring signature scheme which allows each bidder to prove membership in an authorized set without revealing their exact identity.

We have implemented Obsidian and evaluated it against several baselines. Obsidian outperforms generic MPC (MASCOT [49] on MPSPDZ [48]) and homomorphic encryption (SEAL [8, 31]) solutions by orders of magnitude, under the same malicious security model. Compared to Addax [72], which leaks more information for Vickrey auctions[3] and assumes only a weaker covert security model, Obsidian improves the performance by 15.4% for auction latency and $4-10\times$ for ad-exchange throughput. These findings show that Obsidian is a practical solution that can offer strong privacy guarantees without compromising on performance.

2 Motivation

Fig. 1 illustrates the workflow of online advertising. When a *browser* (i.e., a user) visits a *publisher's* website (e.g., Facebook), it triggers three steps that ultimately result in an ad

[2]This model is known as the outsourced MPC paradigm [46]

[3]Leaks the value of the highest bid.

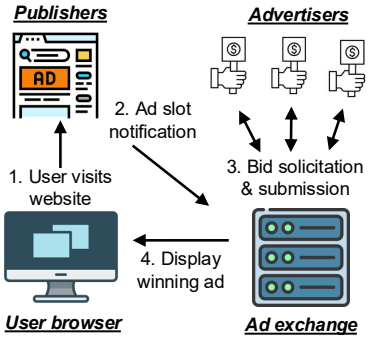


Figure 1: The architecture and workflow of online ads.

being displayed on the page. *1. Bid accumulation.* Upon the user's visit, the publisher contacts an *ad exchange* (e.g., run by Google), which forwards a bid request to a set of *advertisers* [4] (e.g., Adidas, Nike) for the available ad slot. Each advertiser computes a bid based on their strategy, and user information (e.g., browsing history, demographics, and inferred interests), which may be maintained by both the publisher and the advertiser depending on the tracking mechanisms. *2. Auction.* The exchange collects all bids and executes an auction to determine the winner—this is the heart of the entire workflow and the focus of Obsidian. Obsidian aims to provide a privacy-preserving solution for this stage, while remaining efficient enough for real-time deployment. *3. Ad delivery.* The ad exchange forwards the winner's ad tag to the publisher to render their page and subsequently bills the winning advertiser. Additional telemetry (e.g., click-through rates, conversions, and other engagement metrics) may be collected to analyze the performance of the ad campaign based on the user's interaction with the ad.

2.1 Need for privacy

In practice, today's ad exchanges like Google's Display & Video 360 and Meta's Audience Network collect plaintext bids from advertisers on a central server, enforcing bid confidentiality only through contractual terms of service. The OpenRTB protocol [4], which governs the majority of programmatic advertising today, transmits bid requests and responses as plaintext over TLS, treating the exchange as a fully trusted broker. These exchanges further share user data (like demographics and history) to invite bidders, thus both the bidder identities and their bids are known to these exchanges.

In response to growing regulatory and public scrutiny over how user data is handled in online advertising, the ad tech industry is giving browsers greater control over the above process. A prominent trend across several proposals—including Turtledove [3], Parrot [5], Dovekey [1] and the most recent Protected Audience API [11] (formerly known as FLEDGE)—is to run the entire ad auction within the user's browser. This grants users greater agency by allowing them

[4]Typically, a publisher is represented by a *supply side platform*; advertisers are represented by *demand-side platforms*, which operate ad servers and participate in real-time auctions on their behalf.

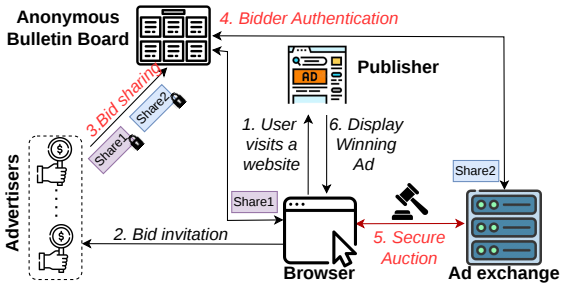


Figure 2: Steps covered by Obsidian are marked in red.

to control which bidders can participate, and hides bidder identities from both publishers and ad exchanges, thereby protecting users' privacy. However, this browser-centric model places complete trust in the user. Since the entire auction runs locally on the user's device, a malicious user can relatively easily extract the values of all private bids, violating bidder privacy. Worse still, a more devious user could arbitrarily tamper with the auction process, compromising its integrity. To mitigate these issues, other proposals such as Dovekey [1] and Ibex [71] instead outsource the auction to two or more non-colluding servers (e.g., the bidders themselves [39, 52]) using MPC. However, generic MPC approaches introduce significant performance overhead, limiting practical deployment.

2.2 Obsidian

Obsidian builds on the existing ad infrastructure with minimal modifications (Fig. 2), in the (2+1)-party setting which is an MPC model known for its efficiency [22, 68]. The protocol is divided into offline and online phases. The offline phase is independent of the input data (i.e., the bids) and can be executed at any time in advance, amortized across many auctions, since a single offline execution generates material for a batch of upcoming auctions and can be scheduled opportunistically during device idle time. The online phase performs the actual auction computation after the bids have been collected: the bidders' values are secret-shared between two parties—the browser and the ad exchange—who jointly execute an MPC protocol to run Vickrey auctions. An auxiliary server (i.e., "dealer") participates *only* in the offline phase to generate and distribute correlated randomness to both the browser and the exchange, but it is *not* involved in the online execution. As a result, our protocol is asymmetric, with the dealer performing a different set of tasks than the browser and the exchange. This design allows us to offload the computationally intensive work to the offline phase, maintaining high performance during the online phase. This setup places an overhead on the browser computing linear operations over secret shares in the online phase, while the offline phase can be run asynchronously. This is consistent with recent industry proposals shifting more computation to the browser for better privacy.

Existing services like Divvi Up [42][5] provide infrastructure for non-colluding auxiliary servers, and could fulfill the role of the dealer in practice. The offline phase can be scheduled during periods of device inactivity (e.g., overnight while charging), minimizing user disruption. It is also possible to eliminate the dealer entirely by using a separate MPC protocol between the browser and the exchange to generate the necessary randomness, albeit at a higher computational overhead during the offline phase.

Threat Model. Obsidian is designed to operate under the *malicious* threat model. During the *offline* phase, either (1) the dealer may be malicious, providing ill-formed correlated randomness that must be verified; or (2) one of the browser or ad exchange parties may be malicious. In this case, the entire MPC computation must be performed in a "dishonest majority" setting [22, 68] due to our asymmetric protocol design, since the dealer does not participate in the online phase. During the *online* phase, either the browser or the ad exchange may be corrupted and attempt to violate the privacy goals stated above. Additionally, any number of bidders may be malicious and collude with a corrupt browser or ad exchange.

Security Guarantees *Bid Confidentiality.* The value of every bid, except the second-highest one, remains private. This is inherent to Vickrey auction semantics, since the winner pays the second-highest price. Any winning party, by design, learns the price they must pay; this does not constitute a privacy leak beyond what the auction mechanism itself requires. *Bidder Anonymity.* Obsidian adheres to the privacy goals of recent in-browser auction proposals, ensuring that only the browser learns the identities of participating bidders. After the auction is executed, *only* the browser learns the identity of the winning bidder, but it does not learn the bid rankings of the losing bidders. Moreover, the ad exchange, publisher, and other bidders learn nothing about the identities of any bidders—not even the winner. As a result, all bidders remain completely anonymous to all parties except the browser.

Non-Goals. Obsidian focuses strictly on the privacy concerns that arise during the computation of the auction outcome—beginning from the moment bids are submitted and ending when a winner is selected. In particular, in the pre-auction phase: (1) how the bidders are decided upon and vetted, (2) how the bidders compute their bids (based on information about user profiles and internal business logic), are beyond the scope of Obsidian. Nevertheless, it is straightforward to address them by combining Obsidian with existing techniques for oblivious bidding, such as those proposed in Ibex [71]. Similarly, in the post-auction phase, (1) how the ad gets delivered and rendered on the webpage and (2) how the exchange privately bills the winning advertiser, are orthogonal to our design. These tasks can also be supported by integrating Obsidian with existing systems [30, 55, 66].

3 Vickrey Auction Algorithm

Challenge. A naïve approach to implementing Vickrey auctions in a two-party MPC setting is to securely (and partially) sort all bids, extract the top two values, and return the index of

[5] Operated by the non-profit Internet Security Research Group

the highest bid along with the second-highest value. However, this is computationally expensive because secure comparisons, being non-linear operations, are not amenable to linear secret-sharing schemes. Each secure comparison incurs significant computational and communication overheads, requiring multiple rounds of interaction [56, 57]. While garbled circuits handle such non-linear operations more efficiently, they incur a high communication cost that grows with the number of bidders. In practice, auctions must complete within hundreds of milliseconds to meet the stringent latency requirements of real-time bidding systems [7, 12]. Consequently, achieving both high throughput and low latency is critical, rendering both approaches ill-suited for practical deployment.

Our Approach. We address the problem by proposing a novel Vickrey auction algorithm specifically designed to be MPC-friendly. The algorithm relies *solely* on linear operations, making it compatible with secret-sharing-based MPC protocols. This is important as linear operations require no interaction between parties during the online phase, and all communication is a single round of share exchange. Network latency is minimized as the round-trip count remains constant regardless of the number of bidders or the bid domain size. For clarity, we first present the algorithm as it would run *in the clear*, and defer its secure implementation in MPC to the next section. We assume n bidders, each with a private bid b_i. Without loss of generality, bids are discrete and belong to the domain $[d] = \{1, \cdots, d\}$. We initially assume there are no ties, and later explain how ties can be handled.

Algorithm. Our algorithm works in three stages:

Encoding Bids. First, a bid value $b \in [d]$ is encoded as a binary vector with 1s followed by b-1 trailing 0s:

$$\mathrm{E}(b) = [\underbrace{1, \ldots, 1}_{d-(b-1)}, \underbrace{0, \ldots, 0}_{b-1}]$$

All bids form a matrix $\mathbf{b} \in \{0,1\}^{n \times d}$, where the row $\mathbf{b}[i][:]$ corresponds to the unary encoding $\mathrm{E}(b_i)$ of the i-th bidder.

Computing the Second-Highest Bid. We compute the column-wise sum of $\mathbf{b}$ and denote the j-th column as $c_j = \sum_{k=1}^{n} \mathbf{b}[k][j]$. Since $\mathbf{b}[k][j] = \mathrm{E}(b_k)[j] = 0, \forall j < b_k$, this corresponds to the number of bidders with bids at most j. Hence, the second-highest bid corresponds to the *smallest* j^* such that $c_{j^*} = n-1$. That is, $n-1$ bidders bid at most j^*, with at least one bidder placing a bid of exactly j^*.

Identifying the Winner. Finally, the winner has placed a bid strictly greater than j^*, making them the highest bidder. We can identify the winner as the unique row k^* in $\mathbf{b}$, such that, $\mathbf{b}[k^*][j^*] = \mathrm{E}(b_{k*})[j^*] = 0$, encoding a bid higher than j^*.

Example: The following table illustrates this for a simple example with four bidders and a bid domain of size 8. The column-wise sum rises to $n-1 = 3$ (where $n = 4$ is the number of bidders) at the column index of 5. Hence, $j^* = 5$ and the second-highest bid is 5. Among the bidders, only Bidder 3 has a 0 at this position, meaning that it has a higher bid and is identified as the winner.

Algorithm 1 Obsidian's Vickrey Auction Algorithm

Input: Bids $\{b_1, \cdots, b_n\}$ where each $b_i \in \{1, \ldots, d\}$
Output: Winner index k^* and second-highest bid j^*

(* Encode bids *)
for $k = 1$ *to* n **do**
  Encode b_k into bit vector $\mathrm{E}(b_k)$ of length $|d|$:
  $$\mathrm{E}(b_k)[i] = \begin{cases} 1 & \text{if } b_k \geq i \\ 0 & \text{otherwise} \end{cases}, \text{ for } i = 1, \ldots, |d|$$
(* Column-wise sum *)
for $j = 1$ *to* $|d|$ **do**
  $c_j = \sum_{k=1}^{n} \mathrm{E}(b_k)[j]$
(* Identify second-highest bid, may have ties *)
$T \leftarrow n-1$ // Initialize threshold to handle ties
while j^* *not found* **do**
  for $j = n$ *to* 1 **do**
    if $c_j = T$ **then**
      $j^* \leftarrow j$
      break
  $T \leftarrow T - 1$ // Decrease threshold
(* Identify winner *)
for $k = 1$ *to* n **do**
  if $\mathrm{E}(b_k)[j^*] = 0$ **then**
    Set $k^* \leftarrow k$ **break**

Resolving Ties. Next, we consider resolving ties when there are $t \geq 2$ bidders who have placed the highest bid (i.e., the maximum value among all bids), the *smallest* index j^* with $c_{j^*} = n - t$ corresponds to the second-highest value. The winner is now selected at random from all t bidders (rows) with a 0 in the j^*-th column.

Bidder	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Bidder 1 (bid=5)	1	1	1	1	0	0	0	0
Bidder 2 (bid=3)	1	1	1	1	1	1	0	0
Bidder 3 (bid=7)	1	1	0	0	0	0	0	0
Bidder 4 (bid=2)	1	1	1	1	1	1	1	0
Column Sum	4	4	3	3	2	2	1	0

The correctness of the algorithm is formalized by the following lemma, we include a proof for this in Appendix 1.

Lemma 1. *Algorithm 1 implements Vickrey auction.*

Moreover, this algorithm can be implemented using simple linear operations, and the structure of our encoding allows us to clearly disentangle the bid values (columns) from the bidder identities (rows). This separation is the key to efficiently implementing it under MPC, discussed next.

4 Obsidian Design

4.1 Building Blocks

Linear Secret Shares. LSS is an MPC technique that allows mutually-distrusting parties to securely compute over secret

inputs. We use $[x]$ to denote a linear secret sharing of an input $x \in \mathbb{F}$. Throughout, $\mathbb{F}$ denotes a finite field, and λ is the security parameter governing the computational security of the cryptographic primitives. Each party P_i holds a share $[x]_i$ such that $\sum_{i=1}^{k}[x]_i = x$. LSS supports local linear operations.

Authenticated Secret Shares. We use SPDZ-style [35, 50] Auth-Secret shares, which ensure the integrity of the shared secrets. This approach augments the secret shares with an additional sharing of an information-theoretic message authentication code (IT-MAC). Each party shares $[\gamma]$ for a secret MAC key $\gamma \in \mathbb{F}$, and for a shared value $[x]$, they also share the corresponding MAC $[\gamma \cdot x]$. We denote $\langle x \rangle = ([x], [\gamma \cdot x])$ to be the authenticated secret sharing for a secret x and $\langle x \rangle_i = ([x], [\gamma \cdot x]) \in \mathbb{F}^2$ as the authenticated secret share held by party P_i. ASS also supports local linear operations.

Function Secret Share (FSS). A two-party function secret sharing [23, 24] scheme splits a function f into succinct function shares such that individual shares provide no information about function f, yet when their evaluations are aggregated at any input point x, they reconstruct the correct output $f(x)$. A two-party FSS construction consists of a pair of algorithms:

- $K_1, K_2 \leftarrow \text{Gen}(1^\lambda, f)$: Takes as input the security parameter 1^λ and a function specification f, generates and returns two keys K_1 and K_2.
- $y_i \leftarrow \text{Eval}(K_i, x)$: Takes as input a key K_i and an evaluation point x, then outputs a value y_i representing the i-th party's share of $f(x)$.

The correctness property ensures that combining the evaluation outputs $y_1 + y_2 = f(x)$ for any input x. We focus on distributed point functions (DPFs), which implement FSS for point functions $f_{\alpha,\beta}$ where $f_{\alpha,\beta}(\alpha) = \beta$ and $f_{\alpha,\beta}(\alpha') = 0$ for all $\alpha' \neq \alpha$. We denote the DPF generator algorithm as $\text{Gen}(1^\lambda, \alpha, \beta)$, with evaluation performed using Eval. For $f_{\alpha,\beta}$, if $e = \text{Eval}(K_1, x) + (K_2, x)$ for x over the entire function domain $|\mathcal{D}|$, the expanded shares are of the one-hot form: $\mathbf{e} = (0, \ldots, 0, \beta, 0, \ldots, 0)$ where β appears at position α. The evaluated share for party i at x is then the i-th share of this one-hot encoding, i.e., $y_i(x)$ such that $y_1(x) + y_2(x) = f_{\alpha,\beta}(x)$ for all x. We use EvalAll to denote a full domain evaluations which can be performed more efficiently.

Verifiable Function Secret Sharing (VFSS). VFSS extends FSS by enabling parties to check that the shares are well-formed. Specifically, in the context of DPFs, it allows the parties to check that the expanded vector is indeed a valid one-hot encoding. This is typically done with the help of malicious sketching protocols [21, 68].

Commitment scheme A commitment scheme allows a sender to commit to a secret value and later open it in a verifiable way, such that a receiver can verify whether the revealed value is consistent with the committed one.

Signature scheme A digital signature scheme enables a signer to provide cryptographic authentication for a message, which a receiver can subsequently verify. The scheme is formally characterized by three algorithms:

- $(\text{sk}, \text{pk}) \xleftarrow{\$} \text{Sig.keyGen}(1^\lambda)$: Takes the security parameter 1^λ as input & generates a sign-verify key pair (sk, pk).
- $\sigma \xleftarrow{\$} \text{Sig.sign}(\text{sk}, m)$: Given a signing key sk and message m, produces a signature σ.
- $b \leftarrow \text{Sig.verify}(\text{pk}, \sigma, m)$: Takes a verification key pk, signature σ, and message m, then returns $b = 1$ if σ is a valid signature on m under pk, and $b = 0$ otherwise.

Ring signatures are a special type of signature, where a user adaptively defines a ring R of users (of which it is a member), and then generates a signature that can be verified as having been generated by some user in that ring (without revealing precisely which one; formal definition is in App. A.3).

Key Agreement Protocol. A key agreement protocol consists of a tuple of the following three algorithms:

- $(\text{pp}) \xleftarrow{\$} \text{KA.param}(1^\kappa)$. The parameter generation algorithm samples a set of public parameters pp with security parameter κ.
- $(\text{pk}, \text{sk}) \xleftarrow{\$} \text{KA.gen}(\text{pp})$. Key generation samples a public/secret key pair from the public parameters.
- $\text{sk}_{ij} \leftarrow \text{KA.agree}(\text{pk}_i, \text{sk}_j)$. The key agreement protocol receives a public key pk_i and a secret key sk_j as input and generates the shared key sk_{ij}.

Unless stated otherwise, we assume the following key generation algorithm: $(\text{sk}, g^{\text{sk}}) \leftarrow \text{KeyGen}(g, \mathbb{G})$, where g is a generator of a cyclic group $\mathbb{G}$ in which the Decisional Diffie-Hellman (DDH) assumption holds.

Anonymized Bulletin Board. We assume the availability of a public, anonymized bulletin board $\mathcal{U}$, accessible to all parties, as in prior work [20, 62, 63]. In practice, $\mathcal{U}$ can be implemented as an append-only log hosted at a public web address, allowing anyone to post messages anonymously. All parties have read and write access to $\mathcal{U}$, which serves as a broadcast mechanism [59].

Authenticated Encryption provides confidentiality and integrity guarantees for messages exchanged between two parties. It consists of a tuple of three algorithms as follows:

- $k \xleftarrow{\$} \mathsf{AE.gen}(1^\kappa)$ The key generation algorithm outputs a private key k where κ is the security parameter.
- $\bar{x} \xleftarrow{\$} \mathsf{AE.enc}(k, x)$. Encryption takes as input a key k and a message x, and outputs a ciphertext $\bar{x}$.
- $x \leftarrow \mathsf{AE.dec}(k, \bar{x})$. Decryption takes as input a ciphertext and a key and outputs either the original plaintext or a special error symbol $\perp$ on failure.

Pseudorandom Function. A pseudorandom function (PRF) is a deterministic function from a random seed and an input, such that the output is computationally indistinguishable from truly random one. Here we use the Dodis-Yampolskiy PRF [36] which is detailed in App. A.1.

Non-interactive Zero-Knowledge Argument of Knowledge. We use non-interactive zero knowledge argument ZK and argument of knowledge ZKPoK for a relation $\mathcal{R}$. For our purposes, we require the following properties from ZK and

ZKPoK: Perfect completeness; Computational Soundness; Computational Zero-Knowledge.

4.2 Workflow

Challenge. Consider the basic approach: each bidder encodes their bid using E, generates secret shares of the encoded vector, and sends these shares to the browser and the exchange to execute Algorithm 1. While this protocol is efficient, as it only involves linear operations, it still faces an important challenge in the malicious threat model. Specifically, a corrupt bidder could send a malformed encoding—that is, a vector that does not conform to the expected format (a contiguous sequence of 1s followed by 0s)—and can compromise the correctness of the protocol. Verifying that an encoding is well-formed involves detecting a single transition from 1 to 0. Naïvely performing this check requires secure multiplications to compare adjacent bits, which scales as $O(nd)$. Since each secure multiplication involves interaction between parties, the verification process incurs significant communication cost.

Our Approach. We address this challenge through a novel use of VFSS, which shifts the task of generating shares of the encoded bids from the bidders to the three computation parties – browser, exchange, and the dealer. Concretely, Obsidian makes two key observations. First, if a bidder with bid b generates VFSS keys for the DPF $f_{b,1}$–which outputs 1 only on input b and 0 elsewhere–and shares these keys with the browser $\mathcal{B}$ and the exchange $\mathcal{E}$, then the resulting outputs can be *locally* transformed into secret shares of $\mathrm{E}(b)$. These shares can then serve directly as inputs to the MPC protocol that securely executes Algorithm 1. Now this introduces the additional overhead of verifying that the FSS keys are well-formed (i.e., they encode a valid DPF). This is where our second observation comes into play: the verification process can be entirely offloaded to an offline phase with the help of the dealer $\mathcal{D}$. As a result, Obsidian incurs *no* additional overhead in the online phase for verifying the well-formedness of bids. By leveraging this novel usage of FSS, Obsidian achieves malicious security *(almost) for free*.

Detailed workflow. For ease of exposition, we begin with a relaxed setting where the exchange can learn the identities of the bidders just like the browser. Concretely, at the end of the protocol execution, both the browser and the exchange learn the identity of the winner and the price to charge, but nothing more. Importantly, the privacy of the losing bid values remains protected, and so does the bid ranking of all losing bidders. In Section 4.3, we extend the protocol to hide bidder identities from the exchange as well.

4.2.1 Setup Phase Obsidian has a one-time setup phase where all parties are initialized with the system-wide parameters, namely the security parameter λ, public parameters for the key agreement protocol $pp \xleftarrow{\$} \mathrm{KA.param}(1^\lambda)$, and a field $\mathbb{F}$. Let $\mathcal{V}^*$ denote the set of all legitimate bidders, i.e., advertisers who have registered with the exchange. In each auction,

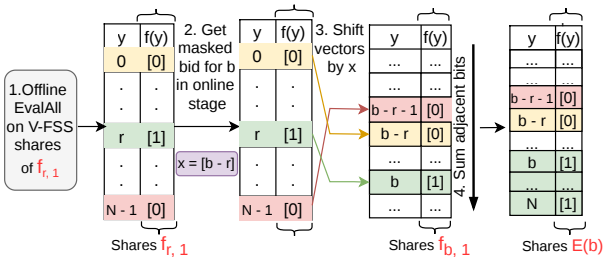


Figure 3: Workflow for using FSS to convert masked bid into shares of the encoded form of the bid.

the browser selects a subset of these bidders to participate. We assume that all the parties–browser $\mathcal{B}$, exchange $\mathcal{E}$ and all the bidders in $\mathcal{V}^*$–are authenticated with the help of standard public key infrastructure (PKI). This PKI enables the parties to register identities (public keys), and sign messages using their identity (associated secret keys), such that others can verify this signature, but cannot impersonate them [61]. Additionally, both the browser and the exchange establish a common secret key with each bidder $\mathcal{V}_i \in \mathcal{V}^*$ using the key agreement protocol, denoted by $\mathrm{sk}_{\mathcal{B},i}$ and $\mathrm{sk}_{\mathcal{E},i}$ respectively. Finally, we assume that the browser and exchange communicate over a private, authenticated channel. All parties also have both read and write access to the public bulletin board $\mathcal{U}$ via anonymous communication channels [37].

4.2.2 Offline Phase. In the offline phase, we leverage the dealer $\mathcal{D}$ to generate the FSS keys. Since the dealer does not know the actual bids (which are only revealed during the online phase), it generates FSS keys for a random DPF, i.e., $f_{r,1}$ for some random $r \in [d]$. These keys can later be converted to correspond to the DPFs of the actual bids during the online phase. We now describe how Obsidian securely verifies the correctness of these keys under the malicious threat model. Our design follows the general approach proposed by [68], which we briefly summarize. In this setting, two main challenges must be addressed:

- *Browser $\mathcal{B}$ or exchange $\mathcal{E}$ is malicious.* Given that the FSS keys are correctly generated by the dealer, the shares obtained by expanding these keys will later be converted into secret shares of the bid encodings during the online phase. Hence, the challenge is to ensure that $\mathcal{B}$ and $\mathcal{E}$ perform the correct computation over these expanded shares for the auction algorithm in the online phase. To address this, we require the expanded values to be authenticated secret-shares, i.e., shares augmented with SPDZ-style MACs (see Section 4.1). Since the DPF encodes a point function, the corresponding MAC shares also form a DPF. As a result, we can meet this requirement by having the dealer generate not only the FSS keys for the DPF $f_{r,1}$, but also additional FSS keys for the MAC shares $f_{r,\alpha}$, where α is a global MAC key.
- *Dealer $\mathcal{D}$ is malicious.* Here, we have to verify that the DPF keys and correlated randomness r generated by the dealer are well-formed. This corresponds to verifiable

FSS (see Section 4.1) and is enforced with the help of a malicious sketching scheme.

Concretely, in the offline phase the dealer $\mathcal{D}$ samples a random index $r \in [d]$ and a MAC key $\alpha \in \mathbb{F}$ and generates two pairs of DPF keys as:

$$(\mathsf{K}_r^{\mathcal{B}}, \mathsf{K}_r^{\mathcal{E}}) \xleftarrow{\$} \mathsf{FSS.gen}(1^\lambda, r, 1), (\mathsf{K}_\alpha^{\mathcal{B}}, \mathsf{K}_\alpha^{\mathcal{E}}) \xleftarrow{\$} \mathsf{FSS.gen}(1^\lambda, r, \alpha)$$

The dealer then sends to the browser $\mathcal{B}$ the keys $\{\mathsf{K}_r^{\mathcal{B}}, \mathsf{K}_\alpha^{\mathcal{B}}\}$, along with the linear secret shares $\{[r]_{\mathcal{B}}, [\alpha]_{\mathcal{B}}\}$. It sends the analogous keys and shares to the exchange $\mathcal{E}$. Next, the parties $\sigma \in \mathcal{B}, \mathcal{E}$ engage in a malicious sketching protocol which verifies the following properties:

- $\mathsf{K}_r^\sigma, \mathsf{K}_\alpha^\sigma$ form shares of well formed DPFs
- $[r]_\sigma$ form shares of the index of the DPFs
- The expanded shares $[y]_\sigma = \mathsf{FSS.evalAll}(\mathsf{K}_r)$ and $[m_y]_\sigma = \mathsf{FSS.evalAll}(\mathsf{K}_\alpha)$ satisfy $m_y = \alpha \cdot y$

The details of the sketching scheme are provided in Appendix A.4. Instead of using two separate keys, an optimization is encoding both the function & its MAC shares within a single DPF key by extending its payload. This effectively halves the cost of FSS.evalAll. The dealer generates n such keys for DPFs of the form $f_{(r_i,1)}, r_i \in [d], i \in [n]$ (and their corresponding MACs) where n is the upper bound on the number of bidders participating in the online phase of each auction. Additionally, it generates d DPFs over the domain $[n+1]$[6].

Lastly, the dealer $\mathcal{D}$ generates shared random bit strings with both the browser $(\{\nu_{\mathcal{B},i}\}_{i=1}^n)$ and the exchange $(\{\nu_{\mathcal{E},i}\}_{i=1}^n)$, each of length determined by the security parameter λ. It then publishes the commitments to the random masks used in the DPFs, specifically $\mathsf{H}(r_i||\nu_{\mathcal{B},i}||\nu_{\mathcal{E},i})$, on the bulletin board.

4.2.3 Online Phase We begin by considering the relaxed privacy setting, where both the browser $\mathcal{B}$ and the exchange $\mathcal{E}$ share the same view of the protocol. In this setting, both $\mathcal{B}$ and $\mathcal{E}$ are aware of the set of bidders $\mathcal{V} \subseteq \mathcal{V}^* = \mathcal{V}_1, \cdots, \mathcal{V}_n$ participating in the current auction. As noted in Section 2.2, the selection mechanism for $\mathcal{V}$ is out of scope for this work.

Recall that from the offline phase, $\mathcal{B}$ and $\mathcal{E}$ hold $n+1$ authenticated shares of one-hot vectors in $\{0,1\}^d$, denoted by $\langle y_i \rangle_{i=1}^{n+1}$. These are constructed by expanding the DPF keys for the functions $f_{(r_i,1)}$ and $f_{(r_i,\alpha)}$. Each share $\langle y_i \rangle$ consists of linear secret shares $[y_i]$ and their corresponding SPDZ-style MACs $[m_{y_i}]$ such that $m_{y_i} = \alpha \cdot y_i$.

We now describe how to use the shares $\langle y_i \rangle$ to obtain authenticated shares of a bidder $\mathcal{V}_i$'s bid b_i. For each bidder $\mathcal{V}_i$, the browser encrypts its share of the corresponding random index $[r_i]_{\mathcal{B}}$ and the bit string $\nu_{\mathcal{B},i}$ using an authenticated encryption scheme with their common secret key $\mathrm{sk}_{\mathcal{B},i}$ as $\mathsf{AE.enc}([r_i]_{\mathcal{B}}||\nu_{\mathcal{B},i}, \mathrm{sk}_{\mathcal{B},i})$ and posts the ciphertext to the public bulletin board $\mathcal{U}$. The exchange performs the same operation with its own share $[r_i]_{\mathcal{E}}$ and the bit string $\nu_{\mathcal{E},i}$ using the common secret key $\mathrm{sk}_{\mathcal{E},i}$.

Each bidder $\mathcal{V}_i$ retrieves both ciphertexts from the bulletin board and decrypts them using the respective shared keys to reconstruct the full mask r_i. The bidder then verifies that r_i matches the commitment com_{r_i} previously published by the dealer $\mathcal{D}$[7]. If the check fails, indicating that incorrect shares may have been submitted, the bidder aborts the protocol. If the check passes, the bidder computes their masked bid value as $a_i = b_i - r_i$. To further prevent the dealer from learning b_i (since the dealer knows r_i), the bidder encrypts a_i separately under the shared secret keys with the browser and the exchange. The bidder then publishes the resulting ciphertexts on the bulletin board.

Upon receiving the ciphertexts, both $\mathcal{B}$ and $\mathcal{E}$ decrypt their respective ciphertexts. Using the revealed value a_i, they can derive additive shares of a shifted vector $\langle \hat{y}_i \rangle$, defined as:

$$\langle \hat{y}_i[j] \rangle = \langle y_i[j + a_i] \bmod d \rangle$$

It is easy to see that this shifted vector corresponds to the shares of the DPF $f_{b_i,1}$. Next, the shares for $\langle \tilde{y}_i \rangle = \langle \mathrm{E}(b_i) \rangle$ can be constructed as

$$\langle \tilde{y}_i[1] \rangle = \langle \hat{y}_i[1] \rangle$$
$$\langle \tilde{y}_i[j] \rangle = \hat{y}_i[j-1] \rangle + \hat{y}_i[j], j \in [2, d].$$

The shares constructed this way are guaranteed to be well-formed (i.e., correspond to a valid encoding) since the FSS keys were verified in the offline phase. We further highlight the workflow for converting a masked bid to its encoded form using FSS keys expanded in the offline phase in Fig. 3.

From here, Obsidian assembles the matrix of bid encodings $\langle \mathbf{b} \rangle$ and proceeds to compute the auction algorithm 1. In particular, the browser and the exchange can now locally compute the shares of the column-wise sum as $\langle c_j \rangle = \sum_{k=1}^n \langle \mathbf{b}[k][j] \rangle$, and securely compute the column sums needed for the auction. However, two technical challenges must be addressed before we can proceed to computing the value of the second-highest bid. First, we have to determine whether there exists a tie. Second, we can reveal *only* the smallest index j such that $c_j = n - k$ where k is the number of bidders tied for the highest bid. Revealing anything else about the rest of the c_js leaks extra information. A straightforward approach is to evaluate the predicate $\langle c_j \rangle == n - 1$ *sequentially* for each j in ascending order until the condition is met. If no such index exists, this implies a tie at the highest bid, and the process can be repeated with the condition $\langle c_j \rangle == n - 2$, and so on. Note that in Vickrey auctions, the number of tied bidders and their identities are part of the output to be revealed.

[6] Again, without loss of generality, we assume n+1 to be a power of 2

[7] The appropriate commitment can be identified using a logical identifier embedded in the ciphertexts by both the browser and the exchange.

However, this naive method requires communication rounds that scale with the size of the bid domain, which is inefficient for latency. To address this, we again leverage FSS. In particular, we first convert $\langle c_j \rangle$ to shares of the DPF $f_{c_j,1}$, over the domain $[n]$. This can be done similarly to the strategy mentioned above, with a slight modification. Let $\langle y'_i \rangle_{i=1}^{d}$ be the shares of DPFs over the domain $[n]$ expanded in the offline phase. Now the browser and the exchange can directly perform an authenticated opening of $t_j = r'_j - c_j$ where r'_j is the random index of $\langle y'_j \rangle$. Using t_j, $\langle y'_j \rangle$ can be shifted to form the shares of $f_{c_j,1}$ as before. Let this resulting share be denoted by the one-hot vector $\langle \tilde{y'}_j \rangle$. For ease of exposition, assume there is no tie at the highest bid. Then there must exist some j such that $c_j = n-1$, and at least one of the vectors $\tilde{y}_j$ will have its $(n-1)^{th}$ entry set to 1, ensuring

$$1 + \sum_{j=1}^{d} \tilde{y'}_j[n-1] > 1.$$

Instead of computing this inequality directly over secret shares—which would be expensive, as it involves non-linear operations—we again employ FSS to perform this check efficiently. We treat this sum as a lookup input to a function represented via FSS: if the sum is 1, the function returns 1; otherwise, it returns 0. We convert $\sum_{j=1}^{d} \langle \tilde{y'}_j[n-1] \rangle$ to the shares of its one-hot vector representation using FSS as above. Let this be denoted by $\langle h \rangle$. Now, for the above equation to hold, we must have $h[1] = 0$; the browser and the exchange can reconstruct $\langle h[1] \rangle$[8] and verify this.

Once the absence of a tie is confirmed, we proceed to solve the second challenge: identifying the smallest index j for which $c_j = n-1$. Since the column-wise sums $\{c_j\}_{j=1}^{d}$ are monotonically increasing, this index can be determined efficiently by testing whether $c_j \geq n-1$ for each j. Unlike the naive sequential approach, this test can be performed *in parallel* across all indices. The smallest index satisfying the condition corresponds to the second-highest bid. This test can be implemented using $\tilde{y'}_j$ obtained earlier. In particular, for each $j \in [d]$, we evaluate

$$\tilde{y'}_j[n-1] + \tilde{y'}_j[n] = 1$$

The smallest index for which this holds is the desired one. Once this index is found, the corresponding column of $\langle \mathbf{b} \rangle$ can be opened to identify the unique row where the value is 0, which corresponds to the winning bidder.

If a tie is detected—i.e., if $h[1] = 1$ from the FSS-based tie detection step—we repeat the above process by checking whether there exists some j such that $c_j = n-2$, and so on, until the appropriate value is found. To further improve efficiency, rather than checking all values linearly, we can adopt a binary search approach over the possible values of $c_j \in 0, \ldots, n-1$. Given k tied for the winner, we can change the previous equation to $\sum_{l=0}^{k} \tilde{y'}_j[n-l+1] = 1$.

[8] Before reconstruction, all the shares are re-randomized.

4.3 Extension for Anonymizing Bidders

Challenge. We now describe extending the above solution to provide anonymity for the bidders with respect to the exchange. In this setting, the browser $\mathcal{B}$ is solely responsible for selecting the set of active bidders $\mathcal{V} \subseteq \mathcal{V}^*$ and is the *only* party that knows the identities of the bidders in $\mathcal{V}$ (as a set). Introducing anonymity, however, raises challenges in maintaining bidder credibility. Without safeguards, malicious actors **could** impersonate legitimate bidders or submit multiple bids in the same auction. To prevent this, we must ensure:

- The exchange can verify that each bidder is legitimate, i.e., in $\mathcal{V}^*$, via an anonymous credential check.
- The exchange can enforce rate limits, so that each bidder can submit at most one bid per auction round.
- The browser can validate that parties submitting a bid are members of the selected set $\mathcal{V}$ and identify them.

While existing cryptographic primitives can address each of these requirements in isolation, designing a unified protocol that satisfies all three simultaneously—while remaining practically efficient—remains an open challenge. In particular, the key challenge—and what makes our setting unique—is that validation is split across two different parties: the exchange must check global legitimacy (membership to $\mathcal{V}^*$) and enforce rate limits. At the same time, the browser must validate membership in the selected subset $\mathcal{V}$. Existing anonymous credential systems are not designed to accommodate this split trust model.

Our Approach. Obsidian co-designs both the system architecture and cryptographic protocols. In particular, we design a lightweight cryptographic protocol in which all three parties—the browser, exchange, and bidders—communicate exclusively via the public, anonymous bulletin board $\mathcal{U}$. Our core idea is a novel ring signature scheme that allows bidders to anonymously prove membership in $\mathcal{V}^*$. This requires a few additional steps at the beginning of the protocol to obtain the masked bids $b_i = r_i - s_i$ from the bidders in an anonymized manner. The rest of the protocol is exactly the same as in Section 4.2.3.

Our protocol begins after the browser $\mathcal{B}$ has selected the set of bidders, denoted by $\mathcal{V} = \mathsf{pk}_1, \cdots, \mathsf{pk}_n$, and notified them through a secure channel. The browser then publishes commitments to each bidder on the public bulletin board in the form $\mathsf{H}(\mathsf{pk}_i | \rho_i)$, where each ρ_i is a random bit string of length determined by the security parameter λ. Additionally, the exchange assigns a unique auction identifier ψ to every auction and publicly announces it.

Anonymous Credential Check. This task has two main objectives. First, the exchange must be convinced that it is interacting with a legitimate bidder, i.e., someone in $\mathcal{V}^*$, without learning their specific identity. Second, the exchange

must establish a session key with the bidder for this auction to securely transmit private information. In particular, the exchange needs to send its share of $[r_i]_{\mathcal{E}}$ to the anonymous bidder and, in turn, obtain the masked bid $r_i - b_i$, which is necessary for the bidder to compute masked bid values used in forming the final secret shares, as described in Section 4.2.3. These objectives can be accomplished as follows. Let R be the set of public keys of all members in $\mathcal{V}^*$, i.e., $R = \{\text{pk}_i \mid \mathcal{V}_i \in \mathcal{V}^*\}$. Suppose a bidder generates a fresh ephemeral key pair $(\text{pk}', \text{sk}') \xleftarrow{\$} \text{KeyGen}(\text{pp})$, and signs pk' using a ring signature w.r.t R, i.e., $\sigma \leftarrow \text{RS.Sign}(R, \text{sk}_i, \text{pk}')$. The bidder then sends (pk', σ) to the exchange. Upon receiving and verifying the ring signature, the exchange computes a common secret key for securely communicating with the anonymous bidder $\mathcal{V}_i$.

Next, we elaborate on the construction of such a ring signature, which is similar in spirit to the one in [47]. For ease of exposition, we describe it only in the context relevant to our setting, with a formal definition in Appendix A.3. The bidder $\mathcal{V}_i$ first computes a PRF evaluation on pk' using their (static) secret key sk_i, as $\gamma = \mathcal{F}(\text{sk}_i, \mathsf{H}(\text{pk}'))$. Then, the bidder $\mathcal{V}_i$ issues a commitment to their (static) public key, denoted com_{pk_i}, and then constructs the following two zero-knowledge proofs: proof of set-membership in R:

$$\pi_1 = \mathsf{ZK}\{\text{Open}(\text{com}_{\text{pk}_i}) \in R\}$$

and proof of correct PRF evaluation:

$$\pi_2 = \mathsf{ZKPoK}\{\text{sk}_i | y = \mathcal{F}(\text{sk}_i, \mathsf{H}(\text{pk}')), \text{Open}(\text{com}_{\text{pk}_i}) = g^{\text{sk}_i}\}$$

We adopt a specific PRF proposed by Dodis and Yampolskiy [36] as its algebraic structure facilitates efficient composition with ZKPs. This PRF supports efficient verification of evaluations using bilinear maps, so the above proof can be reduced to proving the correctness of a single pairing check:

$$\pi_2 = \mathsf{ZK}\{p = \text{Open}(\text{com}_{\text{pk}_i}), e(g^{\mathsf{H}(\text{pk}')}p, \gamma) = e(g,g)\}$$

Thus, π_2 serves as a signature on pk' by the party with the secret key sk_i—i.e., the legitimate owner of pk_i. Combined with the proof of set membership, π_1, which shows that pk_i is one of the valid public keys in R, the tuple $\sigma = \{\text{com}_{pk_i}, \text{pk}', \pi_1, \pi_2\}$ constitutes the desired ring signature on the message pk'.

Anonymous Rate-Throttling. The next challenge is to enforce rate-limiting, ensuring that each bidder can submit at most one bid per auction, without compromising anonymity. For this, we associate each bidder with a unique token for each auction. Each bidder computes a per-auction token as $\tau = \mathsf{H}(\text{sk}_i \,\|\, \psi)$ using there , (static) secret key sk_i. Next, they compute a ZKP of the correct computation:

$$\pi_3 = \mathsf{ZKPoK}\left\{\text{sk}_i \,\middle|\, \tau = \mathsf{H}(\text{sk}_i \,\|\, \psi),\ \text{Open}(\text{com}_{\text{pk}_i}) = g^{\text{sk}_i}\right\}$$

The tuple $(\text{com}_{\text{pk}_i}, \tau, \pi_3)$ proves that the token was correctly generated by the owner of pk_i, and enables the exchange to rate-limit by checking for duplicate tokens.

Authentication. Finally, the browser $\mathcal{B}$ needs to authenticate and identify a bidder, i.e., verify that they are in the set of bidders selected for this round $\mathcal{V}_i \in \mathcal{V}$. To achieve this, each bidder encrypts the opening of their commitment under the secret key $\text{sk}_{\mathcal{B},i}$ shared with the browser: $\bar{s} \leftarrow \text{AE.enc}(\text{sk}_{\mathcal{B},i}, \text{Open}(\text{com}_{\text{pk}_i}))$ and publishes (c, m) on $\mathcal{U}$. The browser $\mathcal{B}$ then decrypts $\bar{s}$ using $\text{sk}_{\mathcal{B},i}$ to recover the opening of com_{pk_i} and verifies that the opening is correct. This establishes a link between the messages published (Eq. 1) by the bidder and their identity, which is known only to the browser.

In summary, each bidder publishes the following tuple

$$(l_i, \text{com}_{\text{pk}_i}, \text{pk}', \gamma, \tau, \bar{s}, \pi_1, \pi_2, \pi_3) \tag{1}$$

on the anonymous bulletin board $\mathcal{U}$, where l_i[9] is a per-auction identifier assigned by the browser when notifying the bidder of their selection.

Upon reading this tuple from $\mathcal{U}$, the browser $\mathcal{B}$ decrypts $\bar{s}$ using $\text{sk}_{\mathcal{B},i}$ to recover the opening of com_{pk_i}, verifies it against the commitment, and then checks the validity of ZKPs. If all checks pass, it publishes 1) a signed message $(\text{msg} = \texttt{OK}, \sigma(\text{msg}, \text{sk}_{\mathcal{B}}))$ on $\mathcal{U}$, signaling to the exchange that the bidder is valid; and 2) the ciphertext of its share of the random mask and the corresponding bit string $\text{AE.Enc}(\text{sk}_{\mathcal{B},i}, [r_i]_{\mathcal{B}} || \nu_{\mathcal{B},i})$ using the secret key common with $\mathcal{V}_i$.

Next, the exchange verifies all ZKPs in the bidder's tuple. Upon successful verification, it generates an ephemeral key pair $(\text{sk}'', \text{pk}'')$, performs a key agreement protocol using the bidder's ephemeral public key pk', and publishes its random mask's share along with the corresponding bit string $\nu_{\mathcal{E},i}$, encrypted under the derived session key. It also includes a signed message containing pk'' to ensure authenticity.

The bidder $\mathcal{V}_i$ then decrypts both shares, $[r_i]_{\mathcal{B}}$ and $[r_i]_{\mathcal{E}}$, reconstructs the full mask r_i, and verifies its consistency with the commitment previously published by the dealer. Upon successful verification, the bidder computes the masked bid value $a_i = b_i - r_i$. This masked bid is then encrypted under both the session key with the exchange and the shared key with the browser, and published on the bulletin board. The remainder of the protocol proceeds exactly as in Section 4.2.3. Ultimately, only the browser learns the identity of the winner, as it is the only party that knows the public keys of all the bidders. To summarize, Fig. 9 and Fig. 10 illustrate the formal workflow of the online phase with this extension and the offline phase, respectively.

[9] This logical ID is a computational optimization that helps the browser efficiently identify which key to use for decryption, instead of iterating over all secret keys in $\mathcal{V}$. It has no bearing on the cryptographic security.

Note. We envision that bidder authentication need not be performed in every single auction. Instead, the browser and the exchange could randomly select a subset of auctions to perform the full authentication process. This selective, randomized verification reduces overhead while still providing a meaningful deterrence against misbehavior.

4.4 Discussion and Security Analysis

Efficiency. Obsidian uses an encoding scheme that decouples bids from bidder identities. The row-wise and column-wise computation precisely identifies the winning bid and the winner, without revealing additional information, and only linear operations are involved. The well-formedness checks can be entirely offloaded to the offline phase, so Obsidian achieves malicious security (almost) [10] for free in the online stage.
Constant Round Communication. Obsidian incurs a constant-round communication, *independent* of both the bid domain size and the number of bidders. This is enabled by the embarrassingly parallel structure of the computation.
General-purpose Algorithm. Our core algorithm can be easily modified to identify the k-th highest value, while providing the same privacy guarantees.

We use the simulation paradigm [61] to prove Obsidian's security guarantees. The ideal functionality (Fig. 12), formal theorem (Thm. 3), and full proof are in App. A.5.

Theorem 1. *Obsidian is secure against a static adversary that can corrupt at most one among the browser, the exchange, or the dealer, and any number of bidders, guaranteeing:*

- ***Correctness***. *Only the set of valid bidders selected by the browser can participate in the auction.*
- ***Privacy of bid values***. *All parties learn only the second-highest bid value, and nothing else.*
- ***Privacy of bidder identity***. *Only the browser learns the identity of the highest bidder; all other parties learn nothing about bidder identities.*

5 Evaluation

Our evaluation answers the following questions: a) How does Obsidian perform at different stages of the auction, and how well does it scale with the number of bidders and bid domains? b) How well does Obsidian perform against other baseline systems? c) What are the auction throughputs and response latencies that Obsidian can achieve?

We have implemented Obsidian in 1500+ lines of Rust code; our implementation and baseline evaluations are open-sourced and available at github.com/archit-bhatnagar/obsidian-nsdi-ae. For the ZKPs, we use the commit-and-prove paradigm and use Legosnark [40], Groth-Sahai proof system [41] and the ZK set-membership proof from [18]. The typical number of bidders for an auction is between 30 and 50 [15, 69], but we test a wider range from 25 to 100 bidders. The bid prices for online ad auctions typically vary between cents to a dollar [70]; therefore, we benchmark with bid domains of size up to 10,000 (i.e., 13-bit wide bids), which can well represent this range with fine step granularity. For the comparison over WAN, we use TCP sockets to connect the different parties and use `netem` to control the network latency. The timing measurements start when bids are received and end when the auction completes. All experiments are performed on an Intel Xeon Silver 4310 server with 48 cores.

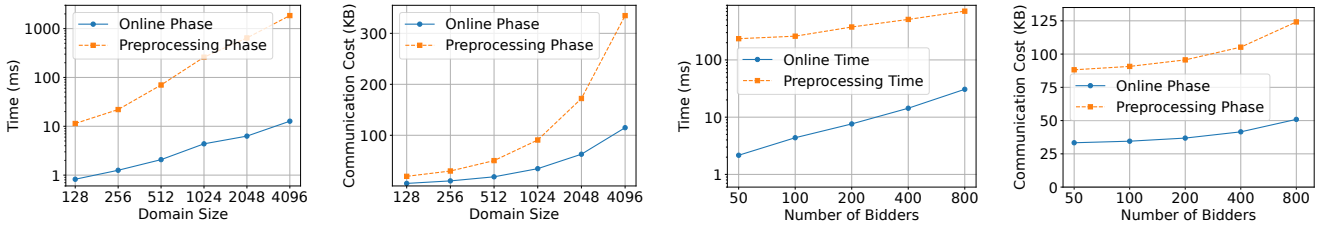
5.1 Microbenchmarks

We benchmark the three stages of Obsidian: offline precomputation, online auction, and credential verification. These benchmarks set the network latency to zero to measure the scalability of the system, along two dimensions: a) the number of bidders (from 50 to 800), and b) the size of the domain (from 128 to 4096). We double the step size along each dimension for experiments, run 1000 trials for each benchmark with different bids, and report the average. When scaling the domain size, we set the number of bidders to be 100; when scaling the number of bidders, we set the domain size to 1024.
Offline stage. The orange, dotted lines in Fig. 4 show the precomputation time (we parallelize DPF evaluations) and the communication overheads in the offline stage. We see that both computation and communication overheads of this stage scale linearly with the domain size (note that both axes are log scale), because the cost is dominated by DPF evaluations, which scale linearly with the domain size. Similarly, the number of DPF evaluations scales linearly with the number of bidders; hence the computation time scales linearly (Fig. 4c).
Online auction. The blue, solid lines in Fig. 4 shows the computation and communication overheads of the online auction stage—both scale roughly linearly across the number of bidders and bid domains. This is expected since, as discussed in Section 4.2.3, the size and the number of DPF shares that Obsidian needs to compute and communicate are linearly dependent on the number of bidders and the domain size.
Credential verification. We now benchmark the credential verification extension, by measuring the total message generation times for the bidders, browser, and the exchange. Fig. 8c shows the results. To measure the overheads at the bidder, we include the time for each bidder to generate the three proofs (as discussed in Section 4.3), and the setup costs for generating these proofs (e.g., commitments, encryption, hashing, and PRF computation). For the browser and the exchange, we benchmark the overheads incurred per bidder, which are the verification time of the three proofs. We found that the verification adds an overhead of only around 270ms, which is low enough to be practical. The browser and the exchange can perform batched verification for further improvement. As stated in Section 4.3, we envision that in practice, this verification will be performed at random instead of for all auctions.

5.2 Comparison Against Baseline Solutions

Next, we compare Obsidian's performance against the baseline systems, where auctions occur through TCP sockets over

[10] The only additional cost over a semi-honest protocol is that the computation is done over authenticated secret-shares instead of linear secret-shares.



(a) Compute latency with different domain sizes (100 bidders).
(b) Comm. cost with different domain sizes (100 bidders).
(c) Compute latency with different #bidders (domain: 1024).
(d) Comm. cost with different #bidders (domain: 1024).

Figure 4: Microbenchmarks of Obsidian for the offline and online auction computation stages (without credential verification). We test its scalability in two dimensions: the number of bidders (from 50 to 800), and the sizes of the domains (from 128 to 4096). We double the step size along each dimension and repeat each measurement 1000 times and report the average. Furthermore, we fix the #bidders when scaling the domain size and vice versa. Across all cases, the online auction time is less than 35ms, which is fast enough to be deployed in a practical online advertising setting.

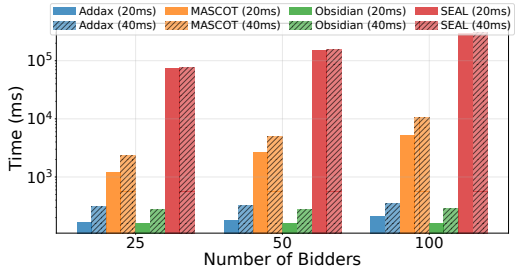


Figure 5: Auction latency across different #bidders

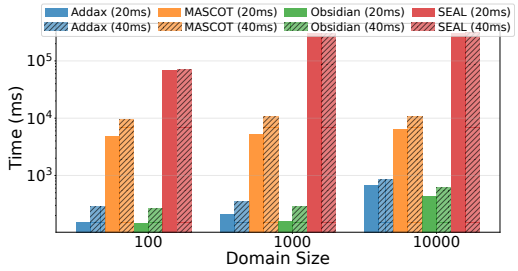


Figure 6: Auction latency across different domain sizes

a `netem`-based network with various WAN latencies set by this tool. We measure the latencies starting from the time when the bids are received until the auction computation (without credential verification for Obsidian).

We use the following baseline systems. (i) A recent auction system, Addax [72], which operates under a weaker (i.e., covert) threat model and leaks additional information under Vickrey auctions (leaks the highest bid value as well). (ii) An MPC framework, MPSPDZ [48], running an efficient sorting algorithm using MASCOT [49] with the optimal performance setting (with 4 threads, based on our measurements). (iii) A homomorphic encryption method, SEAL [8], running the top-k maxID algorithm [32]. For all systems, we compared the latencies for performing Vickrey auctions for the same setting (i.e., domain size, #bidders).

Fig. 5 shows the latency of these systems for different numbers of bidders with a fixed domain size, and Fig. 6 shows the latency for different domain sizes for a fixed number of bidders. In both cases, we can see that Obsidian outperforms

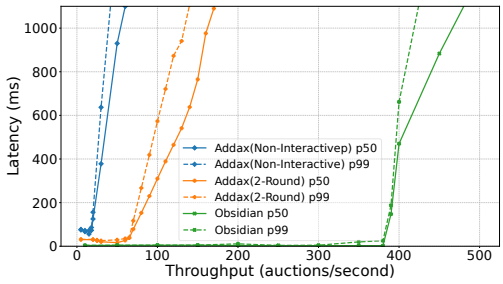
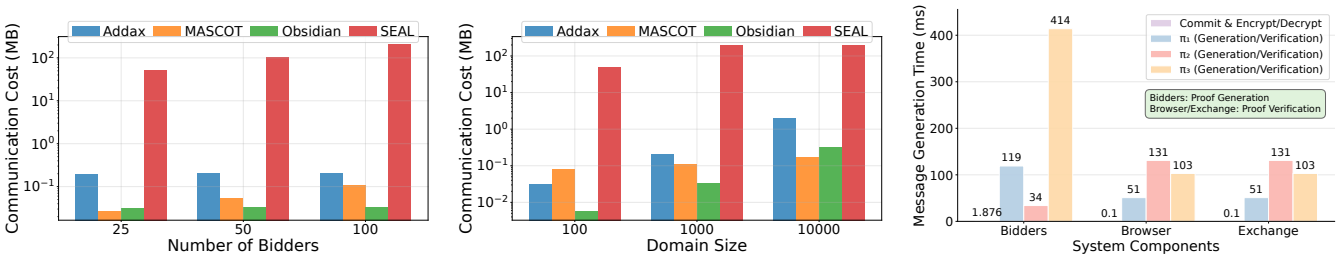


Figure 7: Obsidian achieves higher throughput and better median/99percentile latency than Addax.

MASCOT and SEAL by several orders of magnitude, as these two solutions use generic MPC and homomorphic encryption methods, without any specialization to the tasks at hand. Addax outperforms these generic systems, but still takes longer than Obsidian to perform these auctions, even though we operate under the stronger, malicious threat model. Moreover, Addax leaks the highest bid value as well. Aggregating over these measurements, Obsidian outperforms Addax by 15.4% on average on 20ms WAN RTT. For instance, consider a representative scenario with an RTT of 20ms, 50 bidders, and 1024-bit domain size, the auction computation for Obsidian takes only around 158ms to finish; 120ms comes from network latency. Moreover, Obsidian's performance scales linearly and has a smaller slope compared to the other systems, because the number of rounds of communication is independent of the number of bidders and bid domains. The effect of network latency on the auction latency is discussed in Appendix A.7.

5.3 Throughput of the exchange

Next, we seek to understand the auction throughput of the ad exchange, benchmark both Obsidian (without credential verification) and Addax, which is the system with the closest performance. Since auction systems can easily process different auctions on different cores in a parallel manner, we measure the auction throughput and latency for a single core. We generate a stress-testing trace with different arrival rates, each with 100 bidders and a bid domain of 1024. The bid in-



(a) Comm. cost vs. #bidders (domain: 1024); Obsidian outperforms other baselines.

(b) Comm. cost vs. domain sizes (100 bidders); Obsidian outperforms other baselines.

(c) Message generation time for the pre-auction bidder credential verification.

Figure 8: WAN communication costs (a and b) and credential verification overhead (c).

puts are held in memory without the effect of network latency to measure the raw throughput of the ad exchange. Fig. 7 shows the median and 99-percentile latencies for Obsidian and Addax. For the latter, we use two setups, using the non-interactive and 2-round interactive (i.e., publisher interacts with the bidders) settings. Our finding is that the throughput of Obsidian outperforms Addax by 10x and 4x, respectively, for these two settings. This is a significant result, given that Addax targets a much weaker security model (i.e., covert).

6 Discussion

Practical deployment. The offline phase is input-independent and can be scheduled like background tasks (e.g., browser/OS updates) during idle periods, amortized across many auctions without re-execution until the correlated randomness pool is exhausted. If the dealer is temporarily unavailable, the browser and exchange can fall back to a two-party protocol for correlated randomness generation, preserving liveness at higher cost.

Domain size & scalability. Obsidian fixes domain size d and bidder bound n at offline time. In practice, operators can pre-generate offline material for a small menu of (d,n) pairs and select the appropriate one per auction at negligible cost. For large d (e.g., fractional bids), the offline phase dominates but scales linearly, so operators can expand offline batches during low-traffic windows to trade storage for latency.

Bulletin board. The board U needs to support public append-only availability and anonymous write access. For availability, certificate transparency-style logs [54] are already deployed at scale. For anonymity, write access can be proxied through any mix-net or onion-routing layer.

Broader applicability. Obsidian's MPC protocol is not ad-specific: the encoding scheme and FSS-based validation apply to any sealed-bid Vickrey auction where bids are discrete and bounded. The bidder anonymity extension can be omitted when identities are already authenticated—e.g., in enterprise procurement or cloud spot markets, where the auctioneer knows all participants but still wants bid confidentiality.

7 Related work

Secure auctions. Past work has considered first-price auctions, e.g., using garbled circuit [34], homomorphic encryption [64], MPC [38, 51], or specialized algorithms [53, 72] without revealing individual bids. Secure Vickrey auctions have been studied using homomorphic encryption [16], MPC [58, 71], and specialized algorithms [45], but they typically rely on non-collusion assumptions among auctioneers and only offer weaker guarantees. Auctioneer-free designs enable bidders to jointly compute the auction outcome [17, 25, 26, 27, 39, 52], but they are not suitable for the ad ecosystem. Obsidian introduces a practical two-party auction protocol that provides malicious security.

Ad exchanges. At the pre-auction stage, browser-based solutions can reduce cross-site tracking. Adnostic [67] maintains a local user profile for client-side ad selection. Google Topics [10] summarizes user interests into coarse-grained topic IDs. Microsoft PARAKEET [9] uses a trusted intermediary to anonymize the ad requests. AdVeil [65] and Ibex [71] employ private information retrieval to hide user interests. Brave Browser [6] uses on-device machine learning for local ad selection. In addition, Google Protected Audience API [11] (previously FLEDGE) enables privacy-preserving retargeting through trusted environments. Post-auction conversion measurements can also improve privacy by coarse-grained reporting, and differential privacy to conceal individual data and reduce re-identification risks [14, 19, 33]; using divisible e-cash for conversion billing [55] further improves unlinkability. Obsidian specifically focuses on the auction stage, and the closest work to us, Addax [72] operates under a weaker (covert) threat model.

8 Conclusion

Obsidian is a secure Vickrey auction protocol in the malicious threat model. Its core contribution is an MPC-friendly algorithm that hides both losing bids and bidder identities while computing the outcome. Obsidian is also highly efficient, outperforming baselines in similar or weaker threat models, making it practical for real-world use.

Acknowledgments. We thank the anonymous NSDI reviewers and our shepherd, Nandita Dukkipati, for their valuable feedback. This work is partially supported by a VMware Early Career Faculty Grant, a Cisco grant, and NSF grants CNS-1942219, CNS-2106751, CNS-2107147, CNS-2214272.

References

[1] ads-privacy/proposals/dovekey. https://github.com/google/ads-privacy/tree/master/proposals/dovekey.

[2] Behind a Secret Deal Between Google and Facebook. https://www.nytimes.com/2021/01/17/technology/google-facebook-ad-deal-antitrust.html.

[3] GitHub - WICG/turtledove: TURTLEDOVE. https://github.com/WICG/turtledove.

[4] IAB - Standards and Guidelines — iab.com. https://www.iab.com/guidelines/openrtb/. [Accessed 24-02-2026].

[5] identity-gatekeeper/proposals/PARRROT.md. https://github.com/prebid/identity-gatekeeper/blob/master/proposals/PARRROT.md.

[6] An introduction to brave's in-browser ads. https://brave.com/blog/intro-to-brave-ads/.

[7] Latency Restrictions and Peering | Real-time Bidding | Google for Developers. https://developers.google.com/authorized-buyers/rtb/get-started/peer-guide.

[8] Microsoft SEAL: Fast and Easy-to-Use Homomorphic Encryption Library. https://www.microsoft.com/en-us/research/project/microsoft-seal/.

[9] microsoft/parakeet. https://github.com/microsoft/PARAKEET.

[10] Overview of topics api | privacy sandbox. https://privacysandbox.google.com/private-advertising/topics.

[11] Protected audience api overview | privacy sandbox. https://privacysandbox.google.com/private-advertising/protected-audience.

[12] Real-Time Ad Impression Bids Using DynamoDB | Amazon Web Services. https://aws.amazon.com/blogs/aws/real-time-ad-impression-bids-using-dynamodb/.

[13] The Google Ads Auction Exposed: Antitrust Trial Revelations - Part 1 - PPC Hero. https://www.ppchero.com/the-google-ads-auction-exposed-antitrust-trial-revelations-part-1/.

[14] Uieventattributionview | apple developer documentation. https://developer.apple.com/documentation/uikit/uieventattributionview.

[15] Unpacking Real Time Bidding through FTC's case on Mobilewalla — ftc.gov. https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2024/12/unpacking-real-time-bidding-through-ftcs-case-mobilewalla. [Accessed 24-02-2026].

[16] Masayuki Abe and Koutarou Suzuki. M+1-st price auction using homomorphic encryption. In *Proc. PKC*, 2002.

[17] Samiran Bag, Feng Hao, Siamak F Shahandashti, and Indranil Ghosh Ray. Seal: Sealed-bid auction without auctioneers. *IEEE Transactions on Information Forensics and Security*, 15:2042–2052, 2019.

[18] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. *Designs, Codes and Cryptography*, 91(11):3457–3525, 2023.

[19] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proc. OSDI*, pages 441–459, 2017.

[20] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proc. CCS*, pages 1175–1191, 2017.

[21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *Proc. SP*, 2021.

[22] Elette Boyle, Niv Gilboa, Matan Hamilis, Yuval Ishai, and Ariel Nof. Preprocessing for life: Dishonest-majority MPC with a trusted or untrusted dealer. Cryptology ePrint Archive, Paper 2025/787, 2025.

[23] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Proc. EUROCRYPT*, pages 337–367. Springer, 2015.

[24] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proc. CCS*, pages 1292–1303, 2016.

[25] Felix Brandt. Cryptographic protocols for secure second-price auctions. In *International Workshop on Cooperative Information Agents*. Springer, 2001.

[26] Felix Brandt. Secure and private auctions without auctioneers. Technical report, Citeseer, 2002.

[27] Felix Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5:201–216, 2006.

[28] Clare Duffy Brian Fung. Google is an online advertising monopoly, judge rules | CNN Business — cnn.com. https://www.cnn.com/2025/04/17/tech/google-adtech-trial-decision/index.html.

[29] Giuseppe Calderonio, Mir Masood Ali, and Jason Polakis. Fledging will continue until privacy improves: Empirical analysis of google's Privacy-Preserving targeted advertising. In *Proc. USENIX Security*, 2024.

[30] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. Cryptology ePrint Archive, Paper 2014/785, 2014.

[31] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Proc. ASIACRYPT*, 2017.

[32] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim,

Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *Proc. EUROCRYPT*, pages 415–445. Springer, 2019.

[33] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proc. NSDI*, pages 259–282, 2017.

[34] MALKHI Dahlia. Fairplay-a secure two-party computation system. In *Proc. USENIX Security*, 2004.

[35] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: from passive to active security at low cost. In *Proc. CRYPTO*, 2010.

[36] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. PKC'05, page 416–431, Berlin, Heidelberg, 2005. Springer-Verlag.

[37] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Comput. Surv.*, 42(1), December 2009.

[38] Matthew K Franklin and Michael K Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, 22(5):302–312, 1996.

[39] Chaya Ganesh, Shreyas Gupta, Bhavana Kanukurthi, and Girisha Shankar. Secure vickrey auctions with rational parties. In *Proc. CCS*, 2024.

[40] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proc. EUROCRYPT*, 2016.

[41] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM Journal on Computing*, 41(5):1193–1232, 2012.

[42] Internet Security Research Group. Divvi Up — divviup.org. https://divviup.org/.

[43] Keirsten Hansen. Digital Ad Revenue Surges 15 https://www.iab.com/news/digital-ad-revenue-2024/.

[44] J. Horwitz and K. Hagey. Google's Secret 'Project Bernanke' Revealed in Texas Antitrust Case. https://www.wsj.com/business/media/googles-secret-project-bernanke-revealed-in-texas-antitrust-case-11618097760.

[45] Ari Juels and Michael Szydlo. A two-server, sealed-bid auction protocol. In *Proc. FC*, pages 72–86, 2003.

[46] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. *Cryptology ePrint Archive*, 2011.

[47] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proc. CCS*, 2018.

[48] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proc. CCS*, 2020.

[49] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proc. CCS*, 2016.

[50] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. Cryptology ePrint Archive, Paper 2016/505, 2016.

[51] Hiroaki Kikuchi, Michael Hakavy, and Doug Tygar. Multi-round anonymous auction protocols. *IEICE Trans. Inf. Syst.*, 82(4):769–777, 1999.

[52] Lucy Klinger, Mengfan Lyu, and Lei Zhang. Secure and private vickrey auction protocols: A secure multiparty computation approach. *arXiv:2304.14626*, 2023.

[53] Kaoru Kurosawa and Wakaha Ogata. Bit-slice auction circuit. In *Proc. ESORICS*, 2002.

[54] Ben Laurie, Eran Messeri, and Rob Stradling. Certificate Transparency Version 2.0. RFC 9162, December 2021.

[55] Kevin Liao, Henry Corrigan-Gibbs, and Dan Boneh. Divisible e-cash for billing in private ad retargeting. *PoPETS*, 2024.

[56] Xin Liu, Shundong Li, Jian Liu, Xiubo Chen, and Gang Xu. Secure multiparty computation of a comparison problem. *SpringerPlus*, 5:1–17, 2016.

[57] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. Rabbit: Efficient comparison for secure multi-party computation. In *Proc.FC*, 2021.

[58] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proc. EC*, 1999.

[59] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. Spectrum: High-bandwidth anonymous broadcast. In *Proc. NSDI*, 2022.

[60] Elizabeth Nielson. Dislike: Facbook's anticompetitive monopoly on social media and why u.s. antitrust laws must adapt to the technological era. *SMU Law Review Forum*, 75:120–149, 01 2022.

[61] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009.

[62] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. Honeycrisp: Large-scale differentially private aggregation without a trusted core. In *Proc. SOSP*, 2019.

[63] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. In *Proc. CCS*, 2022.

[64] Kazue Sako. An auction protocol which hides bids of losers. In *Proc. PKC*, 2000.

[65] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. Adveil: A private targeted advertising ecosystem. *Cryptology ePrint Archive*, 2021.

[66] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. AdVeil: A private targeted advertising ecosystem. Cryptology ePrint Archive, Paper 2021/1032, 2021.

[67] Vincent Toubiana, Arvind Narayanan, Dan Boneh, Helen Nissenbaum, and Solon Barocas. Adnostic: Privacy

preserving targeted advertising. In *Proc. NDSS*, 2010.
[68] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proc. PETS*, 2022.
[69] Shuai Yuan, Jun Wang, Bowei Chen, Peter Mason, and Sam Seljan. An empirical study of reserve price optimisation in real-time bidding. In *Proc. KDD*, 2014.
[70] Weinan Zhang, Shuai Yuan, Jun Wang, and Xuehua Shen. Real-time bidding benchmarking with ipinyou dataset. *arXiv preprint arXiv:1407.7073*, 2014.
[71] Ke Zhong, Yiping Ma, and Sebastian Angel. Ibex: Privacy-preserving ad conversion tracking and bidding. In *Proc. CCS*, 2022.
[72] Ke Zhong, Yiping Ma, Yifeng Mao, and Sebastian Angel. Addax: A fast, private, and accountable ad exchange infrastructure. In *Proc. NSDI*, 2023.

A Appendix

A.1 Obsidian Other Building Blocks

Authenticated Encryption provides confidentiality and integrity guarantees for messages exchanged between two parties. It consists of a tuple of three algorithms as follows:

- $k \xleftarrow{\$} \mathsf{AE.gen}(1^\kappa)$ The key generation algorithm outputs a private key k where κ is the security parameter.
- $\bar{x} \xleftarrow{\$} \mathsf{AE.enc}(k,x)$. Encryption takes as input a key k and a message x, and outputs a ciphertext $\bar{x}$.
- $x \leftarrow \mathsf{AE.dec}(k,\bar{x})$. Decryption takes as input a ciphertext and a key and outputs either the original plaintext or a special error symbol $\perp$ on failure.

Dodis-Yampolskiy Pseudorandom Function. A pseudorandom function (PRF) is a deterministic function from a random seed and an input, such that the output is computationally indistinguishable from truly random one. The Dodis-Yampolskiy PRF [36] is defined by $y = \mathcal{F}(\mathrm{sk},x) = g^{\frac{1}{x+\mathrm{sk}}}$ where g is the generator of a bilinear group $\mathcal{G}$ of prime order. This is a verifiable PRF which means that $e(g^x \cdot \mathrm{pk}, y) = e(g,g)$ where $\mathrm{pk} = g^{\mathrm{sk}}$.

Non-interactive Zero-Knowledge Argument of Knowledge. We use non-interactive zero knowledge argument ZK and argument of knowledge ZKPoK for a relation $\mathcal{R}$. For our purposes, we require the following properties from ZK and ZKPoK: Perfect completeness; Computational Soundness; Computational Zero-Knowledge.

A.2 Proof of Lemma 1

Proof. By contradiction. Suppose that the algorithm identifies index j^* as the smallest index satisfying $c_{j^*} = T$ (with $T \le B-1$), but j^* is not the correct second-highest bid according to the auction semantics. By the definition of encoding, c_j counts the number of bidders whose bids are at least j. Therefore, a correct second-highest bid should satisfy $c_{j^*} = T$, meaning that exactly T bidders bid at least j^*. The second-highest bid is defined as the largest value with at least T bidders bidding at least that value but not all B bidders. We consider the following exhaustive cases:

Case 1. There exists some $j' < j^*$ s.t. $c_{j'} = T$. Then, by construction, the algorithm would have selected j' before j^*—contradicting the assumption that j^* was chosen.

Case 2. There exists some $j' > j^*$ s.t. $c_{j'} = T$. However, since the encoding function is monotonic decreasing in j, we have $c_{j'} \le c_{j^*}$. But by assumption, $c_{j^*} = T$, so $c_{j'} \le T$. Since j^* is defined as the smallest index satisfying this threshold, j' cannot be smaller than j^*—contradicting our assumption that j' is the correct second-highest bid.

Case 3. For all j, $c_j < T$. Then by definition, the algorithm would have already reduced the threshold to a lower value (e.g. $T-1$, $T-2$, etc.), contradicting the assumption that j^* with threshold T was chosen.

In all cases, we arrive at a contradiction. Therefore, the index j^* identified by the algorithm correctly corresponds to the second-highest bid according to the monotonic encoding and the auction semantics. □

A.3 Ring Signature Construction

Definition 1 (Unforgeability)**.** *Ring signature scheme* $(Gen, Sign, Vrfy)$ *is unforgeable if, for any ppt adversary* $\mathcal{A}$ *and any polynomial* ℓ*, the probability that* $\mathcal{A}$ *succeeds in the following experiment is negligible in* κ*:*

1. *Keys* $\{(pk_i, sk_i)\}_\ell^{i=1}$ *are generated by* $Gen(1^\kappa)$*. The public keys* $S \overset{def}{=} \{pk_i\}_\ell^{i=1}$ *are given to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *may query an oracle* $Sign'(\cdot,\cdot,\cdot)$*, where* $Sign'(R,i,M)$ *(with* $pk_i \in R$*) outputs* $Sign(R, sk_i, M)$*. (Note that we do not require* $R \subseteq S$*.)*
3. $\mathcal{A}$ *may also query a corruption oracle Corrupt that on input i returns* sk_i*. If* $\mathcal{A}$ *queries* $Corrupt(i)$ *then we say that* pk_i *is corrupted. We let C be the set of corrupted public keys at the end of the experiment.*
4. $\mathcal{A}$ *outputs* M^*, R^*, σ^**. It succeeds if (1)* $Vrfy(R^*, M^*, \sigma^*) = 1$*; (2)* $R^* \subseteq S \setminus C$*; and (3)* $\mathcal{A}$ *never queried* $Sign'(R^*, \star, M^*)$*.*

Definition 2 (Anonymity)**.** *Ring signature scheme* $(Gen, Sign, Vrfy)$ *is anonymous if, for any ppt adversary* $\mathcal{A}$ *and any polynomial* ℓ*, the probability that* $\mathcal{A}$ *succeeds in the following experiment is at most* $1/2 + negl(\kappa)$*:*

1. *Keys* $\{(pk_i, sk_i)\}_\ell^{i=1}$ *are generated by* $Gen(1^\kappa)$ *and all keys (both public and private) are given to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *outputs a message M, a ring R, and* $i_0, i_1 \in [\ell]$*. A uniform* $b \in \{0,1\}$ *is chosen, and* $\mathcal{A}$ *is given* $Sign(R', sk_{i_b}, M)$*, where* $R' = R \cup \{pk_{i_0}, pk_{i_1}\}$*.*
3. $\mathcal{A}$ *outputs a bit* b'*, and succeeds if* $b' = b$*.*

Online Phase: Browser commits to the set of bidders $\mathcal{V} = \mathrm{pk}_1, \cdots, \mathrm{pk}_n$, it has chosen as $\{\mathsf{H}(\mathrm{pk}_i || \rho_i)\}$ where ρ_i is a random bit string of sufficient length as required by the security parameter λ. The exchange announces the auction ID Ψ.

Each bidder $\mathcal{V}_i$:

- Decides on bid $b_i \in \{1, \ldots, d\}$.
- Generates ephemeral key pair $(\mathrm{pk}', \mathrm{sk}') \xleftarrow{\$} \mathrm{KeyGen}(\mathrm{pp})$ and computes PRF evaluation: $\gamma = \mathcal{F}(\mathrm{sk}_i, \mathsf{H}(\mathrm{pk}'))$
- Issues commitment: $\mathrm{com}_{\mathrm{pk}_i} = \mathrm{pk}_i \cdot P + r \cdot Q$ and generates per-auction token: $\tau = \mathsf{H}(\mathrm{sk}_i || \psi)$
- Encrypts opening of the commitment for browser: $\bar{s} \leftarrow \mathrm{AE.enc}(\mathrm{sk}_{\mathcal{B},i}, \mathrm{Open}(\mathrm{com}_{\mathrm{pk}_i}))$
- Constructs zero-knowledge proofs: $\pi_1 = \mathsf{ZK}\{\mathrm{Open}(\mathrm{com}_{\mathrm{pk}_i}) \in R\}$, $\pi_2 = \mathsf{ZKPoK}\{\mathrm{sk}_i | \gamma = \mathcal{F}(\mathrm{sk}_i, \mathsf{H}(\mathrm{pk}')), \mathrm{Open}(\mathrm{com}_{\mathrm{pk}_i}) = g^{\mathrm{sk}_i}\}$, $\pi_3 = \mathsf{ZKPoK}\{\mathrm{sk}_i | \tau = \mathsf{H}(\mathrm{sk}_i || \psi), \mathrm{Open}(\mathrm{com}_{\mathrm{pk}_i}) = g^{\mathrm{sk}_i}\}$
- Publishes $(l_i, \mathrm{com}_{\mathrm{pk}_i}, \mathrm{pk}', \gamma, \tau, \bar{s}, \pi_1, \pi_2, \pi_3)$ on $\mathcal{U}$

The Browser $\mathcal{B}$:

- Decrypts $\bar{s}$ using $\mathrm{sk}_{\mathcal{B},i}$ to recover bidder identity, and verifies commitment opening and checks the validity of ZKPs
- If valid, publishes acknowledgment and encrypted share $\mathrm{AE.enc}(\mathrm{sk}_{\mathcal{B},i}, [r_i]_{\mathcal{B}} || \nu_{\mathcal{B},i})$ on $\mathcal{U}$

The Exchange $\mathcal{E}$:

- Verifies all ZKPs π_1, π_2, π_3 for anonymous credential validation
- Checks token $\tau = \mathsf{H}(\mathrm{sk}_i || \psi)$ for rate-limiting (no duplicates per auction ψ)
- Generates ephemeral key pair $(\mathrm{sk}'', \mathrm{pk}'')$ and establishes a session key with bidder using pk'
- Publishes encryption of $[r_i]_{\mathcal{E}} || \nu_{\mathcal{E},i})$ under common secret key $\mathrm{sk}_{\mathcal{E},i}$ (session key along with signed pk'')

Again each bidder $\mathcal{V}_i$:

- Decrypts the ciphertexts to obtain $r_i, \nu_{\mathcal{B},i}, \nu_{\mathcal{E},i}$ and check consistency with the dealer's commitment $\mathsf{H}(r_i || \nu_{\mathcal{B},i} || \nu_{\mathcal{E},i})$, aborts if check fails
- Computes masked bid $a_i = b_i - r_i$ and publishes the encryption of a_i under $\mathrm{sk}_{\mathcal{B},i}$ and $\mathrm{sk}_{Ex,i}$ (session key)

Browser $\mathcal{B}$ and Exchange $\mathcal{E}$:

- *Map bid shares to corresponding DPF expansions:* For each bidder i, decrypt to get a_i and shift previously generated DPF expansions to get $\langle f_{b_i,1} \rangle : \langle \hat{y}_i[j] \rangle = \langle y_i[(j + a_i) \bmod d] \rangle$
- Convert to shares of the bid encoding $\langle \tilde{y}_i \rangle = \langle \mathrm{E}(b_i) \rangle$: $\langle \tilde{y}_i[1] \rangle = \langle \hat{y}_i[1] \rangle$, $\langle \tilde{y}_i[j] \rangle = \langle \hat{y}_i[j-1] \rangle + \langle \hat{y}_i[j] \rangle, j \in [2, d]$
- *Compute the second-highest bid:* Compute share of column-wise sums: $\langle c_j \rangle = \sum_{k=1}^{n} \langle \tilde{y}_k[j] \rangle$ for $j = 1, \ldots, d$
- Convert each $\langle c_j \rangle$ to the corresponding DPF expansion using :
  - Re-randomize shares and reveal offset: $\langle t_j \rangle = \langle r'_j \rangle - \langle c_j \rangle$, and shift vectors: $\langle \tilde{y}'_j[l] \rangle = \langle y'_j[(l + t_j) \bmod (n+1)] \rangle$
- Detect ties: convert $\langle s \rangle = \sum_{j=1}^{d} \tilde{y}'_j[n-1]$ to DPF expansion $\langle h \rangle$:
  - Re-randomize shares, reveal offset: $\langle t_k \rangle = \langle r''_k \rangle - \langle s \rangle$, shift $\langle \tilde{y}''_k[l] \rangle = \langle y''_k[(l + t_k) \bmod (n+1)] \rangle$, evaluate $h[1]$.
- If no tie ($h[1] = 0$): compute $\tilde{y}'_j[n-1] + \tilde{y}'_j[n]$ and find $j^* = \min\{ j : \tilde{y}'_j[n-1] + \tilde{y}'_j[n] \geq 1 \}$
- If tie detected ($h[1] = 1$): repeat with threshold $n-2, n-3, \ldots$ using binary search

Identify auction winner:

- For each row k, re-randomize shares and reveal $\langle \tilde{y}_k[j^*] \rangle$
- Handle ties by random selection among all k with $\tilde{y}_k[j^*] = 0$

Figure 9: Online protocol description: additional steps required for anonymizing the bidders are colored in red.

Parameters: Number of bidders n, bid domain $\{1,\ldots,d\}$, finite field $\mathbb{F}$.
Output: Auction winner k^* and second-highest bid j^*.
Parties: Bidders $\{\mathcal{V}_1,\ldots,\mathcal{V}_n\}$, Browser $\mathcal{B}$, Ad Exchange $\mathcal{E}$, Dealer $\mathcal{D}$ (preprocessing only).

Setup Phase:

- All parties authenticate via PKI and key agreement: $\mathsf{sk}_{\mathcal{B},i}$ and $\mathsf{sk}_{\mathcal{E},i}$ for each bidder $\mathcal{V}_i \in \mathcal{V}^*$
- Public parameters: $\mathsf{pp} \xleftarrow{\$} \mathsf{KA.param}(\lambda)$, field $\mathbb{F}$ and setup anonymous bulletin board $\mathcal{U}$ accessible to all parties

Offline Phase:
Dealer $\mathcal{D}$:

- Samples a global MAC key $\alpha \in \mathbb{F}$ as two calls to $\mathcal{F}_{\mathsf{Rand}}$ one with the browser and the other with the exchange
- For each bidder $i \in [n]$, samples random masks $r_i \in [d]$ with two calls to $\mathcal{F}_{\mathsf{Rand}}$ as before
- Generates FSS key pairs for $f_{r_i,1}$ and their MACs: $(\mathsf{K}^{\mathcal{B}}_{r_i}, \mathsf{K}^{\mathcal{E}}_{r_i}) \xleftarrow{\$} \mathrm{FSS.gen}(1^\lambda, r_i, 1), (\mathsf{K}^{\mathcal{B}}_{\alpha,i}, \mathsf{K}^{\mathcal{E}}_{\alpha,i}) \xleftarrow{\$} \mathrm{FSS.gen}(1^\lambda, r_i, \alpha)$
- Generates another set of keys $(\mathsf{K}^{\mathcal{B}}_{r'_j}, \mathsf{K}^{\mathcal{E}}_{r'_j}) \xleftarrow{\$} \mathrm{FSS.gen}(1^\lambda, r'_j, 1), (\mathsf{K}^{\mathcal{B}}_{\alpha,j}, \mathsf{K}^{\mathcal{E}}_{\alpha,j}) \xleftarrow{\$} \mathrm{FSS.gen}(1^\lambda, r'_j, \alpha)$ over domain $[n+1]$ for column sum processing, r'_js are randomly chosen as before
- Generates another set of keys $(\mathsf{K}^{\mathcal{B}}_{r''_k}, \mathsf{K}^{\mathcal{E}}_{r''_k}) \xleftarrow{\$} \mathrm{FSS.gen}(1^\lambda, r''_k, 1), (\mathsf{K}^{\mathcal{B}}_{\alpha,k}, \mathsf{K}^{\mathcal{E}}_{\alpha,k}) \xleftarrow{\$} \mathrm{FSS.gen}(1^\lambda, r''_k, \alpha)$ over domain $[n]$ for tie detection, r''_ks are randomly chosen as before
- Distributes keys and secret shares $\{[r_i]_{\mathcal{B}}, [r'_j]_{\mathcal{B}}, [r''_k]_{\mathcal{B}}, [\alpha]_{\mathcal{B}}\}$ to $\mathcal{B}$ and $\{[r_i]_{\mathcal{E}}, [r'_j]_{\mathcal{E}}, [r''_k]_{\mathcal{E}}, [\alpha]_{\mathcal{E}}\}$ to $\mathcal{E}$ and publishes $\mathrm{com}_{r_i} = \mathsf{H}(r_i || \mathsf{v}_{\mathcal{B},i} || \mathsf{v}_{\mathcal{E},i})$ on the bulletin board

Browser $\mathcal{B}$ and Exchange $\mathcal{E}$:

- Expands DPF shares for all keys: $y_i^{(\sigma)} = \{\mathrm{FSS.evalAll}(\mathsf{K}^{(\sigma)}_{r_i})\}, my_i^{(\sigma)} = \{\mathrm{FSS.evalAll}(\mathsf{K}^{(\sigma)}_{\alpha,i})\}$, for $\sigma \in \{\mathcal{B}, \mathcal{E}\}$
- $y'^{(\sigma)}_j = \{\mathrm{FSS.evalAll}(\mathsf{K}^{(\sigma)}_{r'_j})\}, my'^{(\sigma)}_j = \{\mathrm{FSS.evalAll}(\mathsf{K}^{(\sigma)}_{\alpha,j})\}$ and $y''^{(\sigma)}_k = \{\mathrm{FSS.evalAll}(\mathsf{K}^{(\sigma)}_{r''_k})\}, my''^{(\sigma)}_k = \{\mathrm{FSS.evalAll}(\mathsf{K}^{(\sigma)}_{\alpha,k})\}$, for $\sigma \in \{\mathcal{B}, \mathcal{E}\}$
- Execute malicious sketching protocol to verify:
  - $\mathsf{K}^{(\sigma)}_{r_i}, \mathsf{K}^{(\sigma)}_{\alpha,i}$ form shares of well-formed DPFs; $[r_i]_\sigma$ form shares of the DPF indices & MAC correctness: $my_i = \alpha \cdot y_i$

Figure 10: Obsidian: Protocol Description (Setup + Offline)

RS.GenParam(1^κ)

- Generate the parameters of the PRF $\mathcal{F}$ $\text{pp}_\mathcal{F} \xleftarrow{\$}$ PRF.param(1^κ)
- Generate the parameters for the keys $(\mathcal{G}, g) \xleftarrow{\$}$ KA.param(1^κ)
- Generate the parameters of the ZKP system $\text{pp}_\pi \xleftarrow{\$}$ ZKP.setup(1^κ)

Output $pp = (\text{pp}_\mathcal{F}, \mathcal{G}, g, \text{pp}_\pi)$

RS.KeyGen(pp)

- Parse $\text{pp} = (\text{pp}_\mathcal{F}, \mathcal{G}, g, \text{pp}_\pi)$
- Generate a pair of keys $(\text{sk}, \text{pk}) \xleftarrow{\$} \text{KA.gen}(1^\kappa)$

Output. (sk, pk)

RS.Sign(m, R, sk, pp)

- Parse $\text{pp} = (\text{pp}_\mathcal{F}, \mathcal{G}, g, \text{pp}_\pi)$
- Parse $R = \{\text{pk}_1, \cdots, \text{pk}_n\}$
- If $g^{\text{sk}} \notin R$, then return $\perp$
- Generate a commitment com_{pk} where $\text{pk} = g^{\text{sk}}$
- Compute $y = \mathcal{F}(\text{sk}, m)$
- Generate the following ZKPs

$\pi_1 = \mathsf{ZK}\{\text{Open}(\text{com}_{\text{pk}}) \in R\}$, $\pi_2 = \mathsf{ZKPoK}\{\text{sk} | \gamma = \mathcal{F}(\text{sk}, m), \text{KA.valid}(\text{Open}(\text{com}_{\text{pk}}), \text{sk}) = 1\}$

Output. $m, \sigma = (y, \pi_1, \pi_2)$

RS.Verify(m, σ, R)

- Parse $\sigma = (y, \pi_1, \pi_2)$
- If π_1 and π_2 are valid set $b = 1$, else $b = 0$

Output. b

Figure 11: New Ring Signature Scheme

Theorem 2. *The construction in Fig. 11 is a ring signature.*

Proof. The unforgeability of the scheme follows from the pseudo randomness of the PRF and the zero-knowledge property of the ZKPoK π_2. In particular, an adversary who forges a signature for some ring R of uncorrupted keys, we can use the knowledge extractor of the ZKPoK to extract sk such that KA.Public$(\text{sk}) \in R$ for some uncorrupted party i. That is, the adversary is able to find a private key for some uncorrupted party, which happens with negligible probability. The anonymity of the scheme follows similarly. □

A.4 Sketching Scheme

To ensure malicious security against the ill-formed FSS keys, we employ a sketching scheme to verify the correctness of these keys. Over a DPF lookup of size $d = 2^k$, the browser and the exchange evaluate vectors y and m_y, of length d. The scheme has two primary objectives: verification should always succeed for well-formed keys, and malformed vectors y or my should be accepted with only negligible probability. The input shares are constructed over a finite field $F = 2^m$, for some m. We employ the following linear sketch matrix $L \in \mathbb{Z}_F^{4\times d}$:

$$L = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d} \\ a_{3,1} & a_{3,2} & \cdots & a_{3,d} \\ a_{4,1} & a_{4,2} & \cdots & a_{4,d} \end{pmatrix}$$

For each $i \in [d]$, the entries $a_{1,i}, a_{2,i}$ are sampled uniformly from $\mathbb{Z}_{2^s}$ (in implementation, $\mathcal{B}$ and $\mathcal{E}$ share PRG seeds to generate these values), $a_{3,i} = a_{1,i} \cdot a_{2,i} \pmod F$, and $a_{4,i} = i - 1$. The intuition behind this construction is clarified in Eq. 4. This matrix requires generation only once and can be reused throughout the protocol unless a party aborts. Let L_1, L_2, L_3, L_4 represent the rows of L.

Verification. Define $z_j = \langle L_j, y \rangle \in F$ for $j = 1, 2, 3, 4$. Furthermore, let $z^* = \langle L_1, m_y \rangle \in F$. Note that each party only holds a share of these values, i.e., y (and thus each z_j) is secret shared over F between $\mathcal{B}$ and $\mathcal{E}$. The parties then evaluate the following expression (within MPC) and check if the value (as an element of F) is equal to 0:

$$(z_1 z_2 - z_3) + (z_4 - r) \stackrel{?}{=} 0 \tag{2}$$

$$(\alpha z_1 - z^*) \stackrel{?}{=} 0 \tag{3}$$

The browser and exchange run this verification over MPC and use this verification to check correct/incorrect keys.

Completeness. The completeness of this sketching scheme is pretty intuitive. For honestly generated correlated randomness, $(z_4 - r)$ by the construction of the DPF scheme equals zero (the DPF outputs 1 at r). At the same time, since y is of hamming weight one, the first term is zero:

$$z_1 z_2 - z_3 = \sum_i a_{1,i} a_{2,i} (y_i^2 - y_i) + \sum_{i \neq j} a_{1,i} a_{2,j} (y_i y_j) = 0 \tag{4}$$

, where the first term is zero because each $y_i \in \{0, 1\}$ and the second term is zero because if at most one of y_i, y_j is non-zero for any pair of distinct i, j. Thus, each term equals zero for Eq. 2. Moreover, the verification Eq. 3 is valid by honest construction.

Correctness. The correctness of the protocol is easy to follow from the correctness of the DPF:

$$u[i] = y[i + x] \neq 0 \text{ iff } i + x = r$$

A.5 Security Analysis

in Fig. 12. The input I_1 corresponds to the initial set of bidders identified by the browser $\mathcal{B}$. However, participation is not restricted—any entity can choose to join via the anonymous bulletin board, which is captured by I_2. The input I_3 represents the subset of bidders that the browser ultimately approves. Note that $\mathcal{B}$ is permitted to exclude some bidders from I_1 as well.[11] Finally, I_4 represents the set of approved bidders who actually submit bids and are included in the auction. This allows the bidders flexibility to drop out before the auction. We use the simulation paradigm [61] to prove Obsidian's security guarantees (Appendix A.6):

Ideal Functionality

Initialization. All bidders are associated with a unique, logical ID.
Functionality

- Receives a set of IDs I_1 from the browser
- Receives a set of IDs I_2 of all bidders who want to participate in the auction
- Outputs $|I_2|$
- Send I_2 to the browser
- Receive a set of IDs I_3 s.t. $I_3 \subseteq \{I_2 \cap I_1\}$
- Outputs $|I_3|$
- Receives bids from bidders in $I_4 \subseteq I_3$
- Outputs $|I_4|$
- Send I_4 to the browser
- Runs the Vickrey auction algorithm on the bids
- Sends the value of the second-highest bid and the ID of the bidder with the highest bid to the browser
- Sends the value of the second-highest bid to the exchange.

Figure 12: Ideal Functionality: $\mathcal{F}_{\text{Obsidian}}$

A.6 Proof of Theorem 3

Theorem 3. *Assume the ZKPs are Perfect Special Honest-Verifier Zero-Knowledge, i.e., they can be faked by a simulator, H is a random oracle, $\mathcal{F}$ is a PRF, and AE is a semantically secure encryption. Then Obsidian achieves the ideal functionality given by Figure 12 against a static adversary corrupting one out of the browser, exchange, and the dealer, and any number of bidders.*

Proof. **Honest Browser and Exchange.** We first consider the case with only malicious users. First, the simulator creates a simulated browser and exchange to interact with the adversarial bidders, and then extracts the sk from the ZKPoK. For each bidder $\mathcal{V}_i$ with a valid submission, the simulated browser computes $\text{sk}_{\mathcal{B},i}$ and encrypts a random value (r'_1, τ'_1) and sends it to the bidder. For each bidder with a valid submission, the simulated exchange will sample an ephemeral key pair $(\text{pk}'', \text{sk}'')$, and encrypt a random value (r'_2, τ'_2) with the new secret key and sends it to the bidder. Let h be the hash the dealer submitted. The simulator reprograms the random oracle such that $\mathsf{H}(r'_1 + r'_2 || \tau'_1 || \tau'_2) = h$.
Hybrids. Let $\mathcal{H}_0$ be the view of the adversary and output of the protocol in the real world. Let $\mathcal{H}_1$ be the output with the zero knowledge proof extractions performed. This is indistinguishable by the security of the zero-knowledge proof system. Let $\mathcal{H}_2$ be $\mathcal{H}_1$ but with the simulated browser instead of the real one and the real exchange altered to think the public key $\text{pk}_{\mathcal{B}}$ key generated by the simulated browser is the public key of the real browser. We now show $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_2$. Note that the set of bidders receiving a reply is exactly the same in both cases as the simulated browser verifies proofs the same way and is told by the ideal functionality who the real browser would drop. Next, let $\mathcal{H}_3$ be the view in the ideal world. $\mathcal{H}_3$ is with high probability equal to $\mathcal{H}_2$ because of the security of the secret-shares and the associated MAC checks, and FSS.
Malicious Browser. *Offline Phase.* We construct a simulator Sim as follows. Sim invokes $\mathcal{F}_{\mathsf{Rand}}$ to get random values for $[\alpha]_0, [\alpha]_1, [r]_0, [r]_1$. Sim generates the DPF keys honestly and sends a share of $[\alpha r]$ and $\mathsf{K}_r^{\mathcal{B}}, \mathsf{K}_\alpha^{\mathcal{B}}$ to $\mathcal{A}$. Sim also runs Π_{Obsidian} internally to simulate the exchange and the dealer. With the internal run, Sim proceeds to complete the malicious sketch with $\mathcal{A}$. If the check fails, Sim sends abort to the ideal functionality $\mathcal{F}_{\text{Obsidian}}$.
Online Phase. The simulator has access to the random oracle and hence, knows the set of bidders the browser has selected for the current auction. It forwards this to the ideal functionality. We assume all honest bidders chosen for a particular auction will always participate. From the calls to the random oracle, it identifies the set of honest and legitimate bidders who have been selected for the auction $\{\text{pk}_1, \cdots, \text{pk}_c\}, c \leq n$. For each honest bidder, it simulates $\bar{s} = m \oplus k$, creates a commitment for pk using the mask m, and sends the commitment, $\bar{s}$ and a randomly chosen ν to the browser. The simulator can then reprogram the random oracle such that $\mathsf{H}(\text{sk}_{\mathcal{B},i} || \nu) = k$. The simulator also sends a random value as the masked bid to the browser. During reveal, the simulator can create the consistent values from the output of the functionality.
Hybrids. Let $\mathcal{H}_0$ be the adversary's view in the real world.

[11] It is straightforward to enforce that all valid bidders from I_1 be retained by requiring the browser to provide a ZKP that any rejected bidder's public key is not part of the originally committed list. We do not enforce this here, as we allow the browser full discretion in selecting the final set of bidders.

Let $\mathcal{H}_1$ be the output after reprogramming the random oracle. This is indistinguishable since the oracle is random. Let $\mathcal{H}_2$ be $\mathcal{H}_1$ but with the simulated exchange instead of the real one. $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_2$ because of the security of the encryption scheme and zero-knowledge proofs. Let $\mathcal{H}_3$ be the view in the ideal world. $\mathcal{H}_3$ is with high probability equal to $\mathcal{H}_2$ because of the security of the secret-shares and the associated MAC checks, and FSS.

Malicious Exchange. The simulator behaves the same way as that in the offline phase.

Online Phase. The simulator gets the count of the number of messages received from the ideal functionality, and computes how many of them are honest—denoted as k. Next it selects a set of k public keys at random from $\mathcal{V}^* \setminus \{\text{public keys of corrupt bidders}\}$, commits to them and simulates the messages and ZKP for the exchange by correctly generating the pk's. If the ideal functionality aborts at any time, it sends abort to the adversary.

Hybrids. Let $\mathcal{H}_0$ be the view of the adversary in the real world. Let $\mathcal{H}_1$ be the output with the ring signatures replaced by signatures on the random set of public keys. This is indistinguishable by the anonymity of ring signatures. Let $\mathcal{H}_2$ be $\mathcal{H}_1$ but with the simulated browser instead of the real one. In particular, the public key $\text{pk}_{\mathcal{B}}$ key generated by the simulated browser is the public key of the real browser. Since, the simulator aborts if the ideal functionality aborts, $\mathcal{H}_1$ is indistinguishable from $\mathcal{H}_2$. Let $\mathcal{H}_3$ be the view in the ideal world. $\mathcal{H}_3$ is with high probability equal to $\mathcal{H}_2$ because of the security of the secret-shares and the associated MAC checks, and FSS.

Malicious Dealer. Sim invokes $\mathcal{F}_{\text{Rand}}$= to get random values for $\alpha_0, \alpha_1, r_0, r_1$. It receives shares of αr and the DPF keys from $\mathcal{A}$. If the shares are invalid, Sim sends abort to the ideal functionality.

□

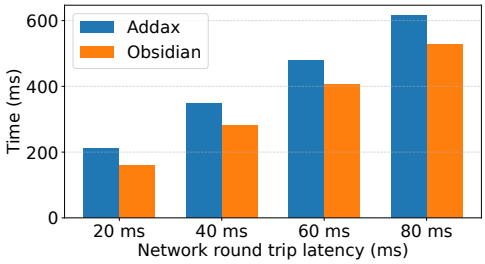


Figure 13: Auction latency for different network latencies (100 bidders, 1024 bid domain)

A.7 Effect of Network Latency

We further evaluated the effect of network latency on the overall latency of the auction. Given the closest comparison is with Addax [72], we primarily compare Obsidian for an auction with 100 bidders and a bid domain of 1024 in Fig 13. Observably, the auction latency Obsidian scales linearly with network latency, is consistently lower than Addax, and even with 80 ms round-trip latency, is less than 500 ms.