VERIFIABLEAUCTION:
AN AUCTION SYSTEM FOR A SUSPICIOUS WORLD

Michael Rosenberg
42micro@gmail.com
Ramaz High School

Dennis Shasha
shasha@courant.nyu.edu
Computer Science Department, Courant Institute, New York University

New York University Computer Science Technical Report 2014-971

Abstract

This paper presents a cryptosystem that will allow for fair first-price sealed-bid auctions among groups of individuals to be conducted over the internet without the need for a trusted third party. A client who maintains the secrecy of his or her private key will be able to keep his/her bid secret from the server and from all other clients until this client explicitly decides to reveal his/her bid, which will be after all clients publish their obfuscated bids. Each client will be able to verify that every other client's revealed bid corresponds to that client's obfuscated bid at the end of each auction. Each client is provided with a transcript of all auction proceedings so that they may be independently audited.

1 Introduction

Thanks to eBay™ and similar sites, auctions happen millions of times a day. In most cases, a trusted third party knows everything about the bidding as well as the identity of the bidders. One can easily imagine communities of people who would like to avoid third parties both to save expense and to ensure privacy. This could apply to bids for a top secret project or for a high-priced piece of art. For the sake of concreteness and for its emotive appeal, we imagine a wealthy family that wants to auction off several items of an estate.

VerifiableAuction allows a community to conduct an auction without the need for a third party and with the additional assurances that (i) neither client nor auctioneer can see any other client's bid until the other client chooses to reveal it and (ii) collusion between the auctioneer and a client conveys no benefit to either.

2 Terminology

An *Auction* is a protocol governing the sale of a single item. The *Phases* of bidding are the steps each client must take in the bidding process (our system requires two Phases per auction). An *Estate* is a series of Auctions on individual items held among some group of people: *members*, also known as *bidders*. For each Estate there is an *Estate Administrator* as well as the members of the Estate (the Estate Administrator is also considered a member). The *Server* is the central location and computer where information about Estates is stored and where auctions are conducted. Corresponding to each *Server* there is a *Global Administrator*. The client software supports participation in multiple Estates over multiple Servers.

3 Specification

Our cryptosystem relies upon the following primitives:

Message Authentication Code Function	$\textsc{mac}_k(\text{msg})$
Hash Function	HASH
Public-Private Key Digital Signature Algorithm	$\textsc{sign}_k(\text{msg})$, $\textsc{ver}_k(\text{msg}, \text{sig})$
Cryptographically Secure Pseudorandom Number Generator	CSPRNG

Every member uses client-side cryptographic software. The advantage is that the clients don't need to trust the Server. The disadvantage is that the clients need to put more effort into the entire process than they would when using eBay,™ for example.

3.1 Setup

The Global Administrator (e.g., the owner of the Server) may create a new Estate and a new Estate Administrator. Alternatively, an unregistered user may register and create an Estate, thereby making himself an Estate Administrator. The Estate Admin then proceeds to add members to the Estate by username. Every member is expected to generate a new keypair corresponding to the particular Estate and disseminate his/her respective public key to all other members of the Estate, preferably in a secure, out-of-band, manner, e.g., by sending it over email. The Server is not expected to know of any information about the members other than their usernames. We considered allowing the Server to disseminate the members' public keys, but determined that this could allow the Server to pretend that a bidder was present whereas in fact he/she wasn't.

3.2 Auction Process

The Estate Admin constructs a list determining the ordering of auctions. The list comprises each item's name, description, and an ID that is randomly generated by the Estate Admin. Each item ID must be unique within the Estate. If it is not the case, the client software will report a warning to the user upon Estate entry. The user who receives this error is to report it immediately to the Estate Admin as well as to all other members of the Estate, requesting a new item list with unique item IDs.

Bidding in an auction is broken up into two phases. In Phase #1, members place bids in an obfuscated but irrevocable manner. The data submitted here is called a *bid commitment*. In Phase #2, members submit data to reveal their bids and allow everyone to confirm that the submitted bid accurately reflects their commitment. This is called a *bid decommitment*.

During Phase #1, each member of the Estate must first generate a new random string of bytes p henceforth known as the *Prekey*. The Prekey and the auction item's name, description, and ID are all hashed, concatenated and hashed again to produce the key to be used with the MAC function:

$$k = \textsc{hash}(\textsc{hash}(\text{name}) \parallel \textsc{hash}(\text{desc}) \parallel \textsc{hash}(\text{id}) \parallel \textsc{hash}(p)).$$

The member then computes and submits $s = \textsc{sign}_i(\textsc{mac}_k(b))$ where i is the member's private key and b is the member's bid.

The Server institutes a waiting period following the end of Phase #1 that allows every member of the Estate to download the bid commitment data of every other member. Each member inputs this block of information into his/her accompanying VerifiableAuction client-side software for local safekeeping. The Server is to wait until each member has acknowledged that the commitment data has been saved before moving on to Phase #2.

During Phase #2 the information necessary to authenticate the bid commitment data entered in Phase #1 is submitted. The member simply submits b and p to the Server. With this information and the public key of the bidder, any other member of the Estate can reconstruct k as above and recompute $z = \textsc{mac}_k(b)$. To audit the auction, each member can test for each other member whether $\textsc{ver}_u(z, s)$ (which is equivalent to the statement $s = \textsc{sign}_i(z)$) succeeds, where u is the public key of the member whose bid is being audited. If this verification succeeds for all members, then the obfuscated bids submitted in Phase #1 correspond to the revealed bids of Phase #2.

If at any point in the protocol one member fails to submit valid data or submit data at all, every other member of the Estate must wait until that member complies. Furthermore, if any member announces that something is not correct, the auction must be restarted. Note that the system cannot determine the perpetrator of the invalidity or determine whether the whistle-blower is telling the truth. Fortunately, each member has an incentive both to behave honestly and to check on others.

4 Analysis

4.1 Assumptions

To formalize VerifiableAuction's correctness, we must first formalize the assumptions that we make about the underlying system:

(1) All cryptographic primitives are secure for their use in the protocol.

(2) No private key is known to anyone other than the owner of the key.

(3) All keys and random strings are sufficiently long for their appropriate secure use.

4.2 Guarantees

VerifiableAuction makes a number of specific guarantees that relate to user privacy and integrity:

(1) No member can retrieve the bid amounts of other members, i.e., before the beginning of Phase #2.

(2) No member can tamper with bid commitments after the end of Phase #1.

(3) If the server omits bids from any subset of Estate members, members will be able to detect the omission.

(4) No member may deny his/her bid commitment after submission.

(5) No member can forge bids or bid commitment data of other members.

4.3 Design Rationale

The goal of this cryptosystem is to ensure that no party under any circumstance should have an unfair advantage over the other participants. Any and all attempts at obtaining such an advantage are detectable though not necessarily traceable to the perpetrator.

We begin with the use of the MAC function. Recall that a MAC, or message authentication code, function is a one-way function that computes the digest of an input of arbitrary length given a fixed-length key. The use of a one-way function ensures that, given $\text{MAC}_k(s)$, someone without the key k is unable to calculate $\text{MAC}_k(s')$ for any $s' \neq s$. A MAC function is often used for proving and authenticating the integrity of data. The key used in our MAC function is a mix of random data and the auctioned item's metadata. The reason for the use of the item's name and description in the construction in the key is to ensure that the bid is valid only for that particular item. The protocol includes a randomly generated item ID in case a name and description are repeated for any reason. Furthermore, the construction $\text{HASH}(\text{HASH}(*)\|\text{HASH}(*)\ \ldots)$ is used as opposed to $\text{HASH}(*\|*\|\ \ldots)$ in order to prevent potential collisions. If a suffix of the item name were removed and prepended to the item description, the value of (name || desc) would remain the same, and thus the digest of it would remain the same as well. This possibility is eliminated by using the concatenated hashing construction.

The choice of using a MAC instead of a symmetric cipher for committing to a datum is, to an extent, arbitrary. However, the use of a MAC has a few advantages [2]. First, the key size is not bounded. If it exceeds the block size of the underlying hashing algorithm, it is hashed and the result is used as the key. Second, the block size can be relatively easily changed by swapping out the underlying hashing algorithm. As an example of the flexibility this allows us to have, note that SHA2, offers more choices in block size than AES (the former supporting 224, 256, 384, and 512 bit block sizes, and the latter supporting 128 and 256 bit block size).

The reason all Estate members must wait for every member to comply is that it is inherently impossible to determine the cause of non-responsive member (whether it be a lack of response or submission of invalid data) or that the Server itself is misbehaving.

Many modifications to the protocol are possible. Instead of committing and decommitting for each auction individually, Estate members could instead commit to every auction in the Estate and decommit only after the

commitment data for all auctions had been entered. This would require only that the Prekey be generated with the Estate ID instead of the individual item IDs and that the Prekey be reused in the commitment operation on each auction item.

4.4 Correctness

Here we demonstrate that all guarantees are satisfied as stated above using proof sketches.

No member can retrieve the bid amounts of other members, i.e., before the beginning of Phase #2.

The only information about a bid that any member of an Estate other than the bidder has access to is $\text{MAC}_k(b)$ where k is the appropriate key and b is the bid amount. Retrieving b from this entails retrieving a known input to a MAC with an unknown key. Based on assumption 1, we assert that the cryptosystem is secure against this threat.

No member can tamper with bid commitments after the end of Phase #1.

Tampering with bidding information on the Server before all members have downloaded the bidding information (recall that Phase #2 does not commence until all members have acknowledged that they have downloaded the bidding information) is possible but could not benefit any member. If a member were to tamper with his own commitment data on the Server (implying that the member has access to the private key used to sign the commitment), that action is equivalent to the member's revising his initial bid commitment. If, on the other hand, one member m_1 were to tamper with another member's m_2's commitment data before any other member has the ability to download it from the Server, Phase #2 would reveal that member m_2's bid does not match the commitment data provided. Since the standard practice for an event like this is to repeat the auction (with a different ID), no member has gained an advantage other than learning what the previous bids were.

The only way an attacker would be able to hide the fact that a bid commitment was tampered with would be if the attacker were able forge a valid commitment with the private key of the target member. This violates assumption 2. Furthermore, if the attacker were to strike after a subset of Estate members had downloaded the commitment data but before a different, non-empty subset of members did the same, the members of the estate could convene and determine that the commitment data was tampered with.

It is also possible that the Server can unilaterally skip the waiting period (whereby all members of the Estate download the commitment data and acknowledge that they have done so) after Phase #1 and move on to Phase #2. The onus is upon the members in this case to blow the whistle on the Server and demand the data before continuing. The members would see that some bids are missing, so the Server's actions would be detected.

If the server omits bids from any subset of Estate members, members will be able to detect the omission.

When commitment data is imported into the client software, the software performs a check to ensure that every member's bid commitment has been submitted. If this is not the case, the member's software presents a visible warning of just that to the member. It is then, again, the responsibility of the member to inform the other members of the Estate as well as the Estate Admin that there is missing data.

No member may deny his/her bid commitment after submission.

Every bid commitment is signed with the corresponding member's private key. This commitment can be verified given the member's public key which was distributed at the setup of the Estate. Commitment denial is defined as the assertion that a signed commitment that has been verified was not signed by the member in question. This claim is equivalent to the claim that, given a signed message $s = \text{SIGN}_i(z)$ and a public key u_1, one can generate a new public key u_2 such that $\text{VER}_{u_1}(z, s)$ and $\text{VER}_{u_2}(z, s)$ both succeed. Under assumption 1, this is

computationally infeasible. Alternatively, this claim is also equivalent to a member's performing a signature operation with another member's private key, which violates assumption 2.

No member can forge bids or bid commitment data of other members.

[For brevity, we define $\textsc{g}(\text{name}, \text{desc}, \text{id}, p, b) = \textsc{mac}_{\textsc{hash}(\textsc{hash}(\text{name}) \,\|\, \textsc{hash}(\text{desc}) \,\|\, \textsc{hash}(\text{id}) \,\|\, \textsc{hash}(p))}(b)$.] There are multiple ways an Estate member could go about forging another member's data. Suppose, firstly, that Mallory (the would-be forger) has managed to circumvent the login system, allowing him to login as another member, Alice. If the current auction is in Phase #1, Mallory may submit his own commitment data, $c = \textsc{sign}_m(\textsc{g}(\text{name}, \text{desc}, \text{id}, p_1, b_1))$, for some private key m, to the Server without Alice's knowledge or consent. Suppose this action was unnoticed by both the Server and Alice. The problem remains that regardless of what Mallory submitted, it could not have possibly been signed with Alice's private key (under assumption 2). By Phase #2, Mallory will be forced to enter p_2 and b_2 (the Prekey and bid amount, respectively). Note that these values do not have to be the same as the values chosen for the commitment generation. Passing verification would require Mallory to solve for m, p_1, p_2, b_1, b_2 such that c is a valid signature of $\textsc{sign}_m(\textsc{g}(\text{name}, \text{desc}, \text{id}, p_2, b_2)$ under Alice's key a. This should not be feasible under assumptions 1 and 3. Furthermore it would be even more difficult for Mallory if he were to fix a particular b_2 as Alice's public perceived bid amount.

It may also be possible that Mallory submits only his own data in Phase #2. In this case, Mallory would be forced to solve for b_2 and p_2 such that Alice's commitment, $A = \textsc{sign}_a(\textsc{g}(\text{name}, \text{desc}, \text{id}, p_1, b_1))$, is a valid signature of $\textsc{sign}_m(\textsc{g}(\text{name}, \text{desc}, \text{id}, p_2, b_2)$. This is at least as difficult as the previous attack, and thus infeasible as well.

Mallory also has the ability to copy Alice's bidding information from previous auctions into a new auction. That is to say that the p and b are known to Mallory before the attack even begins. This is known as a replay attack and it is prevented by the use of unique item names and descriptions as well as unique item IDs for every auction. If IDs and item name and description were equal, that is, when $\text{id}_1 = \text{id}_2$, $\text{name}_1 = \text{name}_2$, and $\text{desc}_1 = \text{desc}_2$, then the solution for p_2 and b_2 in $\textsc{sign}_m(\textsc{g}(\text{name}_1, \text{desc}_1, \text{id}_1, p_1, b_1) = \textsc{sign}_m(\textsc{g}(\text{name}_2, \text{desc}_2, \text{id}_2, p_2, b_2)$ is trivial: $p_2 = p_1$ and $b_2 = b_1$. If the auction that p_1 and b_1 belong to has already ended, then these values are known to every member and can be replayed. Solving this problem when all three values (ID, name, and description) are unique, however, is infeasible.

5 Comparison

The two-part bidding process used here is most similar to that of VEX [1] insofar as they both feature a commitment stage and a revealing stage. The essential difference is how the bids are revealed and to whom. VEX specifies that every user should share his bid amount (more specifically, decommitment data, from which the bid amount can be trivially derived) with the auctioneer and that the auctioneer should keep the bid amount secret from the rest of the bidders. VerifiableAuction specifies that the Server is to collect all of the bid amounts and display them for every user to see.[1]

One of the core differences between VEX and VerifiableAuction is the trust model. The VerifiableAuction system explicitly specifies that users need not trust the Server for any operations during the course of the auctioning process. The only form of communication that requires an inherent level of trust is the out-of-band communication that auction members conduct when setting up an Estate for the first time. The onus is thus placed on the users to ensure that the exchange of item lists, public keys, etc. is trustworthy (in that it is forgery- and tampering-resistant). The trust model of VEX is largely dependent on the auctioneer. Every user of VEX

[1] We choose to not support private integer comparison as it would require a trust in the Server that can fairly easily be violated, i.e., the trust that the Server keeps all bid amounts secret and only releases the necessary information to each member of an Estate to prove that his/her bid was less than the winning bid. The alternative to private integer comparison is an interactive protocol. We did not include interactive protocols in our specification due to the inherent complexity of performing a peer-to-peer interactive protocol over the internet with no additional provided infrastructure.

must implicitly trust that the auctioneer will not tamper with or omit data. VEX proposes a sociological solution: bidders who have repeatedly suspected wrongdoing on the part of a particular auctioneer will simply avoid participating in that auctioneer's future auctions.

There are a few corollaries to this inherent difference in trust. Since VERIFIABLEAUCTION requires that every user submit a bid because omission could indicate misbehavior on the part of the Server, it is a trivial task for a user to intentionally or even unintentionally perform a denial of service (DOS) attack on the Estate. If the user simply refuses to submit any kind of commitment or decommitment data, the rest of the Estate has no choice but to wait or start a new Estate where the attacker is no longer a member. However, the benefit of the bid requirement is that if every user submits a bid and they are all shown in obfuscated form, then nobody can tamper with or ignore any bid.

By contrast, since the auctioneer is implicitly trusted in the VEX system, the auctioneer is free to ignore bids, thus circumventing the DOS attack above. On the other hand, since the auctioneer has this unchecked power, the auctioneer can quite easily collude with a user to produce an auditable record of bids that is ostensibly valid but has in fact been tampered with to show that the colluding user has won the auction. Furthermore, in the case that a bid decommitment is not received by the auctioneer, the auctioneer has the ability to ignore the bidder, continue the auction, and publish the results without granting the bidders the ability to perform an audit on the auction (this is done in VEX by marking the auction as *unauditable*). In such a case, the bidders have no recourse whatsoever regarding the verifiability of the auction; the auctioneer must be trusted.

Another corollary of the difference in trust systems is bid deniability. VEX ensures that there is nothing linking a bid with the bidder. This is to say that even if a user obtained all bid amounts, there would be nothing linking them to the bidders themselves. VERIFIABLEAUCTION by contrast employs the use of a signature algorithm and public key infrastructure to ensure that each bid is strongly tied to its respective bidder. The practical difference between these two approaches manifests itself in the feasibility of a forgery attack or tampering with data. VERIFIABLEAUCTION ensures that a successful forgery would be infeasible under our assumption 1. This also applies to tampering. This comes at a privacy cost: every member of an Estate can view the bid amounts of every other member of the Estate for every item. VEX avoids this problem by trusting the auctioneer to obfuscate identities and to not tamper with or omit data unless absolutely necessary, but a corrupt auctioneer can convey great advantages to certain bidders.

There are a few other differences between the protocols. Because hash chains are used to represent the monetary bid amounts in VEX, scaling for increased precision (such as allowing the user to specify cents or fractions of cents) increases the work performed in the hash chain exponentially. For example, a $100 bid commitment in an auction that only supports dollar values and nothing less would be represented as $H^{100}(s)$ where s is some random seed. If this auction supported bids of dollars and cents, the same $100.00 bid commitment would be represented as $H^{10000}(s)$. The commitment scheme used by VERIFIABLEAUCTION scales linearly with the input length since the bid commitment is represented as a MAC of a string which, in turn, represents the bid amount. More concretely, the difference between a $100 bid when cent values are unsupported and when they are is $\text{MAC}_k(\text{“100”})$ versus $\text{MAC}_k(\text{“10000”})$.

6 Usage Notes

Every member of an Estate participates in every auction. If a member does not wish to bid on an item, the same bidding protocol is to be carried out, the only difference being that $b = 0$. A member not responding is indistinguishable from the Server ignoring the member. That is why the protocol requires that every member submit some bid.

The onus of reporting any error whatsoever is completely on each and every member of an Estate. If any error is encountered, the member should report the error to every other member of the Estate through out-of-band communication.

One essential part of every auction process is the waiting period between Phase #1 and Phase #2 whereby every member is given time to copy every other member's bid commitment into the client software. If this

waiting period is skipped, members cannot download the commitment data and thereby lose the ability to verify other members' bids in Phase #2. If this situation occurs, it is required that the affected member(s) make this known to the rest of the Estate.

7 Conclusion

Under the assumptions outlined above, VerifiableAuction ensures that bids are secret until all bidders have sent in their bids, that the bids asserted are equal to the secret bids, and that each bidder's bids are identified with that bidder. No member need trust any other member, the Estate Admin, or the Server.

7.1 Future Work

It is possible that this system can be orchestrated through a public means whereby identity information and communication are all provided by some trusted source. We highlight that trust should not by any means be given lightly. One practicable candidate for this method of auction conduction is Twitter. It may be feasible for users to simple trade Twitter handles at the creation of an Estate, thereupon conducting all in-band communication through Twitter messages (or several messages using message splitting).

7.2 Acknowledgments

We would like to thank Michael Walfish and Sebastian Angel for their comments and suggestions.

References

[1] S. Angel and M. Walfish. *Verifiable Auctions for Online Ad Exchanges*. https://dl.acm.org/citation.cfm?id=2486038

[2] "What are the pros/cons of using symmetric crypto vs. hash in a commitment scheme?" *Stack Exchange*. http://crypto.stackexchange.com/questions/12247/what-are-the-pros-cons-of-using-symmetric-crypto-vs-hash-in-a-commitment-scheme.