The AuthAuction Secure Auction System

Michael Rosenberg

#  Abstract

In this paper we present a cryptosystem that will allow for fair sealed ﬁrst-price auctions to be con- ducted over the internet without the need for a trusted third party. The cryptosystem consists of a set of protocols deﬁned between clients and a server that utilize cryptographic primitives to ensure the infeasibility of cheating (as deﬁned within) without detection. The server software is simple, as it is required to perform no cryptographic computations. The suite is compared with and contrasted against the  protocol for auctioning in online ad exchanges.

#  Introduction

It is fairly common for particularly wealthy families to host estate auctions following the passing of a family member. This process would normally entail a very steeply-priced lawyer acting as an arbitrator over the entire process to ensure that no party is unfairly treated. The current state of aﬀairs could be greatly improved using an easily deployable bundle of software. AA provides just this. It is a simple, if not rudimentary, and secure suite with which such families may conduct sealed ﬁrst-price auctions without human supervision. Wealthy families are only one use case. Any situation in which the items being auctioned and their prices are to be kept secret from outsiders would also find be a target of this system.

#  Terminology

We will deﬁne several terms so that they may be used in this paper without further ambiguity. The Server is the central location and computer where information about Estates are stored and auctions are conducted. We deﬁne an Estate as a series of Auctions on individual items held among some group of people (called members). The Phases of bidding are the steps each client must take in the bidding process (our system requires two Phases). Corresponding to each Server (the system supports many [explain how]) there is a Global Administrator. To each Estate there is an Estate Administrator as well as the members of the Estate.

#  Speciﬁcation

Our cryptosystem relies upon the following primitives:

|  |  |
| --- | --- |
| Message Authentication Code Function | *k*(msg) |
| Hash Function |  |
| Public-Private Key Digital Signature Algorithm | *k*(msg), *k*(msg*,* sig) |
| Cryptographically Secure Pseudorandom Number Generator |  |

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



The basic strategy in AA is to externalize all cryptographic operations to the users themselves as opposed to the server. The security of the system relies on having every user use client-side software. The advantage is that the clients don’t need to trust the server, but the disadvantage is that the clients have to work harder than they would when using say Ebay.

## . Setup

The Global Administrator (the owner of the auction server) creates a new Estate and a new Estate Administrator. The Estate admin then proceeds to add users to the estate by username. Every user is expected to generate a new keypair corresponding to the particular Estate and disseminate his/her respective public keys to all other members of the Estate, preferably in a secure, out-of- band, manner, e.g. by reading over the phone. The Server is not expected to know any information about the users other than their usernames. One might propose that the Server have a list of the members’ public keys for the sake of ease-of-access but this practice is likely to breed trust in the Server which can be severely detrimental to the system.

## . Auction Process

An auction may be added or removed by the Estate admin only. Each auction’s  must be unique within the Estate. The client software will determine whether this is the case upon entry of a new auction. [How?] If it is not the case, then the user who receives this error is to report it immediately to the Estate Administrator as well as all other members of the Estate.

Bidding in an auction is broken up into two phases. Phase #1 is for the placing of bids in an irrevocable manner. The data submitted here are called bid commitments. Phase #2 is for the submission of the data both revealing the bid and authenticating the commitment.

During Phase #1 , each member of the Estate must ﬁrst generate a new random string of bytes

*p* heretofore known as the Prekey. This key is appended to the auction’s  and hashed to produce the key to be used with the  function *k* = (id*∥p*). The user then computes and submits

*s* = *i*(*k*(*b*)) where *i* is the user’s private key and *b* is the user’s bid.

The Server institutes a waiting period following the end of Phase #1 that allows every mem- ber of the Estate to download the bid commitment information of every other member. Each member inputs this block of information into his/her accompanying AA client-side soft- ware 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 user 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

*z* = (id*∥p*)(*b*)

so as to assert that *u*(*z, s*) succeeds, where *u* is the public key of the user whose bid is being veriﬁed.

If at any point in the protocol one user fails to submit valid data or submit data at all, every other member of the Estate must wait until the user complies. Furthermore, if at any point an-



nounces that something is not correct, the auction is to be restarted. This is due to the fact that it is impossible to determine the perpetrator of the incident and it may sometimes be impossi- ble to even determine if the whistle-blower is telling the truth. Thus the only way to ensure an untampered auction is to complete one without any user complaints. Fortunately, each user has an incentive both to behave honestly and to check on others.

#  Analysis

## . Goals

AA was created with a list of priorities in mind relating to the use and robustness of the system. They are formalized here:

() To allow for a straightforward setup process without requiring a great knowledge of cryptography () To prevent users from retrieving bid amounts prematurely, i.e., before the beginning of

Phase #2

() To prevent users from tampering with bid commitments after the end of Phase #1 () To prevent the Server from ignoring bids from any subset of Estate members

() To prevent users from denying their bid commitments after submission () To prevent users from forging bids or bid commitment data

## . Reasoning

The purpose 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 such an advantage are detectable but not necessarily traceable to the perpetrator. The following is the reasoning that has gone into the design of the cryptosystem.

We begin with the use of the  function. The key used for the message authentication code function is a mix of random data and the auction’s . The reason for this is that if only a random key were used, the same bidding information can be replayed for diﬀerent auctions. This would allow an attacker to copy and paste previous bidding information into a new auction without the knowledge or consent of the actual bidder. Using the auction’s  in the generation of the  key ensures that the key can be used only for that one auction. Thus it is important that the auction

 never repeats on the same server. [Hmm. If an auction fails for some reason, then we should require that it be replaced by a new auction id. But then couldn’t there be an attack based on repeating an id? Maybe the id should include a datetime in greenwichmeantime down to the second as well as an estateid] This is further explained in §§. and ..

We choose to use a  as opposed to a hash because the signature function is not intended to obfuscate the underlying data. Knowing this, an attacker could theoretically extract the hash from the signature *h* = (*b*) (again, where *b* is the user’s bid). Now deriving *b* given *h* is simple. Simply iterate *d* over all possible values of *b* until *h* = (*d*) is satisﬁed. Since there are not many possible values for *b* (in terms of money, around ), this attack is quite feasible. On the other hand, using a  function with a random key of a suitable length, the adversary would not only have to guess the correct value for *b* but the correct value for *p* (the Prekey) which is infeasible.



The choice between using a  vs. a symmetric cipher for committing to a datum is, to an extent, arbitrary. However, MAC has a few advantages. 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 , oﬀers more choices in block size than  (the former coming in ﬂavors of , , , and  bit block sizes, while the latter comes with the choice of a  or  bit block size).

The use of signatures in the system is much more obvious. Without digital signatures, it would be trivial for an attacker to masquerade as another member of the Estate.

The two-Phase system is intentionally very similar to a blinding protocol, the only diﬀer- ences being that it is authenticated, it uses a  instead of an , and the actual plaintext data is submitted in the second step. The logic is that all members of an Estate will have access to each other’s signatures [signed bids?] after Phase #1 . At this point, everyone’s bids are immutable, because if they are mutated later in Phase #2 , they will not match up with the signature [signed bids] given previously. In this way, every user may now submit their bid without the possibility that another user will see it and intentionally out-bid him.

The reason all users must wait for a non-responsive user to comply is that it is inherently impossible to determine the cause of the error (whether it be a lack of response or submission of invalid data). From the perspective of any other user, it may well be the Server itself that is ignoring or modifying data.

## . Correctness

Here we show that goals -. Are satisﬁed as stated above.

*To prevent users from retrieving bid amounts prematurely, 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 (id*∥p*)(*b*). Retrieving *b* from this is equivalent, if not more diﬃcult, than retrieving a known input to a  with an unknown key. Based on the assumption that a secure  function should be resistant to this very attack, we assert that the cryptosystem is secure against this threat.

*To prevent users from tampering with committed bidding information after the end of Phase #1*

Because of the Server-instituted waiting period between phases, an attempt to tamper with bid- ding information on the Server at any point after the clients had already downloaded the informa- tion would be completely ineﬀectual. Tampering with bidding information on the Server before all members have downloaded [somehow I think of this as uploading] the bidding information (recall that Phase #2 does not com- mence until all members have acknowledged that they have downloaded the bidding information) is possible but could not result in any member’s beneﬁt. If a member were to tamper with his own

<http://crypto.stackexchange.com/questions/12247/what-are-the-pros-cons-of->

using-symmetric-crypto-vs-hash-in-a-commitment-scheme



commitment data on the Server (implying that the member has access to the private key used to sign the commitment), the member receives no unfair advantage, as this (albeit unscrupulous) ac- tion is equivalent to the member making his initial bid commitment. If, on the other hand, one were to tamper with another member’s commitment data before any other member has the abil- ity to download it from the Server, it would show after Phase #2 that that member’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 unfair advantage. 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 user, this is in violation of the assumption that keys are private . Furthermore, if the attacker were to strike after a subset of 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. Because the system is assuming human bidders, this makes sense.

It is also a possibility 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.

*To prevent the Server from ignoring bids from any subset of Estate members*

When commitment data is imported into the client software, a cursory check is performed to en- sure that every member’s commitment data has been submitted. If this is not the case, the member is presented with a visible warning of just that. It is then, again, the responsibility of the member to inform the other members as well as the Server that there is missing data.

*To prevent users from denying their bid commitments after submission*

Every bid commitment is signed with the corresponding member’s private key. This commit- ment 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* = *i*(*z*) and a public key *u*, one can generate a new public key *u* such that *u* (*z, s*) and *u* (*z, s*) both succeed. Given a suﬃciently secure set of asymmetrical cryptography primitives, this is computationally infeasible.

*To prevent users from forging bids or bid commitment data*

[For brevity, we deﬁne (id*, k, b*) = (id*∥k*)(*b*).] There are multiple ways a user could go about forging another user’s data. Suppose, ﬁrstly, that Mallory (the would-be forger) has man- aged to circumvent the login system, allowing him to login as another member, Alice. If the cur- rent auction is in Phase #1 , Mallory may submit his own commitment data, *c* = *m*((id*, k**, b*)), 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. By Phase #2 , Mallory will be forced to enter the bid data *k* and *b* (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, k**, k**, b**, b* such that *c* is a valid signature of (id*, k**, b*) under Alice’s key *a*. This should not be feasible under the assumption of secure  and  func- tions. Furthermore it would be even more diﬃcult for Mallory if he were to ﬁx a particular *b* as Alice’s public perceived bid amount.

It may also be possible that Mallory only submits his own data in Phase #2 . In this case, Mal- lory would be forced to solve for *b* and *k* such that Alice’s commitment, *A* = *a*((id*, k**, b*)), is a valid signature of (id*, k**, b*). This should also be infeasible by our assumption that every private-public key pair is private

Mallory also has the ability of copying Alice’s bidding information from previous auctions into a new auction. That is to say that the *k* and *b* are known to Mallory before the attack even begins. This is known as a replay attack and it is prevented with the use of unique auc- tion s for every auction. Mallory would copy one of Alice’s previous bid commitments, *A* =

*a*((id*∥k*)(*b*)), in Phase #1 . This means Mallory would have to solve for a *k**, b* and id such that *A* is a valid signature of (id*, k**, b*). Again, this is not feasible. Note that when id = id the solution is trivial: *k* = *k* and *b* = *b*. [another reason to use datetime]

#  Comparison

The two-part bidding process used here is very similar to that of  insofar as they both feature a commitment stage and a revealing stage. The essential diﬀerence is how the bids are revealed and to whom.  states that every user should share his bid amount with the Server and that the Server should keep the bid amount secret from the rest of the bidders. AA states that the Server is to collect all of the bid amounts and display them for every user to see. 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 their bid was less than the winning bid. This level of trust did not appear to the creators to be worth the risk and so was intentionally left out. The alternative for private integer comparison is through an interactive protocol. This, too, was not included in the speciﬁcation due to the inherent complexity of performing a  interactive protocol over the internet with no additional provided infrastructure.

 happens to take a quite diﬀerent approach to bidder deniability. It is explicitly stated that there is nothing linking a bid with the bidder. AA employs the use of a signature algorithm 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. AA asserts that a successful forgery would be infeasible given a suﬃciently strong asymmetric cryptosystem. This also applies to tampering.  makes no such guarantee and instead proposes that a user who has experienced such cheating on the Server not participate further. The beneﬁt that plausible bid deniability aﬀords the users of the  cryptosystem is

SOURCE THIS

privacy of bid amount. This is to say that even if a user obtained all bid amounts, there would be nothing linking them to the bidders themselves. This particular facet of privacy is one that took precedence in the creation of  and is not supported by AA.

Because hash chains are used to represent the monetary bid amounts in , scaling for in- creased 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 $ bid commitment in an auction that only supports dollar values and nothing less would be represented as *H*(*s*) where *s* is some random seed. If this auction supported bids of dollars and cents, the same $. bid commitment would be represented as *H*(*s*). The commitment scheme used by AA scales linearly with the input length since the bid commitment is represented as a  of a string which, in turn, represents the bid amount. More concretely, the diﬀerence between a $ bid when cent values are unsupported and when they are is *k*(“”) v.s *k*(“”).

’s use of an interactive auditing protocol is necessary to maintain bidder privacy. The pro-

tocol is very beneﬁcial to the system, though it certainly adds a non-negligible amount of com- plexity and overhead. In comparison, AA sacriﬁces bidder anonymity to allow each and every member to independently audit the entire auction procedure. Again, this stresses the importance of deniability versus decentralization in the design of both systems. [Please rephrase that last sentence as a comparison of the systems, e.g. system A values X whereas B values Y]

One less rigorous feature of AA is that it frustrates attacks whereby the Server ig- nores or drops a member’s bid. AA requires that every user know of every other user. This makes it simple for any member to determine whose bid is missing when such a situation arises. The same cannot be said for  whereby not only can no member determine whose bid is missing, but it is possible that no member would realize that a bid was missing in the ﬁrst place.

#  Criticalities

## . Subtleties

A few essential subtleties need be pointed out to ensure that simple but catastrophic errors are avoided by anyone implementing this cryptosystem.

The ﬁrst subtlety is the necessity that 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* = . A user not responding is indistinguishable from the Server ignoring the user, thus we necessitate that every member submit a datum.

The onus of reporting any error whatsoever is completely on each and every member of an Estate. If any error is encountered, the user is to report the error to every other member of the Estate through out-of-band communication. This is the only way that every member can be sure that an auction completed without errors.

To keep track of how much a user has bid, an implementation may opt to give each user a ﬁnite number of coins and have only the winner of an auction have the appropriate number of coins withdrawn from their sum. To avoid trusting the Server, all coin-amount calculation ought to be done by each client individually. The number of coins should only be decremented if the user can verify for himself that the winner of the auction truly was the winner. Thus, the number

of coins each user has should not be stored on the Server to prevent the users from having to trust (or naively trusting) the Server’s calculations.

## . Vulnerabilities

It is just as important to indicate what guarantees a cryptosystem does not make as it is to in- dicate those that it does make. Certain design choices have lead AA to be unable to make certain guarantees to the user. These missing guarantees that are security related are listed as vulnerabilities.

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

The process of cheating mitigation contains a vulnerability within itself. Any user has the ability to claim that they have been misinformed by the Server, a veriﬁcation procedure failed, etc. No other user has any ability to prove whether the whistleblower is telling the truth or not. Thus, the only procedure regardless of the true intentions of the whistleblower is to repeat the process until it completes with no issues reported by any user. Thus any single user can indeﬁnitely stymie the procession of their Estate.

#  Future Work

It is possible that this system can be orchestrated through a public means whereby identity in- formation and communication are all provided by some trusted source. We highlight that trust should not by any means be taken 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. One notable obstacle that stands is that *tweets* (Twitter messages) are limited to  Unicode characters. A potential solution would employ the use of message splitting in an unambiguous manner.