# Verifiable Auctions for Online Ad Exchanges

Sebastian Angel and Michael Walfish
The University of Texas at Austin

## ABSTRACT

This paper treats a critical component of the Web ecosystem that has so far received little attention in our community: ad exchanges. Ad exchanges run auctions to sell publishers' inventory—space on Web pages—to advertisers who want to display ads in those spaces. Unfortunately, under the status quo, the parties to an auction cannot check that the auction was carried out correctly, which raises the following more general question: how can we create verifiability in low-latency, high-frequency auctions where the parties do not know each other? We address this question with the design, prototype implementation, and experimental evaluation of VEX. VEX introduces a technique for efficient, privacy-preserving integer comparisons; couples these with careful protocol design; and adds little latency and tolerable overhead.

**Categories and Subject Descriptors:** C.2.4 [Computer-Communication Networks]: Security and Protection; K.4.4 [Electronic Commerce]: Security

**Keywords:** verifiable, auctions, ad exchanges, online advertisement

## 1 INTRODUCTION

*Ad exchanges* [5, 9, 11] are revolutionizing online advertising [48], an industry responsible for hundreds of billions of dollars yearly [60]. For each visit by a user to a participating publisher's Web site, an ad exchange runs an auction in real time to sell the publisher's inventory (space for an ad) to advertisers. These auctions benefit advertisers (who appreciate fine-grained, targeted ads) and publishers (who can sell space for such ads at a premium).

Despite their financial importance, scale, and strong trust assumptions, ad exchanges—versus online advertisement generally [23, 31, 32, 35, 47, 51, 54–56]—have received little attention from computer scientists. Muthukrishnan [39] has articulated research problems (including the one that we address), but we believe that this paper is the first to examine a concrete solution or system.

The question that we investigate is: how can an ad exchange prove to publishers and advertisers that an auction was conducted correctly, without disclosing information about the bids and bidders? To the extent that auctions today proceed on trust and the reputation of the auctioneer (and hence only a few players are auctioneers), verifiable auctions could lower the barriers to entry for new auctioneers. Moreover, verifiable auctions would benefit even established ad exchanges, by enhancing the service that they provide. Finally, our solution is relevant beyond ad exchanges (§9).

The general problem of auction verification has been widely studied [17, 27, 30, 37, 45, 46]. However, the technical challenges of ad exchanges, given below, mean that previous work is not directly applicable:

- **Delay-sensitive.** An auction must happen between the time when the user visits a Web site and the time when the complete page, including the ad, loads in the user's browser. This window could be very short (e.g., 100 ms), as latency is critical to the display of Web pages.
- **High volume.** Ad exchanges handle billions of transactions per day, consuming significant network and storage resources [43].
- **Difficult to bootstrap trust.** The only entity guaranteed to be known a priori by any party to an auction is the ad exchange itself (for instance, advertisers may not know each other). Moreover, membership is dynamic: publishers and advertisers frequently join and leave a given exchange.

We surmount these challenges with VEX, a system for verifiable auctions. VEX's goal in the online ad exchange context is to *improve the integrity of auctions while not making privacy and availability worse than they are today*. (It would obviously be ideal to improve privacy and availability too, but integrity is a difficult problem on its own.)

VEX has three salient aspects. First, VEX is separated into two phases: it adds some in-band overhead to the auction and relies on a more expensive (but still practical) offline auditing step. This separation keeps auction latency low while permitting cryptographic operations as part of verifying the auction results. This separation is acceptable because misbehavior in this context, even if detected offline, has non-technical ramifications (loss of business, legal action, etc.).

Second, VEX introduces a new technique for privacy-preserving comparisons, which may be useful elsewhere. Specifically, an entity encodes a value $v$ in the *length* of a hash chain and commits to the value by exposing the tail of the hash chain;[1] later, given a query $q$, the entity can prove that $v \geq q$ (without disclosing $v$), by revealing an appropriate node in the hash chain (the technique can also be used to prove statements of the form "$v \leq q$").

Third, VEX uses careful protocol design to compose these privacy-preserving comparisons with the auction protocol, to achieve verifiable auctions. The protocol provides the guarantee that if all parties behave, audits succeed, whereas if misbehavior occurs (auctioneer chooses the wrong winner or the wrong price, ignores a bidder's bid, etc.), it can be detected during the auction phase or the audit phase.

We have implemented and evaluated VEX (§7). Our prototype imposes additional latency in the auction phase of 50 ms (at the 95th percentile), and moderate computation (2× overhead) and storage cost for the auctioneer. The audit phase is higher overhead—in computational cost, the audit-to-auction ratio is roughly 1-to-160

[1] With this scheme, the larger $v$ is, the more work the entity has to do. However, in our context, the maximum value of $v$ will not be very large, as discussed in Section 3.



*SIGCOMM '13*, August 12–16, 2013, Hong Kong, China.

in one of our optimized variants—but would happen less often. All of these costs are affordable and consistent with the usual price of cryptographic guarantees in systems.

Our work has several limitations. First, it may not be easy to persuade today's ad exchanges to adopt VEX; on the other hand, offering this protocol may attract publishers and advertisers. Second, although bidders' identities and bids remain secret, VEX reveals the total number of bids (in an auction) and the sale price (if the auction is audited). Third, our threat model is not the strongest possible; among other things, we do not prevent misbehavior (only detect it). Fourth, the ad exchange invites the bidders, so it can manipulate auctions, by choice of bidders. However, such manipulation is mostly limited (§5) to endowing a colluding bidder with a right of first refusal.[2] Last, a malevolent publisher can deliver the auctioned ad to the wrong user; this issue concerns the authenticity of what is being auctioned and is a different problem from auction integrity, but we can limit abuse by publishers (§5).

While online advertising has received much research attention (§8), verifiability in ad exchanges [39] has not been concretely addressed. Thus, this paper's contributions are:

- A technique for fast, private integer comparisons (§3).
- The design of VEX, a verifiable ad exchange protocol (§4) with concrete guarantees (§5). For the most part, this design generalizes to other online auction platforms (§9).
- The implementation and evaluation of VEX and variants (§6–§7).

## 2 BACKGROUND

This section surveys online advertising (§2.1), ad exchanges (§2.2), and various vulnerabilities (§2.3).

### 2.1 A brief history of online advertising

Advertising is a primary source of revenue for many Web sites that provide free content (e.g., videos, news, blogs). These content providers, also known as *publishers*, have for decades used *ad networks* [6, 8, 13] to sell advertising space on their sites. Ad networks would buy *impressions* by the thousands (a priori) from publishers, and distribute the impressions among the network's advertisers.

Recent advances in behavioral targeting [15, 24] have increased the effectiveness of ad networks by allowing them to better match impressions to advertisers' preferences. This approach inspired a new kind of entity, the *ad exchange*, as a way of dealing with impressions that publishers were unable to sell in advance to ad networks [25]. Because of their simplicity, power, and versatility, ad exchanges were well-received, and they quickly began serving premium impressions. They now serve billions of transactions per day [43] and are expected to account for 27% of the total display ad sales in the United States by 2015 [57] (one estimate is that ads sold via exchanges will be an $8 billion market by 2017 [48]).

### 2.2 Ad exchanges

On an ad exchange, a publisher auctions individual ad impressions to advertisers and ad networks, selling the impression to the highest bidder. The enabling technology is real-time bidding (RTB) [26], which provides advertisers with data on users and Web pages, allowing them to bid only on the impressions that they consider pertinent. This gives publishers greater profit margins. Moreover, publishers get flexibility: they can sell impressions when available, instead of in batches, based on advance forecasts.

Figure 1 gives a high-level overview of ad exchanges. The key parties in an ad exchange are as follows:

- *Users* are the page visitors who ultimately see the ad.
- *Sellers* are publishers who operate Web sites where ads can be displayed. Medium and large publishers are often represented by a third party known as a *supply-side platform*. Small publishers are frequently represented by ad networks.
- *Bidders* are advertisers wishing to bid on a particular impression. Medium and large advertisers often contract with a *demand-side platform*, while smaller advertisers contract directly with ad networks.
- The *ad exchange* performs the auction. It processes requests from sellers of impressions and requests bids from subscribed bidders.
- *Ad storage servers* store the actual advertisement (e.g., a banner). These servers can be maintained by content delivery networks, ad networks, demand-side platforms, or large advertisers.

**Ad exchange protocol.** The ad exchange protocol is initiated when a user visits a publisher's page. The page contains an HTML `iframe` or JavaScript snippet that causes the user's Web browser to send an *ad tag request* to the seller's server (Figure 1, step ①). This request contains a unique identifier that allows the server to determine the properties of the ad space: the URL, the dimensions, or the type of ad supported (e.g., video, image, text). The seller's server then creates an *auction request* (Figure 1, step ②). The auction request contains four types of information: ad space information, financial information (e.g., reserve price), user information, and a time stamp. The user information varies widely and is based on a publisher's targeting strategy. It may include demographic (e.g., age) and geographic information, as well as information about the user's browsing history.

When the ad exchange receives an auction request, it performs three actions. First, it looks up the user, by the process of *cookie syncing* [2, 53]. Second, it chooses a set of bidders to participate in the auction; the specifics of this step vary among ad exchanges. Finally, the ad exchange constructs and sends a *bid request* to the selected bidders (Figure 1, step ③). The bid request includes the original impression information provided by the seller (ad space information, financial information, user information, time stamp) plus additional information held by the ad exchange about the user or the seller (e.g., relevant keywords for the seller's site).

Bidders process each bid request by identifying the potential value of the impression; each bidder may have different bidding strategies. A bidder sends a *bid message* to the ad exchange (step ④). Each bid message contains a bid as well as an ad tag that, should the bidder win the auction, will be delivered to the user's browser and will be used to fetch the bidder's ad.

Upon receiving all of the bids or when a timeout is triggered, the ad exchange evaluates the auction. It chooses a winner (the highest bidder), setting the *sale price* equal to the second-highest bid. The ad exchange then notifies the winning bidder of the sale price (step ⑤) and forwards both the bidder's ad tag and the sale price to the publisher (step ⑥). At this point, the publisher can respond to

[2] Right to meet (and win in preference to) an existing offer before the end of a transaction [1].

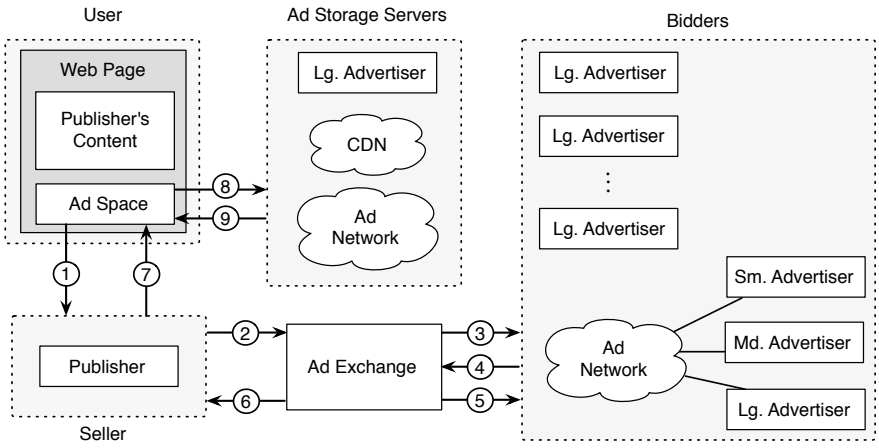



**Figure 1—**Overview of ad exchanges. The steps required to deliver an ad to the user are as follows. ① The user's browser requests an ad tag from the publisher's server. ② The publisher's server requests an ad tag from the ad exchange, thereby initiating an auction. ③ The ad exchange asks interested advertisers and ad networks to bid on the given impression. ④ Bidders submit their bid along with their corresponding ad tag. ⑤ The winning bidder is notified of the result of the auction. ⑥ The seller receives the ad tag as well as the impression's sale price. ⑦ The seller responds to the user's browser request by providing the ad tag. ⑧ The ad tag causes the user's browser to fetch an ad from an ad storage server. ⑨ The ad is fetched and rendered in the user's browser. This process often completes within a few hundred milliseconds.

the user's ad tag request (step ⑦), causing the user's Web browser to fetch the ad from the appropriate ad storage server (steps ⑧, ⑨).

### 2.3 Vulnerabilities

*Commission attack.* The ad exchange receives payment from the winning bidder when an impression is sold. Instead of reporting this payment to the seller, the ad exchange reports a lower price, to increase its "commission".

*Bid discrimination attack.* The ad exchange selects a bidder other than the highest to win, giving that bidder access to the media at a below-market price; the ad exchange may then receive a side payment from this bidder.

*Second-price attack.* The ad exchange can misreport the second-highest bid in a way that forces the winning bidder to pay a higher price. This attack covers spurious bids by auction participants, sometimes known as shill bidding [52], or cases when the ad exchange manufactures bidders.

Are these things happening today? We do not know: these behaviors are easy to carry out but difficult to detect. While a measurement study (e.g., submit multiple bids to auctions) would be worthwhile, it would give only a snapshot, with no guarantees about the future. By contrast, the system described in the sections ahead would rule out (or significantly raise the barrier to) such behaviors.

## 3 PRIVATE INTEGER COMPARISONS

This section introduces a protocol for private integer comparisons. We describe it separately for clarity and modularity, and because the protocol may be useful in other contexts.

We want to emphasize two things before going any further. (1) The costs of what we present are not asymptotically optimal (or even close), nor are the theoretical properties the strongest available. (2) We developed this protocol for VEX because its costs are practical in our context (where integers have limited size, etc.); this low overhead contrasts with prior solutions (see Section 7.7).

The protocol is in two phases. First, a *prover* publishes a commitment to a non-negative integer $x$. Second, a *querier* supplies a query $q$. The protocol provides these guarantees:

- **Completeness.** If $x \geq q$, then the prover can produce a *proof* that convinces the querier that the committed value is greater than $q$.
- **Soundness.** If $x < q$, then it is computationally infeasible for the prover to convince the querier that $x \geq q$.
- **Secrecy.** No information about $x$ is leaked, other than what is implied by a proved statement.
- **Binding.** It is computationally infeasible for the prover to produce integers $x_1 \neq x_2$ and a commitment $c$ such that $c$ is a valid commitment to both $x_1$ and $x_2$.

Although the protocol appears to provide only *greater-than* proofs, we will show below how to obtain *less-than* proofs.

The main idea of the protocol is to encode an integer in the length of a hash chain, publish the tail of the chain as the commitment, and later publish an intermediate value in the chain as the proof.

Others have used hash chains [36] for various purposes, including to encode integers [41, 50]. There is also a rich literature on privacy-preserving comparisons [20, 59] (we compare our technique to others [18, 21, 22, 28] in Section 7.7). However, to the best of our knowledge, the protocol detailed below is the first to build privacy-preserving integer comparisons from hash chains.

**Details.** The protocol relies on a function $H$, which for technical reasons we model as a *random oracle*, and which we instantiate with a cryptographic hash function, such as SHA-256.[3] As notation, we write $H^i(s)$ to mean composing $H$ $i$ times: $H(\cdots H(s)\cdots)$.

Figure 2 depicts the protocol. The commitment is $H^x(s)$, where $s$ is equal to $H(s')$, for some randomly generated secret $s'$ that is distinguishable from elements in the range of $H$. In response to a query $q$, the prover returns $H^{x-q}(s)$ as the proof, or $\perp$.

This protocol provides the four guarantees above, as follows. For completeness, if $x \geq q$, then $H^{x-q}(s)$ convinces the querier: the querier applies $H$ $q$ times to the proof and obtains $H^x(s)$. For sound-

[3] A random oracle [16] maps each input to a randomly chosen output. It has been shown that random oracles cannot exist, but it is a common and accepted practice to use cryptographic hash functions as heuristic substitutes when implementing protocols.

PROVER

**function** SETUP($n$)
  $s' \xleftarrow{R} \Sigma_{n,k}$
  **return** $\{s', s = H(s')\}$

**function** GENCOMMIT($x, s$)
  **return** $c = H^x(s)$

**function** GENPROOF($q, x, s$)
  **if** $q \leq x$ **then**
    **return** $p = H^{x-q}(s)$
  **else**
    **return** $\perp$

QUERIER

**function** GENQUERY( )
  // decide on a value to query, $q$
  **return** $q$

**function** VERIFYPROOF($q, c, p$)
  **if** $p \neq \perp$ **then**
    // $p$ should be $H^{x-q}(s)$.
    **if** $H^q(p) == c$ **then**
      **return** accept
  **return** reject

**Figure 2—**A protocol for fast greater-than proofs. The random oracle $H$ has signature $H\colon \{0,1\}^* \to \{0,1\}^n$. $\Sigma_{n,k}$ is a set of seeds of length $n+k$, for some $k$, whose elements are easily distinguishable from the range of $H$. For example, $\Sigma_{n,k}$ could be $\{0\}^k||\{0,1\}^n$. The prover calls Setup and GenCommit. After receiving the commitment, the querier issues a query $q$ to the prover, who calls GenProof. The querier verifies the proof with VerifyProof.

ness, observe that if $x < q$, then the querier's algorithm could accept the proof only if it were a preimage of $s$ under $H$ (specifically, the $(q-x)$th preimage), yet by the properties of $H$, it is computationally infeasible for the prover to identify such a preimage. For secrecy, notice that the querier cannot determine $x$ from the commitment or the proof. (In fact, even if $q = x$, the querier learns only that $x \geq q$; in that case, the proof is $s$.) For binding, if the property did not hold, then the prover could identify values $x_1 \neq x_2$ and seeds $s_1', s_2'$ such that $H^{x_1}(s_1') = H^{x_2}(s_2')$, which is computationally infeasible, by the properties of $H$.

*Less-than proofs.* Our less-than proofs require a separate commitment and proof (a single commitment does not prove less-than and greater-than). The key step is to impose a maximum integer $m$, known to both parties, to which the prover can commit. Then, the parties follow the protocol in Figure 2 for $x' = m - x$, and $q' = m - q$. That is, the prover commits to $x'$, the querier issues $q'$, etc., and the statement to be proved is $x' \geq q'$, which is equivalent to $m - x \geq m - q$, which is equivalent to $x \leq q$.

**Discussion.** We discuss three points here. First, the cost of the construction above depends on the committed value. However, we describe how to amortize and avoid some of the costs in the auction context (§6). Moreover, as noted earlier in the section, the protocol is practical for our purposes, despite its unappealing *asymptotic* costs, relative to prior work (see §7.7).

Second, the protocol admits actions by the prover (and querier) that might intuitively seem like misbehavior. For instance, the prover can do the following without violating the guarantees: return $\perp$ even if $x \geq q$; commit to a very large number, thereby ensuring that it always "passes"; or in cases when both greater-than and less-than proofs are needed (which requires two separate commitments), the prover could issue commitments to different numbers. However, whether such actions are actually problematic depends on the layers above the protocol. For instance, in the ad context, we have found, to our surprise, that (a) the prover does not have much ability to cheat; and (b) we need only less-than proofs (§5.1).

Finally, the protocol presented here is not meant to be the last word: if more suitable realizations of the primitive emerge, VEX can use those too.

# 4 DESIGN OF VEX

At a high level, the purpose of VEX is to create verifiable records of low-latency, high-frequency auctions. The records are constructed so that any party to the auction, including the seller, can verify that it was conducted correctly, without learning the bids (only the sale price). This section describes our model and VEX's design, Section 5 analyzes VEX and considers limitations, and Section 6 describes optimizations.

## 4.1 Requirements and model

We adopt the following requirements for VEX:

- *Preserve existing relationships and protocols.* To the extent possible, we want our solution to "fit" in today's ad exchange ecosystem and protocols (Figure 1).
- *Preserve current privacy regime.* As stated in Section 1, our goal is to protect the *integrity* of auctions, but we should not make privacy worse. Today, bidders' privacy is discretionary (the auctioneer can leak bids; the bidder can in turn plausibly deny that it issued a given bid). Thus, for us, an audit should not require that the ad exchange disclose bidders' bids or their identities.
- *Do not undermine availability.* As with privacy, VEX should not make availability worse than in the status quo. Thus, availability can depend on the auctioneer and seller, but not on bidders; bidders should be expected to fail.
- *Avoid introducing trusted third parties.* A new trusted third party would hinder adoptability and create an additional vulnerability. We are willing to trust services that the Web depends on, such as DNS.
- *Permit bilateral verification.* An auditor should not have to deal with any party to the auction besides the auctioneer.
- *Do not burden the auction phase.* The auction phase must have low overhead (in resources and latency). However, the audit phase can be more expensive, as long as it is practical.
- *Make auditing, and its costs, optional.* To facilitate auditing, we are willing to tolerate a small cost to all participants, but no party should bear a heavy cost unless it wishes to perform an audit.

**Threat model.** We model bidders, the seller, and the ad exchange as covert adversaries [14], which are assumed malicious and can deviate arbitrarily from protocols, or collude with each other. However, they can cheat only *covertly* (for example, their reputation is valuable, or the legal ramifications of being detected are intolerable, etc.). Under this model, we consider a defense sufficient if it makes misbehavior detectable under an audit. This is obviously not the strongest possible threat model, but we believe that it matches today's ad exchange ecosystem.

**Assumptions.** We make standard cryptographic hardness assumptions. We also assume that each publisher has a well-known public key. This does *not* require a PKI, only that the publisher's public key is available in a canonical Web location, say *D*/key.pub, where *D* is the publisher's domain name. Note that relying on DNS to associate public keys and publishers does not introduce vulnerabilities in this context, as the parties *already* rely on DNS to associate *D* to the publisher itself: in deciding to respond to a bid request for *D*, a bidder assumes that the ad will appear in a Web page from *D*.

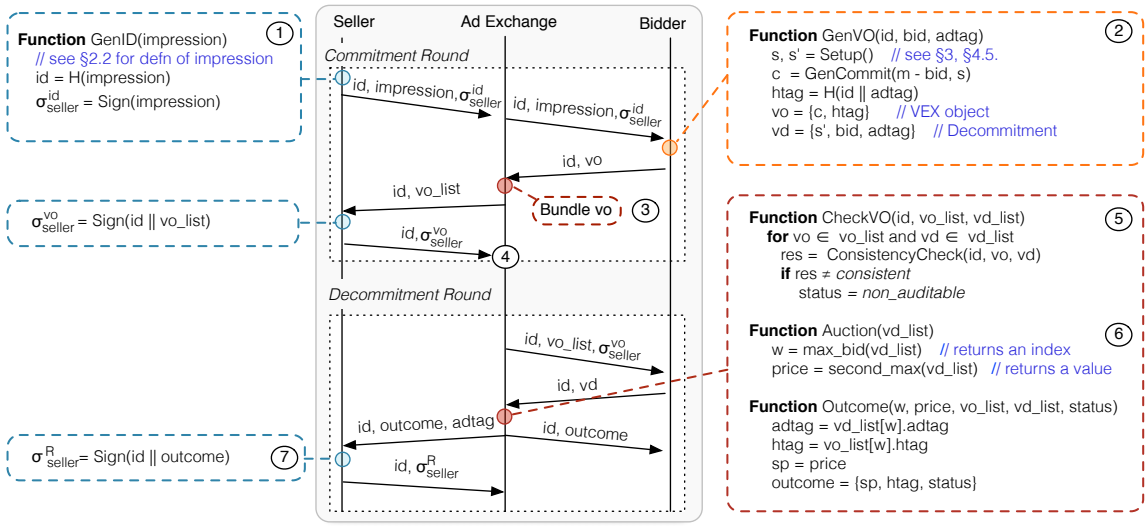



**Figure 4—**VEX's auction phase. In the first round, bidders commit to their bids; in the second round, the ad exchange computes the auction. The most relevant computations are shown; minor checks and actions are omitted. The arrows are labeled by the contents of protocol messages.

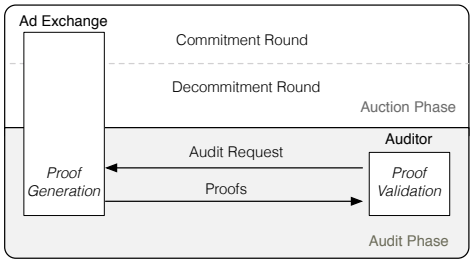



**Figure 3—**VEX is divided into two phases: an auction phase, in which the auction takes place, and an audit phase, in which an auditor verifies the correctness of the auction.

## 4.2 Overview

Besides the ad exchange, the system participants include sellers (or their representatives; see Section 2.2) and bidders (or their representatives). Also any party to the auction (the seller, a bidder, or a delegate of them) can decide to be an *auditor* of the auction.

Figure 3 depicts VEX. It is divided into two phases: the auction phase (§4.3) and the audit phase (§4.4). The auction phase has two rounds, and at a high level this structure ensures that (1) the auctioneer and seller acknowledge a bid before the bidder reveals it to the auctioneer; (2) the bidder is bound to its bid; and (3) there is an auditable record of the auction. The purpose of the audit phase is to validate an auction's correctness. We say that an auction is conducted correctly if the following condition is met:

- *Auction correctness.* The seller receives the highest bidder's ad tag, and a sale price equal to the second-highest bid (§2.2).

## 4.3 Auction phase

Figure 4 depicts the auction phase. As stated above, it proceeds in two rounds. In the *commitment round*, the seller initiates an auction by marshaling the relevant information (ad space, user, etc.; see Section 2.2) and generating a unique identifier (Figure 4, step ①). The seller transmits these contents, signed, to the auctioneer, who forwards them to the chosen bidders. Interested bidders respond with a *VEX object*, which contains a hash of the bidder's ad tag prepended with the auction's id and a commitment to the bid, intended for less-than proofs (§3) and equal-to proofs (§4.5) during the audit phase (step ②). The auctioneer bundles these VEX objects, for signing by the seller (step ③).

This operation freezes the VEX objects, creating a unique auction whose input is the bundle itself. The commitment round finishes when the auctioneer receives the signed bundle (step ④).

In the *decommitment round*, each bidder receives the signed bundle of VEX objects. After ensuring that its own VEX object is included in the bundle, a bidder decommits: it provides to the auctioneer the information needed to decode and verify its VEX object (see "vd" in GenVO, step ②).

Before continuing, the auctioneer checks that each VEX object has been constructed properly; the procedure is depicted in Figure 5. If a VEX object fails this check, then the auctioneer will be unable to pass the audit (§4.4). In this event, the auction proceeds—it is too late for the parties to revoke their VEX objects—but the auctioneer labels the auction non-auditable (step ⑤). We discuss non-auditability in Section 5.2.

The rest of the round proceeds similarly to the status quo. The auctioneer uses the decommitted bids to compute the auction, choosing a winner and the appropriate sale price (step ⑥). The auctioneer then informs the parties of the outcome. The outcome includes the sale price, the index of the winning bidder's VEX object in the VEX object bundle (indexing within the bundle is arbitrary),

```
1: function ConsistencyCheck(id, vo, vd)
2:     if vd.s' ∉ Σ_{n,k} then
3:         return inconsistent
4:     c = GenCommit(m − vd.bid, vd.s')
5:     htag = H(id || vd.adtag)
6:     if c ≠ vo.c or htag ≠ vo.htag then
7:         return inconsistent
8:     return consistent
```

**Figure 5—**Pseudocode for the consistency check. *id* is the auction's identifier, *vo* is a bidder's VEX object, *vd* is the corresponding decommitment, and *m* is the maximum bid (§7.1). This check ensures that the VEX object and the decommitment match.

```
1:  function GENERATEAUDITPROOFS(sp, B, w, S′)
2:      let P ← ∅
3:      constructed_eq ← false
4:      for i = 1 to |B| do
5:          if B_i ≥ sp and i == w then
6:              P_i.label = greater-than
7:              P_i.proof = ⊥
8:          else if B_i == sp and constructed_eq == false then
9:              P_i.label = equal-to
10:             P_i.proof = S′_i
11:             constructed_eq = true
12:         else // B_i ≤ sp
13:             P_i.label = less-than
14:             P_i.proof = GenProof(m − sp, m − B_i, H(S′_i))
15:     return P
```

**Figure 6—**Pseudocode for proof generation. *sp* is the auction's sale price, $B$ is the set of bids, $w$ is the index of the winning bidder's bid in $B$, $S'$ is the set of secret seeds, and $m$ is the maximum allowed bid. This procedure uses the integer comparison protocol described in Section 3 and an extension (Section 4.5).

a hash of the winning bidder's ad tag, and a bit indicating whether the auction can be audited.

The auction phase of the protocol completes when the seller acknowledges the outcome (step ⑦). To do so, the seller computes the hash of the winning bidder's ad tag and checks that the value included in the auction's outcome is consistent. If so, the seller sends a signature of the auction's outcome to the auctioneer, who stores it as proof of completion.

The next section describes how the auctioneer uses the received decommitments to generate proofs of correctness, and how auditors use VEX objects to validate these proofs.

### 4.4 Audit phase

An auditor begins an audit by submitting the unique identifier of the auction to the auctioneer. The auctioneer then returns the VEX object list and outcome record (which contains *sp*, the sale price, and $h^w_{tag}$, the ad tag hash of the winning bidder); both objects should have been signed by the seller. The auctioneer must also return a list of proofs in which it *labels* each bid in the VEX object list with greater-than, equal-to, or less-than; the comparison is to *sp*.

Figures 6 and 7 depict the procedures for a correct auctioneer and an auditor, respectively, to generate proofs and validate proofs. The high-level idea is that the protocol requires the actioneer to label the $|P|$ bids (see above); meanwhile for the labeling to appear valid, $|P|-2$ of the bids must be labeled with less-than (with corresponding proofs), and one bid must be labeled with equal-to (again with a correct proof). Although no greater-than proof appears, the requirement for $|P|-1$ correctly labeled bids gives the auctioneer little leeway in its claim about which proof corresponds to the winning bid. We make this reasoning rigorous in Section 5.1.

### 4.5 Extending the integer comparisons protocol

We extend the protocol described in Section 3, to allow a prover to generate proofs of strict equality. We use this extension, instead of a more natural proof-of-equality (e.g., providing the hash of a value and then later revealing the value), because the correctness argument for audits depends on there being a single commitment object for each bid (§5.1).

Figure 8 depicts the extension. With this extension, greater-than-or-equal proofs remain the same. The difference is that the prover could choose to reveal $s'$ to prove strict equality. The querier can

```
1:  function VERIFYAUDITPROOFS(sp, VO, h^w_tag, P)
2:      for i = 1 to |P| do
3:          if P_i.label == greater-than then
4:              if VO_i.h_tag ≠ h^w_tag then
5:                  return reject
6:          else if P_i.label == equal-to then
7:              if VerifyEqProof(m − sp, VO_i.c, P_i.proof) ≠ accept then
8:                  return reject
9:          else // P_i.label == less-than
10:             if VerifyProof(m − sp, VO_i.c, P_i.proof) ≠ accept then
11:                 return reject
12:     if exactly one greater-than label and exactly one equal-to label then
13:         return accept
14:     return reject
```

**Figure 7—**Pseudocode for proof verification. *sp* is the auction's sale price, *VO* is the set of VEX objects, $h^w_{tag}$ is the hash of the winning bidder's ad tag, and $P$ is the set of labeled proofs provided by the auctioneer. This procedure relies on the protocol described in Section 3 and an extension (Section 4.5).

```
function VERIFYEQPROOF(q, c, p)
    // p should be s′
    if VerifyProof(q + 1, c, p) == accept and p ∈ Σ_{n,k} then
        return accept
    return reject
```

**Figure 8—**Extension of the protocol depicted in Figure 2, to enable equality proofs. The extension overloads a greater-than or less-than commitment, to let the prover also prove equality, should it choose. The prover calls Setup and GenCommit as before, and reveals $s'$ when it wishes to prove strict equality.

then verify the proof with VerifyEqProof. This extension does not subvert the guarantees provided by the base protocol, though of course the Secrecy guarantee is vacuous under equality proofs.

## 5 ANALYSIS, REVISITING REQUIREMENTS, AND ISSUES

We consider the auction correctness condition from Section 4.2, the design requirements from Section 4.1, and other issues.

### 5.1 Auction correctness

For now, we assume that each impression is auctioned at most once and that at most one bidder is told that it won the given auction; we revisit these assumptions in Section 5.3. Our focus here will be on the *Auction correctness* condition, stated in Section 4.2. We show that, assuming a successful audit, if there is no auctioneer-bidder collusion, then the correctness condition holds. If there is such collusion, a colluding bidder receives a right of first refusal. (Auctioneer-seller collusion is considered in Section 5.3.)

As described in Section 4.4, two inputs to an audit are (1) the VEX object list for that auction (which the auditor fetches from the auctioneer and which is signed by the seller) and (2) a set of *proofs*, which the auctioneer generates in response to an audit request. To pass verification (Figure 7), the auctioneer must label one proof with greater-than, one with equal-to, and the rest with less-than. For ease of exposition, we will refer to the auctioneer as labeling *bids*; this phrasing is justified because the proof entries and committed bids are in one-to-one correspondence. For notation, let $e$ be the index of the bid labeled equal-to, $g$ be the index of the bid labeled greater-than, and $L$ be the set of indices of bids la-

beled less-than. Notice that the algorithm in Figure 7 passes only if $L \cup \{e, g\} = \{1, 2, \ldots, |B|\}$.

Now, consider the following four conditions:

- $B_e = sp$, where $sp$ is the sale price reported to the seller.
- The bidder of $B_g$ was designated the winner.
- $B_e \geq \max\{B_\ell \mid \ell \in L\}$.
- $B_g \geq B_e$.

These conditions together imply *Auction correctness* (§4.2), as follows. The first condition implies that the sale price $sp$ equals one of the bids; the second condition implies that the winner's ad tag was provided to the seller. The latter two conditions imply that the auction obeyed the second-price computation (§2.2) and its generalization to the case of ties among bids. Indeed, in the case that all of the bids are different, the latter two conditions express that $B_g$ was the highest bid and $B_e$ the second-highest.

**Claim 5.1.** If the audit verification algorithm (Figure 7) passes, then the first three conditions above hold.

*Proof.* If the first condition does not hold, then the auctioneer labeled $B_e$ incorrectly, and VerifyEqProof fails (Figure 7, line 7).

Recall that $B_g$ is defined to be the bid labeled with greater-than. If the bidder who issued $B_g$ was not the one whose ad tag hash was signed by the seller (the designated winner), then the audit algorithm rejects in line 4 (Figure 7). Thus, the second condition holds.

If the third condition does not hold, then there exists $i \in L$ such that $B_e < B_i$. But $P_i$ (the proof corresponding to $B_i$) is labeled less-than, so the verification algorithm (Figure 7, line 10) expects a proof that $B_i \leq sp = B_e$. Yet $B_i > B_e$, so the soundness of the integer comparison protocol (Section 3) implies that it is computationally intractable for the auctioneer to formulate such a proof. Thus, the audit does not pass. □

What about the last condition? The auctioneer *can* violate this condition but only under limited circumstances:[4]

**Claim 5.2.** If the audit passes but $B_e > B_g$, then the auctioneer set $sp$ equal to the highest bid and designated a winner who bid less than this amount.

*Proof.* If the audit passes, then by Claim 5.1, $sp = B_e$ and $B_e \geq \max\{B_\ell \mid \ell \in L\}$. And by the given condition, $B_e > B_g$. Thus, *all* of the bids are $\leq B_e = sp$, so this bid is the highest (and is perhaps tied for that place). Since the second condition holds, the designated winner bid $B_g$, yet the sale price is $sp = B_e > B_g$. □

Thus, if the fourth condition is violated without triggering detection, then the designated winner must have colluded, possibly tacitly, as this bidder is now obligated to pay more than its bid!—an incorrect outcome. In effect, such a colluding bidder is given a right of first refusal: it has the option to win the auction and the impression but only by paying the highest offered bid.

The guarantees, then, that the auction and audit protocols provide are: assuming a successful audit, (1) if there is no collusion, then all four conditions above hold, and hence so does *Auction correctness* (§4.2); and (2) if there is collusion, a colluding (possibly low) bidder can exercise a right of first refusal. Giving this right to a colluder is not ideal, but we consider this a (partial) victory because this right is well-understood in other contexts (law, real estate, etc.), and this is the limit of explicit manipulation.

[4]This vulnerability is a consequence of our integer comparison protocol (§3); the issue would not exist under one of the more powerful (but more expensive) alternatives in Section 7.7.

## 5.2 Revisiting the design requirements

We now consider the requirements from Section 4.1 in turn, and ask to what degree the protocol follows them.

*Existing relationships are preserved*, though slightly more information leaks out of VEX versus in the status quo: in VEX's auction phase, bidders learn how many other bidders there are, and in VEX's audit phase, the auditor learns the sale price of the auction. We do not believe that these revelations significantly alter the ecosystem *beyond* the changes introduced by verifiability, but we admit that this is speculation.

*Bidder privacy.* A bidder who creates a VEX object has plausible deniability about having done so: nothing in the protocol links a given VEX object to its bidder, the indexing of bids in the VEX object list is arbitrary, and the ad tag appears in the VEX object list only in hashed form.

*Availability.* As in the status quo, the availability of the *auction* phase depends on the seller and the auctioneer but not bidders. Bidders affect the availability of the *audit* phase: a bidder renders an auction non-auditable if it commits to a VEX object without decommitting (because then the auctioneer cannot generate proofs for the corresponding VEX object, which stymies the audit protocol). The protocol requires the auctioneer to declare a non-auditable auction *online* (though we explore the performance benefits of relaxing this requirement in Section 6). Such publicity would ideally disincentivize auctioneers from abusing the non-auditability designation. The auctioneer, in turn, could dissuade bidders from inducing this behavior, perhaps by levying penalties and eventually dropping a bidder who persistently fails to decommit. Note that although we are relying on publicity and soft defenses here, we still need VEX: without it, there is no protocol deviation to publicize.

*No new third parties* are introduced by the protocol.

*Bilateral verification* is a feature of the protocol.

The requirements that concern costs—*low burden on auction phase* and *optional auditing*—are evaluated in Section 7.

## 5.3 Other issues

*Manipulation by the auctioneer.* Because the auctioneer can invite parties, it has a fundamental ability to manipulate the auction. Nevertheless, this ability is limited. First, although the auctioneer can choose the bidders, perhaps excluding known high bidders to give other bidders favored treatment, doing so is detectable: persistently excluded bidders would notice the dearth of invitations. In the status quo, by contrast, the auctioneer can mask such malfeasance, by inviting bidders but not recognizing them as having won.

Second, the auctioneer can collude with a bidder (including a fake one). As explained in Section 5.1, such a colluding bidder can receive a right of first refusal. Third, the auctioneer can use historical knowledge of bids (but not the current auction's bids: those are hidden during the commitment round) to bid between the highest and second-highest bids. The effect is to bid up the price paid by the winner, which is similar to the right of first refusal in that, to be competitive, a bidder's price must now be higher than it otherwise would have needed to be.

*Impression delivery.* VEX does not ensure that the auctioned impression is actually delivered to the claimed user: the seller can send the ad to a different user, or even a bot. However, based on

the HTTP request that fetches the ad, the winning bidder can check that the fetching user matches the original user information (§2.2), in terms of browser, IP prefix, cookies, etc.—and from the advertiser's perspective, matches on these axes are likely "good enough". A seller could certainly spoof these fields (by imitating a legitimate user's browser, commandeering bots, etc.), but with the bidder's checks in place, a successful spoof now requires decidedly wrong (borderline criminal) behavior.

*Auction uniqueness.* VEX does not prevent the auctioneer or seller from auctioning the same impression multiple times to disjoint sets of bidders. However, that would also be an attack—a more obvious one than above—on impression delivery, and the defrauded bidder would notice that its ad was not fetched.

*Auctioneer-seller collusion.* VEX does not prevent the seller and auctioneer from colluding to produce two conflicting outcome records for the same auction, thereby convincing more than one bidder that they won that auction. But here again, only one bidder can actually receive the impression, allowing the other to suspect misbehavior. While we cannot prove that conflicting outcome records are the limit of auctioneer-seller collusion, we have been unable to identify another attack enabled by such collusion.

# 6 VEX VARIANTS: VEX-NOCC AND VEX-CP

We now briefly describe two variants of VEX. In the first variant, VEX-NOCC, the auctioneer performs the consistency check (Figure 5) the first time that an auction is audited instead of during the auction phase. VEX-NOCC dramatically reduces the auction's delay, while adding very little overhead to the audit phase. The reason is that, in generating audit proofs, the auctioneer does much of the work of the consistency check (specifically, line 14 in Figure 6 does most of the work of line 4 in Figure 5). The disadvantage of this approach is that auctions can no longer be labeled as auditable (or not) during the online auction phase. This in turn means that auctioneers might have more leeway to render auctions non-auditable.

The second variant, VEX-CP, retains the online consistency check but accelerates it, by paying additional storage costs. In VEX-CP, the auctioneer pre-computes hash chains of maximum length, storing some of the intermediate nodes as *checkpoints*; we find that having roughly 20 checkpoints leads to satisfactory performance while introducing minor storage overhead (§7.5). The auctioneer retains the seeds used to generate these chains and supplies them to bidders, as part of the first message (with the id, impression information, etc.; see Figure 4). To issue a commitment, a bidder now generates a *precommitment* from the seed, appends a random bit string as a *salt* (which hides the committed value even from the auctioneer), and hashes these to produce the commitment. Bidders decommit by revealing the salt rather than the seed.

At that point, the auctioneer can perform the consistency check quickly, by traversing the chain from the closest checkpoint, rather than from the root. Note that these checkpoints also accelerate proof generation, leading to performance benefits during the audit phase.

VEX-CP requires minor changes to VerifyProof (Figure 2), VerifyEqProof (Figure 8), and VerifyAuditProofs (Figure 7). The modified VerifyProof is depicted in Figure 9; it handles salted commitments. The modified VerifyEqProof and VerifyAuditProofs handle an extra parameter, *salt*. In addition, auditors must obtain from the auctioneer the list of all of the salts used in a particular auction.

With VEX-CP, the auctioneer can sometimes re-use chains and checkpoints across multiple auctions. We elide the details for the sake of brevity. Here is a brief summary. We assume a fixed window

```
function VERIFYPROOF(q, c, p, salt)
    if p ≠ ⊥ then
        // p should be H^{x−q}(s).
        if H(H^q(p) || salt) == c then
            return accept
    return reject
```

**Figure 9—**Refinement of secrecy-preserving integer comparisons (Figure 2) to use a salt.

of time during which an auction *could* be audited (say 1 month). If this interval expires, the auction is guaranteed not to be audited. At that point, it is safe to reuse the chain, provided it has not been used in an audited auction. Replenishing checkpoints happens in the audit phase. In particular, after an audit, the auctioneer generates a new set of chains and checkpoints, and marks all of the ones associated with the current audit non-reusable.

# 7 EXPERIMENTAL EVALUATION

This section investigates the price of the guarantees that VEX provides, by asking the following questions: (1) What is VEX's effect on throughput? (2) What is VEX's effect on latency in the online phase? (3) What are VEX's storage and bandwidth costs? (4) What is the auditing cost in absolute terms, and what is the audit/auction cost ratio? (5) What are the costs of the protocol introduced in Section 3, and of alternative schemes that could be used to implement private comparisons in VEX? We answer these questions in the context of a prototype implementation, described below.

## 7.1 Prototype implementation

We implement model applications in C++, for an ad exchange (auctioneer), a seller, and a bidder. We do not implement interactions with the user (Figure 1, steps ①, ⑦, ⑧, and ⑨). The auctioneer application is 1215 lines of C++ (measured by SLOC [12]). We also implement the two variants described in Section 6, requiring an additional 165 lines of C++. The seller model application is 659 lines of C++; the bidder, 939 lines. We also implement a model auditor in 471 lines of C++ (and 30 more for VEX-CP).

For $H$, we use an optimized implementation of SHA-256; it uses Intel's AVX instructions. The implementation is based on a release from Intel [7] and is adapted to work with our application (requiring 400 lines of assembly and C).

We perform all public key operations using the ESIGN signature scheme [42], with 2048-bit keys, and parameter $e = 8$. The seeds (§3) are 512-bit strings, where the first 256 bits form a well-known pattern (see Figure 2), and the remaining 256 bits are randomly generated. In the case of VEX-CP, the salts are 256 bits. We use integers to represent bids from \$0.01 to \$100 in \$0.01 increments (which is typically the smallest billable unit in ad exchanges [4]); thus, for the purposes of the integer comparison protocol (§3), the maximum ($m$) is 10,000. Also, VEX-CP computes a total of 20 checkpoints (§6). All of the protocol messages are based on DoubleClick's RTB protocol [3], and are extended to contain the required fields for VEX and its variants.

## 7.2 Experimental setup and method

We compare our implementation's resource use to that of a *baseline* protocol; we implemented the baseline in 1635 lines of C++, following the protocol depicted in Figure 1.

We consider CPU utilization (measured using the PAPI library [10]), auction latency (measured at application-level from the

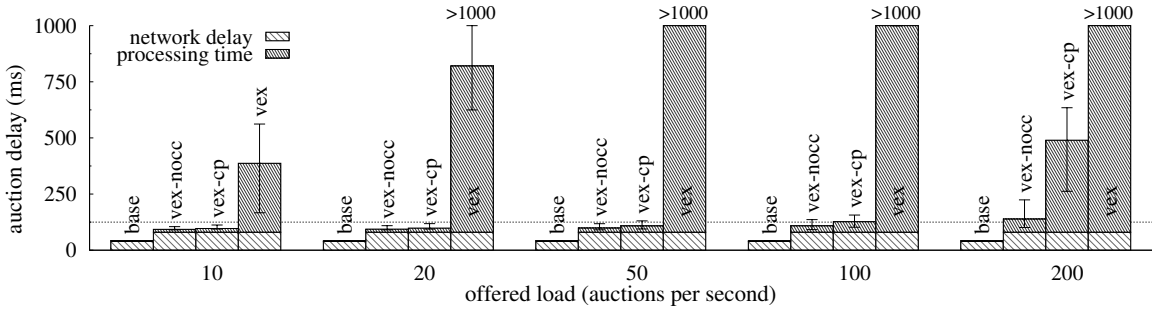



**Figure 11—**Effect of load variation on auction latency with 20 bidders. Bar heights depict median per-auction latency over 100 trials (each trial represents an end-to-end run of the entire system for 30 seconds); error bars depict 5th and 95th percentiles. Network latency is 10 ms (introducing a delay of ∼40 ms for the baseline and ∼80 ms for all of VEX's variants); see Section 6. The horizontal dashed line at 120 ms represents a standard ad exchange timeout; this is the maximum time that the ad exchange waits for bidders to respond.

| | auctions/sec (closed loop) | | |
|---|---|---|---|
| # bidders | baseline | VEX-CP | VEX-NOCC |
| 10 | 25,000 | 1,100 | 10,000 |
| 20 | 12,000 | 550 | 6,000 |
| 50 | 4,700 | 220 | 2,800 |
| 100 | 2,200 | 110 | 1,500 |

**Figure 10—**Maximum achievable auctions/sec measured at the auctioneer on a single core and reported as means over 100 trials to two significant digits. The standard deviations in all experiments are less than 10% of the means. The baseline does no cryptographic operations, so its auction computation is considerably lighter.

seller's point of view, specifically from steps ② to ⑥ in Figure 1),[5] network utilization (reported in terms of bytes sent and received at application-level), and storage (computed as bytes stored on disk).

We run all of our experiments on Utah's Emulab [58], configuring VEX, VEX-CP, VEX-NOCC, and the baseline identically. Each party (bidder, seller, auctioneer) runs on a single core. Our experiments have four physical nodes; one is dedicated to the auctioneer, two are shared by all bidders and the seller, and the remaining one is an Emulab delay node. All simulated links have fixed 10 ms latency, 1 Gbps throughput, and zero packet loss. Each node is a Dell r820 2U server, with four 2.2 Ghz 64-bit 8-core E5-4620 "Sandy Bridge" processors, 128 GB of 1333 MHz DDR3 RAM, running a standard 64-bit Ubuntu 12.04 Linux operating system.

## 7.3 Throughput

To understand throughput, we measure the capacity (i.e., maximum number of auctions per second) that the auctioneer can handle for each of our variants. In this experiment, we pre-compute and load into memory all of the inputs that the auctioneer expects from other parties (e.g., commitments, signatures, etc), and bombard the auctioneer with auction requests issued in closed loop, for 30 seconds; this stresses the auctioneer's part of the protocol in Figure 4. The experiment thus includes the cost of message serialization but not I/O and network overhead. We perform 100 trials.

Figure 10 depicts the results. VEX's results are elided for readability, but its performance is two orders of magnitude lower than VEX-CP's. The baseline is roughly 20× better than VEX-CP's implementation at higher loads. Under VEX-NOCC, however, the slowdown is only 2×. While these results might produce some sticker shock, they are actually not surprising: the baseline performs negligible work (computing the maximum of a set, etc.) while VEX and its variants handle twice as many messages and perform cryptographic operations.

We perform a similar experiment for the seller and bidder applications and find that they can sometimes be the bottlenecks in VEX's variants. Specifically, we observe that both applications are roughly 2 orders of magnitude slower than their baseline counterparts. The seller reaches a capacity of 5500 auctions per second on a single core (for 10 bidders; this number is only slightly lower for a higher number of bidders), while the bidder application can support up to 650 auctions per second on a single core. The seller's throughput is limited by the three required public key operations; each takes on average 50 $\mu$s. The limiting factor for bidders is commitment generation (§4), averaging 1.2 ms per commitment. Note that our baseline seller application performs no computations besides sending the request (Figure 1, Step ②) and logging the response (Figure 1, Step ⑥), so we are not measuring the time needed to generate the ad space and user information in these experiments.

Our results suggest that the computational bottleneck for now is the bidder application (rather than the auctioneer). This is obviously a limitation and a natural starting point for future work.

## 7.4 Latency

To investigate latency during the auction phase, we have the model seller generate requests according to a Poisson process of varying rate. We fix the number of bidders at 20.

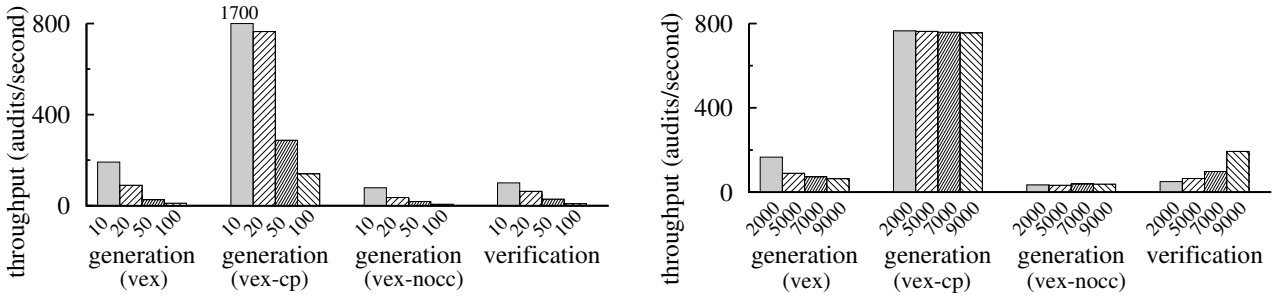
Figure 11 depicts the results. When lightly loaded ($\leq$50 requests/sec), VEX-CP and VEX-NOCC process auctions in less than 120 ms (at the 95th percentile). The latency overhead versus the baseline is 55 ms for VEX-CP and 51 ms for VEX-NOCC, the majority (∼40 ms) coming from an additional round of communication.

VEX's latency rises with offered load because the load is exceeding VEX's (anemic) capacity; VEX-CP experiences the same behavior starting at 200 auctions per second. By contrast, VEX-NOCC and the baseline can handle significantly higher loads, as they do not perform cryptographic operations along the critical path (§6).

## 7.5 Network and storage costs

Figure 12 tabulates the network and storage overhead of VEX. The majority of the overhead comes from transmitting and storing the VEX objects and the seller's signatures (§4.3). In addition, VEX-CP's auctioneer incurs a moderate local storage cost from the checkpoints of precomputed hash chains (§6).

[5] In the case of VEX and its variants, we stop the timer before the seller generates $\sigma^R_{\text{seller}}$ (see Figure 4).




(a) Audit throughput varying the number of bidders

(b) Audit throughput varying the auction's sale price

**Figure 13—**Effect of the auction's sale price and the number of bidders on the throughput of the audit phase, under the three variants of VEX (the auctioneer generates audit proofs, and auditors verify them). In figure (a), the number of bidders varies, and the auction's sale price is fixed at \$50.00 ($sp = 5000$; see §4.3); one of the bars is truncated. In figure (b), the sale price varies, and the number of bidders is fixed at 20.

| | VEX/VEX-NOCC | VEX-CP |
|---|---|---|
| **network (bytes/auction)** | | |
| seller ↔ auctioner | $804 + 64n$ | $804 + 64n$ |
| auctioneer ↔ bidder | $672 + 64n$ | $704 + 64n$ |
| **storage (bytes/auction)** | | |
| seller | $40 + 64n$ | $40 + 64n$ |
| auctioneer | $328 + 64n$ | $328 + 32n + cp$ |
| bidder | $608 + 64n$ | $608 + 64n$ |

**Figure 12—**Network and storage costs beyond the baseline. $n$ is the number of bidders participating in the auction, and $cp$ is the size (in bytes) of the checkpoint list. The checkpoint list is composed of 20 checkpoints per seed, or a total of 640 bytes per bidder ($640n$). Not depicted is an additional $64n$ bytes (for the VEX objects) that need to be stored by an auctioneer under VEX-NOCC (see Section 6).

| | commit. size | network cost | rounds |
|---|---|---|---|
| Boudot (2000) [18] | 128 bytes | 5.9 KB | 3⋆ |
| Camenisch et al. (2008) [21] | 32 bytes | 3.8 KB | 3⋆ |
| Chaabouni et al. (2012) [22] | 32 bytes | 1.7 KB | 1 |
| Fauzi et al. (2013) [28] | 32 bytes | 1.3 KB | 1 |
| VEX (§3) | 32 bytes | 32 bytes | 1 |

⋆Can be made non-interactive by applying the Fiat-Shamir heuristic [29].

**Figure 14—**Comparison of VEX's integer comparisons (§3) to zero-knowledge range proofs, in terms of storage and network costs, and number of rounds of interaction. Costs are estimated from the papers; we report the variant with the lowest communication costs.

## 7.6 Audit cost

To quantify the cost of an audit, we run our auditor application and the auctioneer (for each of the variants) independently, handing them auctions, in closed loop, to prove or verify, and measure capacity. For VEX-CP, we do not depict the cost of generating new seeds and checkpoints after an audit (§6); adding such costs would result in performance similar to that of VEX-NOCC.

We perform two sets of experiments. First, we vary the number of bidders that originally participated in the auction, holding sale price constant. Figure 13(a) depicts the results. The CPU costs of the proof generation and proof verification algorithms (Figures 6 and 7) rise linearly with the number of bidders; the reason is that the auctioneer generates a proof for every submitted bid.

Second, we vary the auction's sale price, holding constant the number of bidders. Figure 13(b) depicts the results. Sale price has little effect on the work performed by the auctioneer (proof generation) in the optimized variants of VEX but significantly impacts an auditor's throughput. The reason is that the principal cost for both parties is hash operations. Meanwhile, under VEX-NOCC, the auctioneer must perform, for each bid $b_i$, $(sp-b_i)+(m-sp) = (m-b_i)$ hash operations, which is independent of $sp$. Under VEX-CP, the number of hash operations performed by the auctioneer in the audit phase is upper-bounded by $\sum_{i=1}^{n} \min\{\frac{m}{20}, sp - b_i\}$, and $m/20$ is the controlling component, in our experiments. By contrast, the auditor must perform $m - sp$ hash operations for each less-than proof, plus $m + 1 - sp$ hash operations for the equal-to proof.

Based on these results, the auctioneer can sustain audits at roughly 1/18 and 1/160 the rate at which it can sustain auctions under VEX-CP and VEX-NOCC, respectively (we ignore VEX as it can actually perform more audits than auctions!). This means that with 33% (say) additional resources over its auction-handling resources, the auctioneer could support an audit rate of 1 in 24 and 1 in 480 auctions for VEX-CP and VEX-NOCC, respectively.

## 7.7 Integer comparison protocol

We now compare the protocol in Section 3 to *zero-knowledge range proofs* (ZK:RP) [18, 21, 22, 28], which, like the protocol in Section 3, provide privacy-preserving integer comparisons. However, a ZK:RP provides stronger properties than VEX (it can answer queries about *ranges* versus inequalities), but here our focus is on costs: the size of commitment, the length of proofs, and the number of rounds of interaction. Figure 14 depicts the comparison. VEX is relatively lightweight in terms of commitment size and proof size, and it requires only one round of communication.

Of course, another important axis for comparison is computational (CPU) cost of commitment generation, proof generation, and proof verification. However, it is difficult to estimate the cost of the alternatives because most of them have not been implemented, and do not appear in the literature in a way that exposes all of their constants. It is clear that their asymptotic performance is superior to VEX's, though we think VEX is likely to perform better over the range that it deals with (numbers less than 10,000; see Section 7.1). Indeed, commitment generation, proof generation, and proof verification all take roughly 1 ms in this range. In contrast, a recent implementation of Boudot's scheme [38] requires relatively large proofs (5.3 KB), and proof generation and verification take 74 and 31 ms, respectively.

### 7.8 Summary and outlook

Our evaluation indicates that the additional latency imposed by VEX and its variants is fairly low (tens of milliseconds per auction) and comes primarily from an additional round of communication. Network costs are moderate (hundreds of bytes per bid), storage costs for the auctioneer are considerable (but still tolerable), and auctioneer throughput is reasonable (one of the optimized variants of VEX imposes overhead of 2×). The audit-to-auction ratio (in terms of the expense of an audit) is 1 for every 18 auctions for VEX-CP, and 1 for every 160 auctions for VEX-NOCC. This ratio could in principle permit a significant fraction of auctions to be audited. The principal cost during the auction phase is the bidders' commitment generation. This cost is comparable to proof generation and verification in the audit phase, and accounts for about 1.2 ms of CPU time.

## 8 OTHER RELATED WORK

We covered work on hash chains and zero knowledge range proofs in Sections 3 and 7.7. Below, we discuss three categories of research: online advertisement, secure auctions, and general-purpose auditing.

**Online advertising.** Privad [31], Adnostic [51], and work by Juels [35] apply to ad *networks*, versus our concern of ad *exchanges*. Also, their focus is privacy for Web users, versus our concern of integrity in auctions. Reznichenko et al. [47] develop auction protocols, but again the focus is ad networks, and the goal is user privacy.

Recent work by Dave et al. [23] and Haddadi [32] focuses on measuring, detecting, and preventing fraudulent clicks in online ad networks; this work could complement VEX by helping to defend against attacks on impression delivery (§5.3). Stone-Gross et al. [49] characterize the different types of fraudulent activity in ad exchanges, such as click fraud and keyword stuffing, but unlike VEX, their work assumes the correctness of the ad exchange itself. Lastly, Muthukrishnan [39] describes ad exchanges generally, and poses nine theoretical problems that range from mechanism design to the development of a verifiable ad exchange. We address the latter problem with VEX.

**Secure auctions.** A large body of work in secure auctions focuses on privacy [19, 34, 40]. This work is valuable but orthogonal to the focus of this paper. Other work [17, 30, 37, 44–46, 50] recognizes the importance and the challenge of designing verifiable online auctions, and proposes several schemes based on novel cryptographic techniques and distributed protocols. We provide a summary of the most relevant work below.

The work by Rabin et al. [45] is the most recent in a line of work [44, 46] that aims for auction integrity. This work achieves stronger guarantees than VEX (for instance, unlike VEX, their scheme does not require disclosing the auction's sale price to verify that an auction was conducted properly), and reports impressive performance results. On the other hand, as the authors acknowledge, the performance results correspond to a construction with soundness (probability of detecting an incorrect outcome) of only 25%. Amplifying this rate would require bidders to submit a large number of commitments, leading to large computational, storage, and communication costs.

Some work [30, 37, 50] proposes the design of secure auctions by splitting up the role of the auctioneer into multiple parties. Note that some of these schemes prevent misbehavior (versus detecting it, as in VEX). The tradeoff is that they require a non-collusion assumption among some of the parties. This is a reasonable assumption in many instances, but the centralized nature of ad exchanges makes this work not directly applicable to our requirements (§4.1).

**Auditing.** General-purpose auditing systems, such as PeerReview [33], provide strong guarantees that would benefit ad exchanges, but they require participants (and their public keys) to be well-known. The more specialized SPIDeR [61], though not directly applicable to auctions, has a similar ethos to VEX: SPIDeR uses low cost, weak cryptographic primitives and couples them with careful design to obtain security guarantees.

## 9 DISCUSSION AND CONCLUSION

As mentioned in the introduction and described in Section 5, VEX has limitations. We now return to two of them. First, VEX has undeniable overhead. However, it is not absurd and in fact is consistent with the price of cryptographic guarantees in systems.

Second, the auctioneer can manipulate auctions, by choice of participants, by colluding with a bidder, or by labeling auctions non-auditable. However, under our analysis, this is *all* that the auctioneer can do, and given the nature of the problem—that there are billions of latency-sensitive auctions per day, that none of the participants know each other, that the integer comparison protocol (§3) gives weak guarantees, that the auctioneer seems to hold all of the cards, etc.—we were not sure that we could provide any guarantees at all. So we have been pleasantly surprised that the guarantees (§4.2, §5.1) appear strong enough for real use.

In fact, a natural avenue of future work is to broaden VEX's applicability: to other ads, to other auctions, and to other contexts. Concerning other ads, we have assumed impression ads; we would like to know if VEX applies to cost-per-click ads and other types. Concerning other auctions, we expect that VEX's core protocol, under generalizations, can verify $k$-price auctions (winning bidder pays $k$-highest bid), generalized Vickrey auctions ($m$ items are allocated to $m$ bidders, who pay the $m-1$-highest bid), and modified second-price auctions (the highest bidder pays the second-highest bid plus a fixed amount). And concerning other contexts, we have phrased the core protocol in terms of ads (Figures 4–8), but we believe that the design is mostly independent of this context. With minor modifications, it should apply to eBay, online funding platforms, and other online auctions.

This is not to say that misbehavior is rampant in these contexts, just that verifiability would be a Good Thing: it would strengthen the service given to auction participants while lowering the barrier to entry for new auctioneers, by reducing the trust required.


### Acknowledgments

This paper was improved by comments from Allen Clement, Alan Dunn, Lon Ingram, Arvind Krishnamurthy, Josh Leners, Srinath Setty, Emmett Witchel, Edmund L. Wong, and the anonymous reviewers; and by conversations with Bobby Bhattacharjee, Amar Goel, Saikat Guha, and Jon Howell; Maxwell Krohn brought ad exchanges to our attention. We thank our shepherd, Dave Levin, for his detailed attention, and abhi shelat for a particularly careful read that identified several problems and for suggesting fixes. The research was supported by AFOSR grant FA9550-10-1-0073 and by NSF grants 1055057 and 1040083, a Sloan Fellowship, and an Intel Early Career Faculty Award.

# REFERENCES


[1] *Blacks Law Dictionary*. 6th edition, 1990.

[2] Cookie Matching. `http://developers.google.com/ad-exchange/rtb/cookie-guide`, Apr. 2013.

[3] DoubleClick Ad Exchange Real-Time Bidding Protocol. `https://developers.google.com/ad-exchange/rtb/downloads`.

[4] DoubleClick Bid Request Message. `https://developers.google.com/ad-exchange/rtb/downloads/realtime-bidding-proto.txt`.

[5] DoubleClick Ad Exchange. `http://google.com/doubleclick/advertisers/ad_exchange.html`.

[6] Google Display Network. `http://google.com/ads/displaynetwork`.

[7] Fast SHA-256 implementations on Intel architecture processors. `http://download.intel.com/embedded/processor/whitepaper/327457.pdf`.

[8] Microsoft Media Network. `http://advertising.microsoft.com/advertise/microsoft-media-network`.

[9] OpenX. `http://www.openx.com`.

[10] PAPI. `http://icl.cs.utk.edu/papi/index.html`.

[11] Rightmedia. `http://www.rightmedia.com`.

[12] Source lines of code counter. `https://github.com/bytbox/sloc`.

[13] Yahoo! Advertising Solutions. `http://advertising.yahoo.com`.

[14] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[15] H. Beales. The value of behavioral targeting. `http://www.networkadvertising.org/pdfs/Beales_NAI_Study.pdf`, 2010.

[16] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, Nov. 1993.

[17] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, Feb. 2009.

[18] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, May 2000.

[19] F. Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5(4):201–216, Oct. 2006.

[20] E. Brickell, D. Chaum, I. Damgård, and J. V. de Graaf. Gradual and verifiable release of a secret. In *CRYPTO*, Aug. 1987.

[21] J. Camenisch, R. Chaabouni, and a. shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, Dec. 2008.

[22] R. Chaabouni, H. Lipmaa, and B. Zhang. A non-interactive range proof with constant communication. In *Financial Cryptography and Data Security*, Feb. 2012.

[23] V. Dave, S. Guha, and Y. Zhang. Measuring and fingerprinting clickspam in online ad networks. In *ACM SIGCOMM*, Aug. 2012.

[24] DoubleClick. Search before the purchase: Understanding buyer search activity as it builds to online purchase. `http://google.com/doubleclick/research/index.html`, Feb. 2005.

[25] DoubleClick. Profiting from non-guaranteed advertising: The value of dynamic allocation and auction pricing for online publishers. `http://www.google.com/doubleclick/pdfs/DC_Ad_Exchange_WP_100713.pdf`, 2010.

[26] DoubleClick. Google white paper: The arrival of real-time bidding and what it means for media buyers. `http://google.com/doubleclick/research/index.html`, July 2011.

[27] E. Elkind and H. Lipmaa. Interleaving cryptography and mechanism design: The case of online auctions. In *Financial Cryptography*, Feb. 2004.

[28] P. Fauzi, H. Lipmaa, and B. Zhang. New non-interactive zero-knowledge subset sum, decision knapsack and range arguments. Cryptology ePrint Archive, Report 2012/548, Feb. 2013.

[29] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO*, Nov. 1986.

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

[31] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In *NSDI*, Mar. 2011.

[32] H. Haddadi. Fighting online click-fraud using bluff ads. *ACM SIGCOMM CCR*, 40(2):22–25, Apr. 2010.

[33] A. Haeberlen, P. Kuznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *SOSP*, Oct. 2007.

[34] M. Harkavy, J. D. Tygar, and H. Kikuchi. Electronic auctions with private bids. In *USENIX Workshop on Electronic Commerce*, Aug. 1998.

[35] A. Juels. Targeted advertising... and privacy too. In *RSA Security Conference, The Cryptographers' Track*, Apr. 2001.

[36] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[37] H. Lipmaa, N. Asokan, and V. Niemi. Secure Vickrey auctions without threshold trust. In *Financial Cryptography*, Mar. 2002.

[38] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security Symposium*, Aug. 2010.

[39] S. Muthukrishnan. Ad exchanges: Research issues. In *Workshop on Internet and Network Economics*. Dec. 2009.

[40] M. Naor, B. Pinkas, and R. Summer. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, Nov. 1999.

[41] K. Q. Nguyen, Y. Mu, and V. Varadharajan. Digital coins based on hash chain. In *National Information Systems Security Conference*, Oct. 1997.

[42] T. Okamoto and J. Stern. Almost uniform density of power residues and the provable security of ESIGN. In *ASIACRYPT*, Nov. 2003.

[43] OpenX. Openx handles more than one trillion ad requests in 2011. `http://www.openx.com/content/openx-handles-more-one-trillion-ad-requests-2011`, 2011.

[44] D. C. Parkes, M. O. Rabin, S. M. Shieber, and C. A. Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. In *International Conference on Electronic Commerce*, Aug. 2006.

[45] M. O. Rabin, Y. Mansour, S. Muthukrishnan, and M. Yung. Strictly-black-box zero-knowledge and efficient validation of financial transactions. In *International Colloquium on Automata, Languages and Programming*, July 2012.

[46] M. O. Rabin, R. A. Servedio, and C. A. Thorpe. Highly efficient secrecy-preserving proofs of correctness of computations and applications. In *IEEE Symposium on Logic in Computer Science*, July 2007.

[47] A. Reznichenko, S. Guha, and P. Francis. Auctions in do-not-track compliant internet advertising. In *ACM CCS*, Oct. 2011.

[48] N. Singer. Your online attention, bought in an instant. `http://www.nytimes.com/2012/11/18/technology/your-online-attention-bought-in-an-instant-by-advertisers.html`, Nov. 2012. The New York Times.

[49] B. Stone-Gross, R. Stevens, R. Kemmerer, C. Kruegel, G. Vigna, and A. Zarras. Understanding fraudulent activities in online ad exchanges. In *IMC*, Nov. 2011.

[50] K. Suzuki, K. Kobayashi, and H. Morita. Efficient sealed-bid auction using hash chain. In *Information Security and Cryptology*, Dec. 2000.

[51] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *NDSS*, Feb. 2010.

[52] J. Trevathan and W. Read. Undesirable and fraudulent behavior in online auctions. In *SECRYPT*, Aug. 2006.

[53] V. Vaidya. Cookie synching. `http://www.admonsters.com/blog/cookie-synching`, Apr. 2010.

[54] N. Vratonjic, J. Freudiger, and J.-P. Hubaux. Integrity of the web content: The case of online advertisement. In *Workshop on Collaborative Methods for Security and Privacy*, Aug. 2010.

[55] N. Vratonjic, J. Freudiger, J.-P. Hubaux, and M. Felegyhazi. Securing online advertising. Technical Report LCA-REPORT-2008-017, Laboratory for Computer Communications and Applications, École Polytechnique Fédérale de Lausanne, July 2008.

[56] N. Vratonjic, J.-P. Hubaux, M. Raya, and D. C. Parkes. Security games in online advertising: Can ads help secure the web? In *Workshop on Economics of Information Security*, June 2010.

[57] K. Weide. Real-time bidding in the United States and Western Europe, 2010–2015. `http://info.pubmatic.com/rs/pubmatic/images/IDC_Real-Time%20Bidding_US_Western%20Europe_Oct2011.pdf`, Oct. 2011.

[58] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, Dec. 2002.

[59] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS*, Oct. 1986.

[60] ZenithOptimedia. Quadrennial events to help ad market grow in 2012 despite economic troubles. `http://www.zenithoptimedia.com/about-us/press-releases/zenithoptimedia-adspend-forecast-update-dec-2011`, Dec. 2011.

[61] M. Zhao, W. Zhou, A. J. Gurney, A. Haeberlen, M. Sherr, and B. T. Loo. Private and verifiable interdomain routing decisions. In *ACM SIGCOMM*, Aug. 2012.