Masters Dissertation

Verifiable e-auction over a Block-Chain

Sayantan Chakraborty

MTech Cryptology and Security
Indian Statistical Institute, Kolkata

Verifiable e-auction over a Block-Chain

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Cryptology and Security

Submitted by

Sayantan Chakraborty

Roll No CRS1907
MTech Cryptology and Security
Indian Statistical Institute Kolkata

Under the guidance of
Prof. Feng Hao
Professor of Security Engineering
University of Warwick, Warwick, UK

Prof. Bimal Kumar Roy
Applied Statistical Unit
Indian Statistical Institute, Kolkata, India

July 9, 2021

Dedicated to the asteroid "33179 Arsènewenger"

CERTIFICATE

This is to certify that the dissertation entitled **"Verifiable e-auction over a Block-Chain"** submitted by **Sayantan Chakraborty** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology** in **Cryptology and Security** is a bonafide record of work carried out by him under our supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in our opinion, has reached the standard needed for submission.

Feng Hao
Professor of Security Engineering,
Department of Computer Science,
University of Warwick, Warwick, UK.

Bimal Kumar Roy
Professor,
Applied Statistics Unit,
Indian Statistical Institute, Kolkata, India.

Acknowledgments

I would like to show my highest gratitude to my advisors, *Prof. Feng Hao*, Department of Computer Science, University of Warwick, Coventry, UK and *Prof. Bimal Kumar Roy*, Applied Statistics Unit, Indian Statistical Institute, Kolkata, India for their guidance and continuous support and encouragement. Prof. Hao has taught me how to do good research, and motivated me with great insights and innovative ideas. Prof. Roy has made it possible for me to take part in such a good research work in collaboration with such prestigious institute and has helped me in every manner starting from the beginning of my journey at Indian Statistical Institute till date.

I would specially thank *Subhra Majumder* for her guidance to make me understand the implementation details over Test Ethereum Network from scratch. She did help me a lot to digest the earlier developed models, on basis of which I have designed the protocols in this period. I would also thank *Anisha Dutta* for helping me throughout while implementing the protocols and writing the thesis report in a suitable manner.

My deepest thanks to *Somnath Panja* for his suggestions and discussions. I shall be thankful to *Prabal Banerjee*, and *Bibhas Chandra Das* for their initial help to make me understand the subject and the SEAL protocol properly. They have made my work a lot easier to me with their important suggestions. Last but not the least, I shall be grateful to my parents to keep me motivated in such an unfortunate pandemic times, which let me left with no option but to have the internship virtually, staying in the house for the maximum period of this six-month duration of the internship.

Sayantan Chakraborty
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

Auction has been an integral part of trading throughout ages. Sealed-bid e-auction (both first-price and second-price) is considered to be a classic example of Secure Multi-Party Computation problem. In this project, without assuming any role of auctioneer, we try to formulate a protocol in which the bidders jointly compute the highest bid, keeping all the losing bids secret to the bidders. The focus of the work is to study and implement a verifiable sealed-bid e-auction scheme in decentralized settings. We have studies some possible choices of schemes and decides to further work on the SEAL protocol, which eventually is the first decentralized sealed-bid auction protocol to achieve linear computation and communication complexity. The project aims to build a platform which shall take bit-strings as bid inputs from the bidders and output the final result which is maximum among the bids. We have implemented the segments of the protocol using Solidity Language over a Test Ethereum Network and the front end is to be done using HTML Languages. This implementation targets to achieve publicly verifiability without leaking information about the bids form the bidders.

Keywords : SEAL, e-auction, Verifiability, Sealed-Bid Auction, Implementation, Block-Chain, Solidity Language, Test Ethereum Network, secure Multi-Party Computation.

Contents

List of Figures

1 Introduction

1.1 Basics of Auction

Auction is a well-known method of buying and selling goods or services by offering them to a set of bidding-allowing people, said as bidders to compete against each other and bid. The winner gets the good or service by offering the highest bid. The word *auction* is derived from the Latin *auctum*, which means *I increase*, which is self-explaining the name of the process. The origin of auctions can be traced back to approximately 500 B.C., in some places of ancient Greece. Auction is now-a-days a common practice in our society from the usage of allocation of bandwidth spectrum to the sales of antiques, painting, expensive wines, commodities, livestock, radio spectrum, used cars and rare collectibles. It is also vastly used by governments as the long-term securities are sold in weekly or monthly auctions conducted by many government bodies. Investment bankers also use auctions to attract the highest possible price when selling a company.

Auctions are mainly of two types - open cry auction and sealed bid auction. These two types are further classified into different sub-types as well. In an open cry auction, either the bid starts from a reserve price until there is only one bidder left (commonly known as English Auction) or the bid goes in descending way from a high price until the first bidder agrees to pay (known as Dutch auction). The open ascending price English auction is arguably the most common form of auction in use throughout history. Participants bid openly against one another, with each subsequent bid required to be higher than the previous bid. An auctioneer may announce prices, bidders may call out their bids themselves or have a proxy call out a bid on their behalf, or bids may be submitted electronically with the highest current bid publicly displayed. Interestingly, in the Ancient Greece, women were auctioned off for marriage and that auction followed a descending pricing Dutch method [12], beginning with the highest price and going lower until the lowest bid was found, as long the bid price was more than, or equal to, the reserve price set by the seller. In fact, the bidders were allowed to recover their money as well, if they could not provide successful bids.

In a sealed-bid auction, each bidder hands over a sealed envelope containing their bid to an auctioneer. The bid is supposed to be a secret to all other participants. After receiving the secret bids from all the bidders, the auctioneer opens all envelopes in private and declares the highest bidder as the winner, while keeping losing bids suppressed. Hence, if some bidder do not want to reveal his/her bid to other bidders, the bid will remain secret. Sealed-bid auctions mainly are of two varieties - first-price sealed-bid auction, commonly known as Blind auction and second-price sealed-bid auction, also known as Vickrey Auction, as a tribute to William Vickrey, who academically described the procedure. In the first-price sealed-bid auction, the winner pays for the amount he/she has bidden *i.e.* the highest bid. While in the second-price sealed-bid auction, the winner only needs to pay the second highest bid.

1.2 Need for Auctioneer-free e-auction

The advantage behind the sealed-bid auctions lies in the fact that no bidder learns any information about the other bids. Hence, the bidders are encouraged to bid according to their monetary valuation of the asset. The traditional sealed-bid auction schemes generally assume the role of an honest auctioneer to conduct the whole process and evaluate the result. But, in practise, we can have several drawbacks because of this assumption. A dishonest auctioneer

primarily may not follow the actual procedure of the auction and instead of declaring the correct winner, it may disclose some other bid/bidder as winner. Since other bids are kept as commercial secret, no one can directly identify the flaw. Also, a dishonest auctioneer might disclose the losing bids to other parties, which should be kept as a secret. Moreover, in Vickrey auction scheme, auctioneer may surreptitiously substitute the second highest price to one slightly below the top price to increase the revenue. Since the winner also do not know the exact figure of the second highest bid, he/she. once knowing that amount is less than what he/she has bidden, would pay the cost. This affects the integrity and reliability of the whole auction scheme.

Since the main drawbacks of the schemes are related to the distrust on the auctioneer, the obvious target over years was to nullify the trust problem about the auctioneer. Some schemes were proposed to apply threshold cryptography or multiparty computation techniques to distribute the trust from a single auctioneer to two or several. However, one can never rule out the possibility that the auctioneers may collude all together to compromise the privacy of the bids. Therefore, the introduction of auctioneer-free e-auction schemes were welcomed.

Web-based online commercial activity for e-auctions were introduced back in 1995, when two auction sites were founded independently with alternative business models. Though primarily e-auction protocols assumed a role of an unknown auctioneer, but later on varieties of sealed-bid e-auction schemes have been proposed over years, which were the auctioneer-free. These auctions are completely run by the bidders themselves without involving any auctioneer, and all operations are publicly verifiable.

1.3 Objective of the Project

Sealed-bid e-auction is considered to be an instance of a secure multiparty computation (MPC) problem, in which the bidders jointly compute the highest bid, keeping all the losing bids secret. In this project, we wish to **study and implement a verifiable sealed-bid e-auction scheme in decentralized settings**. In a decentralized setting without any auctioneer, it is desirable not to use any secret channels between bidders, so all operations are publicly verifiable. Hence, third-party observer like the seller or any representative from any of the bidders, who is not directly involved in the bidding part, can also verify the integrity of the auction process. We hereby consider the SEAL protocol [13], which assumes availability of only an authenticated public channel to all the participants. SEAL protocol is also the first decentralized sealed-bid auction protocol, which achieves a linear computation and communication complexity. No sealed-bid auction scheme prior to this has achieved the linear system complexity in a decentralized setting.

Ethereum is an open-source, decentralized software platform, based on Block-Chain technology. It enables Smart Contracts and Distributed Applications (DApps) to be built and run without any downtime, fraud, control or interference from a third party. Once DApps are deployed on the Ethereum network, it is unchangeable. DApps can be decentralized because they are controlled by the logic written into the contract, not an individual or a company. To achieve the decentralized setting we need for the protocol, we shall use this platform for implementation. Therefore, the primary objective of this project is to **implement the Self-Enforcing Auction Lot (SEAL) protocol over a Test Ethereum Network (testnet)**. For a practical purpose usage, we initially intend to execute the protocol over a Test Ethereum Network and further would try to have a real world demo, if possible.

1.4 Our Contribution

With the bigger picture of studying and designing the whole Self-Enforcing Auction Lot protocol in mind, we can fairly claim to have these following works as our contributions towards the mentioned SEAL protocol.

1. We thoroughly discussed the SEAL protocol and the journey towards the protocol over years, starting from Dinning Cryptographer's problem, DC Net, AV Net, Modified AV Net and how that shaped the idea of SEAL protocol. We have analyzed earlier schemes who tried to solve the specific problem of trust-issue free auction schemes and how SEAL achieve that with linear computational and communication complexity.

2. We implemented the full tally part of the SEAL protocol. We have presented both the cases with single bit inputs and how that can be extended into multiple bit inputs.

3. We have adapted the design rationale of the implementation of the SEAL protocol from the earlier developed implementation of Open Vote Network by Patrick McCorry. We have presented the idea in section 6.1. in details.

4. We have also implemented a total of four different NIZKPs, namely (Schnorr's) ZKP for well-formedness of Public Keys, one-out-of-two ZKP for well-formedness of commitments, one-out-of-two ZKP for well-formedness of cryptograms in Stage I calculations and one-out-of-three ZKP for well-formedness of cryptograms in Stage II tally. The ZKPs are implemented in Solidity language over Ethereum Network. For generation of the text file required for the registration process, which contains all random nonces, public and private keys, we have implemented a JAVA code, which is also provided.

2 Background

2.1 Number Theoritic Primitives

Number theory is a source of several computational problems that serve as primitives in the design of cryptographic schemes. Asymmetric cryptography in particular relies on these primitives. As with other beasts that we have been calling primitives [7], these computational problems exhibit some intractability features, but by themselves do not solve any cryptographic problem directly relevant to a user security goal. But appropriately applied, they become useful to this end. In order to later effectively exploit them it is useful to first spend some time understanding them.

Let G be a cyclic group and let g be a generator of G *i.e.* $G = \{g^0, g^1, \ldots, g^{m1}\}$, where $m = |G|$ is the order of G. The discrete logarithm function $DLog_{G,g} : G \to \mathbb{Z}_m$ takes input a group element a and returns the unique $i \in \mathbb{Z}_m$ such that $a = g^i$. There are several computational problems related to this function that are used as primitives.

2.1.1 Discrete Logarithm Problem

Discrete Exponentiation function takes input $i \in \mathbb{Z}_m$ and returns the group element g^i. Discrete Logarithm function is the inverse of the Discrete Exponentiation function. The definition measures the one-wayness of the discrete exponentiation function according to the standard definition of one-way function.

Let G be a cyclic group of order m, let g be a generator of G, and let $\mathcal{A}$ be an algorithm that returns an integer in $\mathbb{Z}_m$. We consider the following experiment.

Experiment $\mathbf{Exp}^{DL}_{G,g}(\mathcal{A})$

$x \leftarrow_R \mathbb{Z}_m$
$\bar{x} \leftarrow \mathcal{A}(X)$
If $g^{\bar{x}} = X$, then **return** 1
else return 0

Then the DL-advantage of $\mathcal{A}$ is $\mathbf{Adv}^{DL}_{G,g}(\mathcal{A}) = Pr[\mathbf{Exp}^{DL}_{G,g}(\mathcal{A}) = 1]$

The discrete logarithm problem is said to '**hard**' in G if the DL-advantage of any adversary of reasonable resources is small. Resources here means the time-complexity of the adversary, which includes its code size as usual.

2.1.2 Decisional Diffie-Hellman Problem

The Decisional Diffie–Hellman (DDH) assumption is a computational hardness assumption about a certain problem involving discrete logarithms in cyclic groups. It is used as the basis to prove the security of many cryptographic protocols, most notably the ElGamal and Cramer–Shoup cryptosystems. This is named after Whitfield Diffie and Martin Hellman [8].

The formalization of DDH Problem considers two-worlds setting. The adversary gets input X, Y, Z. In either world, X, Y are random group elements, but the manner in which Z is chosen depends on the respective world. In World-1, $Z = g^{xy}$ where $x = DLog_{G,g}(X)$ and $y = DLog_{G,g}(Y)$. In World-0, Z is chosen at random from the group, independently of X, Y. The adversary must decide in which world it is.

Experiment $\mathbf{Exp}_{G,g}^{DDH-1}(\mathcal{A})$	Experiment $\mathbf{Exp}_{G,g}^{DDH-0}(\mathcal{A})$
$x \leftarrow_R \mathbb{Z}_m$	$x \leftarrow_R \mathbb{Z}_m$
$y \leftarrow_R \mathbb{Z}_m$	$y \leftarrow_R \mathbb{Z}_m$
$z \leftarrow xy \mod m$	$z \leftarrow_R \mathbb{Z}_m$
$X \leftarrow g^x$	$X \leftarrow g^x$
$Y \leftarrow g^y$	$Y \leftarrow g^y$
$Z \leftarrow g^z$	$Z \leftarrow g^z$
$d \leftarrow \mathcal{A}(X, Y, Z)$	$d \leftarrow \mathcal{A}(X, Y, Z)$
return d	**return** d

Then the DDH-advantage of $\mathcal{A}$ is

$$\mathbf{Adv}_{G,g}^{DDH}(\mathcal{A}) = Pr[\mathbf{Exp}_{G,g}^{DDH-1}(\mathcal{A}) = 1] - Pr[\mathbf{Exp}_{G,g}^{DDH-0}(\mathcal{A}) = 1]$$

The Decisional Diffie-Hellman (DDH) Problem is said to '**hard**' in G if the DDH-advantage of any adversary of reasonable resources is small, where the resources is the time-complexity of the adversary, as stated before.

When using a cryptographic protocol whose security depends on the DDH assumption, it is important that the protocol is implemented using groups where DDH is believed to hold. Here is some standard example of groups for which DDH is assumed to hold.

- A prime-order elliptic curve E over the field $GF(p)$, where p is prime, provided E has large embedding degree.
- A Jacobian of a hyper-elliptic curve over the field $GF(p)$ with a prime number of reduced divisors, where p is prime, provided the Jacobian has large embedding degree.
- The subgroup of kth residues modulo a prime p, where $(p-1)/k$ is also a large prime (also called a Schnorr group). For the case of $k = 2$, this corresponds to the group of quadratic residues modulo a safe prime.

Importantly, the DDH assumption does not hold in the multiplicative group $\mathbb{Z}_p^*$, where p is prime. This is because if g is a generator of $\mathbb{Z}_p^*$, then the Legendre symbol of g^a reveals if a is even or odd. Given g^a, g^b and g^{ab}, one can thus efficiently compute and compare the least significant bit of a, b and ab, respectively, which provides a probabilistic method to distinguish g^{ab} from a random group element. Also, the DDH assumption does not hold on elliptic curves over $GF(p)$ with small embedding degree (say, less than $\log^2(p)$).

2.2 Zero-Knowledge Proofs

A zero-knowledge proof is one of the most abstract and fascinating concepts in applied cryptography today. From potentially being used in nuclear disarmament to providing anonymous and secure transactions for public blockchain networks, a zero-knowledge proof is a profound example of cryptographic innovation. In cryptography, Zero Knowledge Proof is a method by which one party (the prover) can prove to another party (the verifier) that she has the knowledge of a value x, without conveying any information apart from the fact that she knows the value x. The essence of a zero-knowledge proof is that it is trivial to prove that someone possesses knowledge of certain information by simply revealing it. The challenge is to justify such possession without revealing the information itself or any additional information.A zero-knowledge proof must satisfy the following three parameters.

1. **Completeness.** If the statement is true, the honest verifier, the one that is following the protocol properly will be convinced of this fact by an honest prover.
2. **Soundness.** If the statement is false, no cheating prover can convince the honest verifier that it is true, except for some small probability, and
3. **Zero Knowledgeness**. If the statement is true, no verifier learns anything, except the fact that the statement is true. Completeness and soundness are properties of more general interactive proof systems. The addition of zero knowledge is what turns the verification process into a zero-knowledge proof.

Zero-knowledge proofs were first conceived in 1985 by Shafi Goldwasser, Silvio Micali, and Charles Rackoff [19]. This paper introduced the IP hierarchy of interactive proof systems and conceived the concept of knowledge complexity, a measurement of the amount of knowledge about the proof transferred from the prover to the verifier. They also gave the first zero-knowledge proof for a concrete problem, that of deciding quadratic non-residues $\mod m$. Non-Interactive Zero Knowledge Proofs (NIZKP) are special type of Zero Knowledge Proofs that require no interaction between the prover and the verifier. Formally a non-interactive zero-knowledge proof system for a relation R is the triplet $\Gamma = (K, P, V)$ where K, P and V are three PPT algorithms defined as following.

- K is the *set-up algorithm*. It generates a common reference string(σ). It takes a λ as input and returns the common reference string. Mathematically we can say: $\sigma \leftarrow K(1^{\lambda})$.
- P is the *prover algorithm*. It takes as input a statement x and a corresponding witness w such that $R(x, w) = True$. With x, w and also common input σ, P outputs the proof π. Mathematically we can write $\pi \leftarrow P(\sigma, x, w)$.
- V is the *verifier algorithm*. It takes as input a statement x and the proof π. With x, π and also common input σ, V returns $v \in \{0, 1\}$. Mathematically we can write $v \leftarrow V(\sigma, x, \pi)$.

By definition, an efficient NIZKP Γ must satisfy the following three properties.

- **Completeness.** Γ is complete if verification succeeds for every $\sigma \leftarrow K(1^\lambda)$ and for every valid pair $(x, w) \in R_\sigma$. Mathematically we can say for completeness.

$$Pr\left[\sigma \leftarrow K(1^\lambda); \pi \leftarrow P(\sigma, x, w), 1 \leftarrow V(\sigma, x, \pi) \wedge R(x, w) = True\right] = 1$$

- **Soundness.** To achieve soundness, Γ has to guarantee that no prover can make the verifier accept a wrong statement $x \notin L$, except with some small probability. Mathematically we can say for soundness.

$$Pr\left[\sigma \leftarrow K(1^\lambda); (x, \pi) \leftarrow \mathcal{D}(\sigma), 1 \leftarrow V(\sigma, x, \pi) \wedge x \notin L\right] < negl(\lambda)$$

  Here $\mathcal{D}$ is the algorithm that the cheating prover adapts to generate a statement $x \notin L$ and the proof π.

- **Zero Knowledgeness.** We say that Γ is zero-knowledge if there exist a simulator $Sim = (S_1, S_2)$ such that for any non-uniform PPT adversary $\mathcal{A}$ the following condition holds true.

$$Pr\left[\sigma \leftarrow K(1^\lambda) : \mathcal{A}^{P(\sigma,x,w)}(\sigma) = 1\right] \equiv Pr\left[(\sigma, \tau) \leftarrow S_1 : \mathcal{A}^{Sim(\sigma,\tau,x,w)}(\sigma) = 1\right]$$

  $Sim(\sigma, \tau, x, w)$ outputs S_2 for $(x, w) \in R_\sigma$.

2.2.1 Example of Interactive ZKP

Suppose a voter has casts his vote x in encrypted manner, where instead of revealing x, she publishes $X = g^x$ publicly, where g be a generator of group G, in which the Discrete Logarithm Problem is hard. Now she wants to prove the knowledge of $x = \log_g X$. Now to form a Zero-Knowledge Proof, we design the following game.

Prover		**Verifier**
Chooses $r \leftarrow_R \mathbb{Z}_p$ Computes $\varepsilon = g^r$		
	$\xrightarrow{\varepsilon}$	
		Chooses $ch \leftarrow_R \mathbb{Z}_p$
	$\xleftarrow{ch}$	
Computes $\rho = r - x \ . \ ch$		
	$\xrightarrow{\rho}$	
		Checks if $\varepsilon \stackrel{?}{=} g^\rho X^{ch}$

Correctness. The ZKP is accepted if the verification equation holds true *i.e.* $\varepsilon = g^\rho X^{ch}$. The correctness proof lies to the fact that $\varepsilon = g^r = g^{\rho + x.ch} = g^\rho (g^x)^{ch} = g^\rho X^{ch}$.

Soundness. The above calculation itself explains that any incorrect entry by the dishonest prover won't be able to satisfy the equation.

Zero Knowledgeness. Note that the verifier knows information about ε, ch, ρ and X. From X, it cannot extract x due to the hardness of Discrete Logarithm problem. Also, it cannot extract x through the equation $\rho = r - x \ . \ ch$ as she cannot extract r from the knowledge of $g^r = \varepsilon$. Therefore, no leakage is there.

2.2.2 Example of Non-Interactive ZKP (NIZKP)

We wish to evaluate the same problem we discussed in the above example for Interactive ZKP. As above, a voter publishes $X = g^x$ in some public bulletin and wants to prove the knowledge of $x = \log_g X$. The difference with the above proof is that here no interaction between the prover and the verifier is allowed. Hence, we need a random challenge of the obtained through feeding the commitment and all other available argument into a random oracle. We hereby introduce a secure Hash function $\mathcal{H}$, which outputs a random challenge z. Infact there is no need for any verifier here as the code itself checks if the verification equations hold or not.

`Prover`	`Verification Equation`
Chooses $r \leftarrow_R \mathbb{Z}_p$	
Computes $\varepsilon = g^r$	$\varepsilon \stackrel{?}{=} g^\rho X^{ch}$
Computes $ch = \mathcal{H}(g, g^r, g^x, i)$	
Computes $\rho = r - x \ . \ ch$	

In this NIZKP, we call ch as challenge, ε as commitment and ρ as response. The prover needs to do just one exponentiation for generating the proof that contains one challenge, one commitment and one response, 3 parameters in total. The verifier needs to do 2 exponentiations to verify the proof. The NIZKP is accepted if the verification equation holds true *i.e.* $\varepsilon = g^\rho X^{ch}$. The correctness and soundness proofs are exactly like the previous case. Also this is zero-knowledge proof since we cannot extract information about x.

2.3 Journey of Sealed-bid Auction

In the previous section, we discussed that Sealed-bid e-auction protocol has some major disadvantage related to the distrust on the auctioneer. As a result, the obvious target over years was to nullify the trust problem about the auctioneer. Since the beginning of the research work in this field, we have seen different evaluations and innovations in this topic. Generic MPC techniques were used to solve the problem but since these techniques require pairwise secret channels between probably each pair of the participants and an authenticated public channel for all, achieving these in practical scenario seems almost impossible. In addition, generic MPC techniques suffer from various efficiency issues. We shall discuss some well-studied e-auction schemes over ages to understand how this problem had a journey through various proposed schemes.

The initial research papers, beginning with Franklin-Reiter's work [3] in 1996, have all assumed the role of a honest auctioneer, which is not appreciated for stated reasons. Hence, in the next years, the mainstream research regarding this problem focuses on applying cryptography to distribute trust on the auctioneer. In general, there were two main approaches.

1. The first approach is to apply threshold cryptography, or MPC techniques to distribute the trust from a single auctioneer to several auctioneers. Franklin et al. [3] presented a second-price sealed-bid auction scheme. In this scheme, a number of servers play the role of the auctioneer, and they apply Shamir's secret sharing technique to split

each bid among themselves so no single server sees all bids. However, there still can be similar problem if a sufficient number of servers collude, and eventually the secrecy of all bids could be lost. A similar approach like the last one was made to apply *Threshold cryptography* by Sako [4] to let auctioneers jointly decrypt submitted bids. Kurosawa and Ogata had a *Bit-Slice approach* to compute the highest bid bitwise, assuming the majority of the auctioneers are honest. Their system involves m bidders and n auctioneers, while the auctioneers apply secure multiparty computation on a bit-slice circuit, and decrypt the result at each bit position using verifiable threshold decryption. The threshold is set such that compromising the decryption requires compromising at least the majority of the auctioneers. The number of rounds required for threshold decryption is $\mathcal{O}(nc)$, where c being the bit length of the bid. But the desired complete trust-free condition has still not been satisfied by these protocols.

2. The second approach to solve the trust issue was introduction of more trusted third parties in addition to auctioneers. Naor et al. presented a second-price sealed auction scheme [24]. This scheme uses two different auction servers who communicate using an oblivious transfer protocol. One server takes the role as an auctioneer and the other as an auction issuer. The two servers are assumed not to collude. However, the original Naor et al.'s scheme has a weakness in which one of the two servers can cheat to modify bids without detection. Later on similar approaches were made by Abe and Suzuki using homomorphic encryption [25], by Montengero et al. employing an auctioneer and a randomness server [26], by Lipmaa et al. involving a seller and an auction authority [27], and various other cryptographers.

Brandt was among the first to argue that neither of the above approaches is desirable due to the involvement of trusted auctioneers or third parties. Brandt proposed *Bidder Resolved Auction* and an auctioneer-free solution by applying secret sharing techniques. A major drawback of this is that seller is actively involved in the protocol and if seller colludes with bidders, the bidders can learn other bids. Inspired by Brandt's scheme [2], Wu et al. remove the seller and propose a decentralized sealed bid auction scheme based on a general *Socialist Millionaire Protocol*. Though the cost of the computational load and the bandwidth usage per bidder is $\mathcal{O}(2^c)$. SEAL [13] is an auctioneer-free sealed-bid auction protocol with a linear computation and communication complexity $\mathcal{O}(c)$, which makes this suitable choice for study and further implementation in our project. We shall discuss the scheme thoroughly and also further discuss the implementation details of this protocol on a testnet.

3 Ethereum Network

3.1 Block-Chain and it's Applications

A blockchain is a growing list of records, called *Blocks*, that are linked together using cryptographic Hash function. Structurally, a Block-Chain is a linked list that is built with hash pointers instead of pointers. Each block contains a cryptographic hash of the previous block, namely a timestamp, and transaction data (generally represented as a Merkle tree). The timestamp proves that the transaction data existed when the block was published in order to get into its hash. As blocks each contain information about the block previous to it, they form a chain, with each additional block reinforcing the ones before it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks. Blockchains are typically managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks.

Cryptographer David Chaum first proposed a blockchain-like protocol in 1982 [20]. The first blockchain was conceptualized by a person or group of people (interestingly the identity is yet to be public) known as Satoshi Nakamoto [21] in 2008. Nakamoto improved the design in an important way using a Hashcash-like method to timestamp blocks without requiring them to be signed by a trusted party and introducing a difficulty parameter to stabilize rate with which blocks are added to the chain. The design was implemented the following year by Nakamoto as a core component of the cryptocurrency bitcoin, where it serves as the public ledger for all transactions on the network.

As discussed, blockchain is a decentralized, distributed, and oftentimes public, digital blocks consisting of records that is used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks. This allows the participants to verify and audit transactions independently and relatively inexpensively. A blockchain database is managed autonomously using a peer-to-peer network and a distributed time-stamping server. They are authenticated by mass collaboration powered by collective self-interests. Every node in a decentralized system has a copy of the blockchain. Whenever a new transaction is entered, it is transmitted to a network of peer-to-peer computers scattered across the world. Mining nodes validate transactions, add them to the block they are building, and then broadcast the completed block to other nodes. Blockchains use various time-stamping schemes, such as proof-of-work, to serialize changes. Once confirmed to be legitimate transactions, they are clustered together into blocks. The blocks are chained together creating a long history of all transactions that are permanent. And hence, the transaction gets completed.

Block-chain technology can be integrated into multiple areas. The primary use of blockchains was as a distributed ledger for cryptocurrencies. Cryptocurrencies are digital currencies that use blockchain technology to record and secure every transaction. A cryptocurrency (for example, Bitcoin, Ethereum, etc) can be used as a digital form of cash to pay for everything from everyday items to larger purchases like cars and homes. It can be bought using one of several digital wallets or trading platforms, then digitally transferred upon purchase of an item, with the blockchain recording the transaction and the new owner. Block-chains do have several applications in implementing Smart contracts, developing video games, peer-to-peer energy trading, supply chain management, etc.

3.2 Benefits of Ethereum Platform

Ethereum is a decentralized, open-source blockchain with smart contract functionality. Ether, represented as *ETH* is the native cryptocurrency of the ethereum platform. It is the second-largest cryptocurrency by market capitalization, while Bitcoin being the largest one. Though Ethereum is the most actively used blockchain [23].

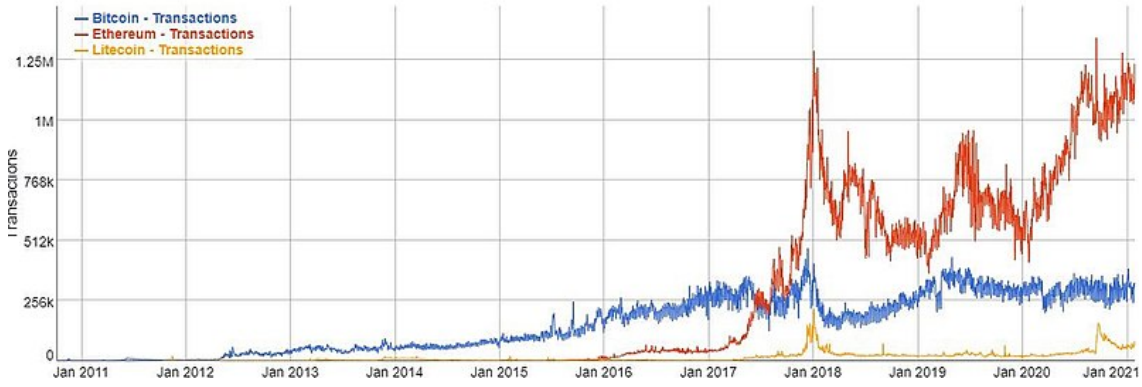

Figure 1: Bitcoin, Ethereum and Litecoin Transactions per day (Jan '11 – Jan '21)

Ethereum was initially described in a white paper in 2013 by Vitalik Buterin [22]. Buterin argued that Bitcoin and blockchain technology could benefit from other applications besides money and needed a scripting language for application development that could lead to attaching real-world assets, such as stocks and property, to the blockchain. The development of Ethereum was crowdfunded in 2014, and the network went live on 30 July 2015, with an initial supply of 72 million coins. The Ethereum Virtual Machine (EVM) can execute scripts and run decentralized applications. Ethereum is massively used for decentralized finance, the creation and exchange of non-fungible tokens, etc.

Ethereum is a permissionless, non-hierarchical network of computers (nodes) which build and come to consensus on an ever-growing series of "blocks", or batches of transactions, known as the blockchain. Each block contains an identifier of the block that it must immediately follow in the chain if it is to be considered valid. Whenever a node adds a block to its chain, it executes the transactions therein in their order, thereby altering the ETH balances and other storage values of Ethereum accounts. These balances and values, collectively known as the state, are maintained on the node's computer separately from the blockchain, in a Merkle tree. Each node communicates with a relatively small subset of the network, known as its peers. Whenever a node wishes to include a new transaction in the blockchain, it sends the transaction to its peers, who then send it to their peers, and so on. A transaction in Ethereum is consists of (I) *From* : A signature from a user-controlled account to authorise the transaction, (II) *To* : The receiver of the transaction and can be either a user-controlled or contract address.(III) *Data* : Contains the contract code to create a new contract or execution instructions for the contract. (IV) *Gas Price* : The conversion rate of purchasing gas using the ether currency. (V) *Total Gas* : The maximum amount of gas that can be consumed by the transaction, and (VI) *Nonce* : A counter that is incremented for each new transaction from an account.

Ether (ETH) is the cryptocurrency generated by the Ethereum protocol as a reward to miners in a proof-of-work system for adding blocks to the blockchain. It is the only currency accepted in the payment of transaction fees, which also go to miners. The block reward together with the transaction fees provide the incentive to miners to keep the blockchain

growing (i.e. to keep processing new transactions). Therefore, ETH is fundamental to the operation of the network. Each Ethereum account has an ETH balance and may send ETH to any other account. The smallest subunit of ETH is known as a Wei and is equal to 1018 ETH.

The Ethereum block-chain presumably provides the highest support for smart contracts creation. Smart contracts are executed by a simple stack-based Turing complete 256-bit virtual machine known as the Ethereum Virtual Machine (EVM). Solidity is the common scripting language for writing smart contracts. We shall implement the scheme described in the last section in Ethereum platform because of the following reasons.

- Ethereum is a public communication channel *i.e.* it is a peer to peer network.
- All communication in Ethereum Network is authenticated as transactions are signed by the voter's Ethereum address.
- Ethereum uses an immutable public ledger to store the necessary information like eligibility white list, private and public keys and bids.
- Ethereum allows anyone with the read access to the public bulletin to verify the execution of the program, and that the protocol is executed correctly.

There are two types of accounts on Ethereum, namely user accounts (also known as externally-owned accounts) and contracts. Both types have an ETH balance, may send ETH to any account, may call any public function of a contract or create a new contract, and are identified on the blockchain and in the state by their address. User accounts are the only type which may create transactions. For a transaction to be valid, it must be signed using the sending account's private key, a 64-character hexadecimal string that should only be known to the account's owner. The signature algorithm used is ECDSA. Importantly, this algorithm allows one to derive the signer's address from the signature without knowing the private key. Gas is a unit of account within the EVM used in the calculation of a transaction fee, which is the amount of ETH a transaction's sender must pay to the miner who includes the transaction in the blockchain. Each type of operation which may be performed by the EVM is hardcoded with a certain gas cost, which is intended to be roughly proportional to the amount of resources (computation and storage) a node must expend to perform that operation. When creating a transaction, the sender must specify a gas limit and gas price. The gas limit is the maximum amount of gas the sender is willing to use in the transaction, and the gas price is the amount of ETH the sender wishes to pay to the miner per unit of gas used. The higher the gas price, the more incentive a miner has to include the transaction in their block, and thus the quicker the transaction will be included in the blockchain. The sender buys the gas up-front, at the start of the execution of the transaction, and is refunded at the end for any gas not used. If at any point the transaction does not have enough gas to perform the next operation, the transaction is reverted but the sender still pays for the gas used. This fee mechanism is designed to mitigate transaction spam, prevent infinite loops during contract execution, and provide for a market-based allocation of network resources.

3.3 Setting up Test Ethereum Network

Test networks exist to ease development and provide developers and companies an easy solution to deliver their product on networks that are not exchanging real value but providing the exact same service. The Ethereum Test Network is a simulation of Ethereum, with the same environment and conditions found on the Ethereum network. Users can test new

projects and changes on this system before deploying them to the real Ethereum network. And because it works the same way as the real one, developers can correct any errors or mistakes here. The system also allows them to see firsthand how the project works and reacts, so they can modify or improve it before it goes live. This process saves companies money because they don't have to use gas. Developers do use Ether and tokens, but these have no real-world value outside the testnet. In this and the next section, we shall be walking through the set up process of the Ethereum development environment along with local private blockchain in Ubuntu and how to create ethereum accounts for the testnet. We thoroughly discuss the required software, the process to start the Ethereum node and performs basic transactions. We shall set up the Test Ethereum Network in the following configuration.

Operating System : `Ubuntu 20.04`
OS Version : `20.04.2 LTS (Focal Fossa)`
Node Version : `v10.19.0`
NVM (Node Version Manager) Version : 6.14.4
Geth (Go Ethereum) version : `1.10.2-stable`

We need Geth to be running in the background. We shall use localhost as host and the Web3 provider endpoint would be `http://127.0.0.1:8545`. At first, we run the following command in ubuntu terminal. This will create the desired environment.

```
geth --allow-insecure-unlock --dev --rpc --ipcpath " /.ethereum/geth.ipc"
--rpcapi="db,eth,net,web3,personal" --rpcport "8545" --rpcaddr "127.0.0.1"
                        --rpccorsdomain "*" console
```

We wish to run our final protocol in a testnet environment, where there will be one main ethereum account, which serves as the *Admin* of the protocol, at least three different *Bidder* accounts and one *Charity* account. No Bidder account can serve as either Admin or Charity account. The main ethereum account will have some initial test ether to perform the experiments we want to have. We design our protocol in a way that the Bidder account must initially have at least one ether *i.e.* to participate in the bidding. The charity account may own no ether. To set up the Ethereum accounts, we follow the mentioned steps.

- With the command given in the last section, while creating the Geth JavaScript console, it automatically creates an ethereum account. This account is assumed to be the *Admin* account and it automatically owns some test ether.
- We can get the list of ethereum accounts by running `eth.accounts` command.
- To get the balance of i-th account, run `eth.getBalance(eth.accounts[i-1])`.
- To create each new ethereum account, we run `personal.newAccount()`. We also need to put a passphrase for each account. The passphrase for the main account *i.e.* `eth.accounts[0]` is empty. Hence, we do not need to put any passphrase as Admin.
- For further transactions, we need to unlock the newly created accounts. The command is `personal.unlockAccount(<account address>,<passphrase>,<unlock duration>)`
- Since to perform in bidding, each Bidder account needs at least one ether. Hence, we send enough ether from the Admin account to newly created Bidder account. The command `eth.sendTransaction(from:eth.accounts[i],to:eth.accounts[j],value:<amount>)` is used to send ether from account i to account j.

The picture in the following page holds an working example of the above mentioned details.

```
> eth.accounts
["0xd50db2d03b0acdfcde9549ba7eb056586b0083d0", "0x0cd4c89f7c20cfe8028a6516b818f9c8a1cc1bb0"]
>
> personal.newAccount()
Passphrase:
Repeat passphrase:
INFO [04-26|23:26:27.283] Your new key was generated                address=0xbeE35a528DCb827A57C50F36F21053c3d8785868
WARN [04-26|23:26:27.283] Please backup your key file!              path=/tmp/go-ethereum-keystore994968692/UTC--2021-04-26T17-56-25.809233717Z-
-bee35a528dcb827a57c50f36f21053c3d8785868
WARN [04-26|23:26:27.283] Please remember your password!
"0xbee35a528dcb827a57c50f36f21053c3d8785868"
>
> personal.unlockAccount("0xbee35a528dcb827a57c50f36f21053c3d8785868","",1000)
true
>
> eth.accounts
["0xd50db2d03b0acdfcde9549ba7eb056586b0083d0", "0x0cd4c89f7c20cfe8028a6516b818f9c8a1cc1bb0", "0xbee35a528dcb827a57c50f36f21053c3d8785868"]
>
> eth.getBalance(eth.accounts[2])
0
>
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[2], value : 21480000000000000000})
INFO [04-26|23:27:28.580] Submitted transaction                     hash=0xf134e1f834bfc2530c492b2d4ec6469831a9ceafdfb3aba40feb6dd8eafb24cc from
=0xd50DB2D03B0AcDfcde9549Ba7eB056586B0083d0 nonce=1 recipient=0xbeE35a528DCb827A57C50F36F21053c3d8785868 value=21480000000000000000
"0xf134e1f834bfc2530c492b2d4ec6469831a9ceafdfb3aba40feb6dd8eafb24cc"
> INFO [04-26|23:27:28.580] Commit new mining work                   number=2 sealhash="14878d…12fef2" uncles=0 txs=1 gas=21000 fees=2.1e-14 el
apsed="246.861µs"
INFO [04-26|23:27:28.580] Successfully sealed new block            number=2 sealhash="14878d…12fef2" hash="f80789…cbd1de" elapsed="377.84µs"
INFO [04-26|23:27:28.580]  mined potential block                   number=2 hash="f80789…cbd1de"
INFO [04-26|23:27:28.581] Commit new mining work                   number=3 sealhash="80bbb3…79a0af" uncles=0 txs=0 gas=0     fees=0      elap
sed="231.599µs"
INFO [04-26|23:27:28.581] Sealing paused, waiting for transactions

> eth.getBalance(eth.accounts[2])
21480000000000000000
>
> eth.getBalance(eth.accounts[0])
1.157920892373161954235709850086879078532699846656405640394532880079131296399 27e+77
```

Figure 2: Working Example of forming Ethereum Accounts

4 Earlier Developments towards SEAL Protocol

As we have discussed, SEAL protocol was the first auctioneer-free sealed-bid auction protocol with a linear computation and communication complexity, published in 2019 by Samiran Bag, Feng Hao, Siamak F. Shahandashti, and Indranil G. Ray. This protocol uses a modified version of the 2-Round Autonomous Veto Network Protocol by Feng Hao and Piotr Zieliński as the building blocks for the protocol. Notably, Hao-Zielinski's Anonymous Veto network (AV-net) protocol is an alternative solution to the Dinning Cryptographer problem. In this section, we briefly discuss all the protocols which eventually motivated the next protocol of the sequence to finally reach the SEAL protocol.

4.1 Dining Cryptographers Problem

The dining cryptographers problem [5] studies how to perform a secure multi-party computation of the boolean-OR function. The formal problem statement is the following.

Problem Statement : Three cryptographers are having dinner in a table, where the bill to be paid anonymously. US National Security Agency (NSA) or one of the cryptographers may pay the bill. They respect each other's right to make an anonymous payment, but they wonder if NSA is paying for them or one of them is paying.

David Chaum first proposed this problem in the early 1980s and used it as an illustrative example to show that it was possible to send anonymous messages with unconditional sender and recipient intractability. Anonymous communication networks based on this problem are often referred to as DC-nets. In the next subsection, we describe how Chaum proposed a 2-stage protocol [1] to find if NSA pays or not.

4.2 Dining Cryptographer Network (DC Net) Protocol

David Chaum proposed this solution named Dinning Cryptographer's Network, in short DC-Net in 1988 through a research paper in Journal of Cryptology. We primarily focus on the problem with three cryptographers, but this can be generalized further. The solution is a two-round protocol, described in the following.

Round 1. Each cryptographer flips an unbiased coin so that only him and the cryptographer on his right hand side can see the outcome of the toss.

Round 2. Each cryptographer then states aloud whether the two coins he can see (one he flipped, one his left-hand neighbor cryptographer flipped) fell on the same side or on different sides. The cryptographer basically XORs the outputs (without loss of generality, assume getting head is 1 and getting tail is 0) and tell the result. If one of the cryptographers is the payer, he states the opposite of what he gets as the result and if he didn't pay, then he announces the actual result of the XOR.

Tally. Finally we XOR all the announced bits. An odd number of differences uttered at the table indicates that a cryptographer is paying, while an even number indicates that NSA is paying. Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is. A working example of this method is illustrated in the following figure.

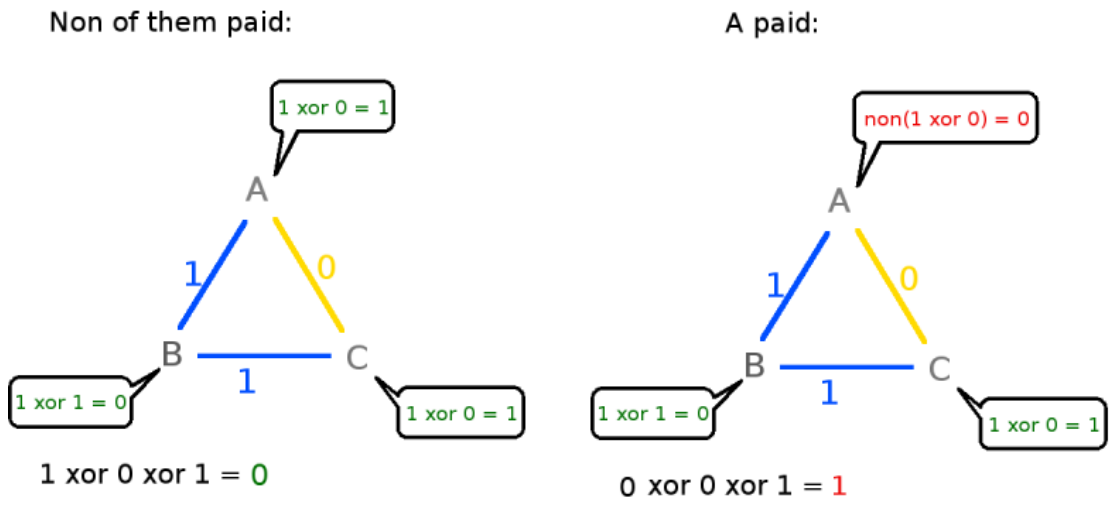

Figure 3: Brief Illustration of DC Net Protocol

The DC Net protocol is indeed simple and elegant. But, it also has several limitations.

- If two (any even number for generalized case) cryptographers have paid for the dinner, their messages will cancel each other out, and the final XOR result will be 0.
- A dishonest cryptographer may send random bits instead of the correct result of the XOR. This occurs because the original protocol was designed without using any public key technology and lacks reliable mechanisms to check whether participants are honest.
- Though DC Net protocol is unconditionally secure, but that depends on the assumption that there exists unconditionally secure channels between pairs of the participants, which is not easy to achieve in practice.

4.3 Anonymous Veto Network (AV Net) Protocol

The anonymous veto network (or AV-net) is is an alternative solution for the Dining Cryptographers Problem. Hence, like the DC Net protocol, this also serves as a multi-party secure computation protocol to compute the boolean-OR function. AV Net was first proposed by Feng Hao and Piotr Zieliński in 2006. We can extend the Dinning Cryptographer Problem from three to any finite number of participants. The AV-net protocol will check if anyone of the n cryptographers have paid the bill or not anonymously. AV-nets require no secrecy channels, have no message collisions, and are more resistant to disruptions. It only assumes an authenticated broadcast channel available to every participant, which can be achieved in the practical world pretty easily.

Construction of AV Net protocol assumes certain mathematical assumptions. Let G be a finite cyclic group of prime order q, in which Decisional Diffie–Hellman problem is intractable. Let g be a generator in G. n-many participants $P_1, \ldots, P_n$ agree on (G, g).

Round 1.

- Each participant P_i selects a 'secret' random value $x_i \leftarrow_R \mathbb{Z}_q$.
- Each P_i publishes g^{x_i} and a ZKP for x_i

After Round 1 gets over, each P_i checks the validity of other participant's ZKPs and computes

$$g^{y_i} = \prod_{j=1}^{i-1} g^{x_j} / \prod_{j=i+1}^{n} g^{x_j} = g^{\sum_{j<i} x_j - \sum_{j>i} x_j} \text{ i.e. } y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$$

Round 2.

- Depending on the participant wants to veto or not, he calculates c_i.

$$c_i = \begin{cases} x_i & P_i \text{ sends 0 (No veto)} \\ r_i \leftarrow_R \mathbb{Z}_q & P_i \text{ sends 1 (Veto)} \end{cases}$$

- Every participant broadcasts $g^{c_i y_i}$ and a ZKP for c_i.

Zero Knowledge Proofs.

In Round 1, P_i publishes g^{x_i} ZKP for x_i and in Round 2, publishes $(g^{y_i})^{c_i}$ ZKP for c_i. We can use Schnorr's signature for the ZKPs of the knowledge of the exponents in both the times. For knowledge of x, Schnorr's Signature works in the following way. Assume $H : \{0,1\}^* \to \mathbb{Z}_q$ to be publicly agreed, secure Hash function.

- P_i chooses $v \leftarrow_R \mathbb{Z}_q$
- P_i calculates $z = H(g, g^v, g^x, i)$ and $r = v - xz$
- P_i sends (g^v, r, z) to verifier.
- Verifier check if $g^v = g^r g^{xz}$

Note : $g^v = g^r (g^x)^z \iff r = v - xz$ and since v is chosen randomly and H is secure Hash function, one cannot deduce x_i from published information, but ownership is established.

Tally. Finally we compute $\prod_{i=1}^{n} g^{c_i y_i}$ and check if anyone vetoes or not.

We claim that for y_i as defined above, $\sum_i x_i y_i = 0$. Note that,

$$y_i = \sum_{j<i} x_j - \sum_{j>i} x_j \implies x_i y_i = \sum_{j<i} x_i x_j - \sum_{j>i} x_i x_j \implies \sum_i x_i y_i = \sum_i \Big(\sum_{j<i} x_i x_j - \sum_{j>i} x_i x_j \Big)$$

$$= \sum_i \sum_{j<i} x_i x_j - \sum_i \sum_{j>i} x_i x_j = \sum_{j<i} \sum x_i x_j - \sum_{j>i} \sum x_i x_j = \sum_{j<i} \sum x_i x_j - \sum_{j>i} \sum x_j x_i = 0$$

If no one vetoes, then $c_i = x_i \ \forall i$ and hence, $\prod_{i=1}^{n} g^{c_i y_i} = \prod_{i=1}^{n} g^{x_i y_i} = g^{\sum_{i=1}^{n} x_i y_i} = 1$.
If one or more participants vetoes, $\prod_{i=1}^{n} g^{c_i y_i} \neq 1$ with very high probability.

We have bypassed the first drawback we mentioned for DC Net protocol. Because of the existence of the ZKPs, the participants can not be dishonest and by construction,, it is clear that we do not need any secret channel between the participants. Thus, the one-bit message has been sent anonymously without having the drawbacks we encountered for DC Net.

4.4 Modified Anonymous Veto Network Protocol

Let G be a (finite) cyclic group of prime order p, in which the Decisional Diffie–Hellman problem is intractable. Let g be a random generator in G. All computations in G are modular operations with respect to a prime modulus q.

Aim of the protocol : A group of n bidders $V_1, V_2, \ldots V_n$ wish to find out if there is one voter who would like to veto a motion *i.e.* to securely compute $\bigvee_{i=1}^{n} v_i$, the logical-OR function of a number of input bits $v_i \in \{0, 1\}$, each (secret) bit coming from a separate V_i.

Round 1.

- Each bidder V_i selects two random values $x_i \leftarrow_R \mathbb{Z}_p$ and $r_i \leftarrow_R \mathbb{Z}_p$
- V_i computes $X_i = g^{x_i}$ and $R_i = g^{r_i}$
- Each V_i publishes (X_i, R_i) and NIZKPs for x_i and r_i using Schnorr's Signature.

After Round 1 gets finished, each V_i checks the validity of NIZKPs of others.

Round 2.

- V_i computes $Y_i = \prod_{j=1}^{i-1} X_j / \prod_{j=i+1}^{n} X_j$
- V_i holds a secret bid $v_i \in \{0, 1\}$
- V_i posts encrypted ballot $b_i = \begin{cases} Y_i^{x_i} \text{ if } v_i = 0 \\ R_i^{x_i} \text{ if } v_i = 1 \end{cases}$ on the bulletin board with NIZKP of b_i

NIZKP for Round 1.

While publishing $(X_i, R_i) = (g^{x_i}, g^{r_i})$ in Round 1, each bidder V_i uses Schnorr's signature to prove the knowledge of x_i and r_i. The idea is exactly similar to the ZKP of AV Net, only difference is that we use an extra random variable here. The process is described in the following. Suppose, $H : \{0, 1\}^* \to \mathbb{Z}_q$ be publicly agreed, secure Hash function.

- P_i chooses $v \leftarrow_R \mathbb{Z}_q$
- P_i calculates $z_1 = H(g, g^v, g^{x_i}, i), w_1 = v - x_i z_1;\ z_2 = H(g, g^v, g^{r_i}, i), w_2 = v - r_i z_2$
- P_i sends $(g^v, z_1, w_1, z_2, w_2)$ to verifier.
- Verifier check if $g^v = g^{w_1}(g^{x_i})^{z_1}$ and $g^v = g^{w_2}(g^{r_i})^{z_2}$

NIZKP for Round 2.

While publishing b_i in Round 2, P_i posts a NIZKP on the well-formedness of b_i, which is a disjunctive proof of two sub-statements.

- P_i needs to prove $(b_i = Y_i^{x_i}) \vee (b_i = R_i^{x_i})$
- Proving $b_i = Y_i^{x_i}$ is equivalent to prove that $(X_i, Y_i, b_i = Y_i^{x_i})$ is a DDH tuple. To prove that $(X_i, Y_i, b_i = Y_i^{x_i})$ is a DDH tuple, we need to prove that $(g^{x_i}, g^{y_i}, (g^{y_i})^{x_i})$ and $(g^{x_i}, g^{y_i}, g^{z_i})$ are computationally indistinguishable, where $x_i \leftarrow_R \mathbb{Z}_p$ for $1 \le i \le n$, $y_i = \sum_{j<i} x_j - \sum_{j>i} x_j$ and $z_i \leftarrow_R \mathbb{Z}_p$.

- Proving $b_i = R_i^{x_i}$ is equivalent to prove that $(X_i, R_i, b_i = R_i^{x_i})$ is a DDH tuple. To prove $(X_i, R_i, b_i = R_i^{x_i})$ is a DDH tuple, we need to prove that $(g^{x_i}, g^{r_i}, (g^{r_i})^{x_i})$ and $(g^{x_i}, g^{r_i}, g^{z_i})$ are computationally indistinguishable, where $x_i, r_i, z_i \leftarrow_R \mathbb{Z}_p$.

Tally. Note that, if nobody vetoes *i.e.* $v_i = 0 \; \forall i \in \mathbb{N}_{\leq n}$ then,

$$b_i = Y_i^{x_i} = (\prod_{j=1}^{i-1} X_j \; / \; \prod_{j=i+1}^{n} X_j)^{x_i} = \left(g^{\sum_{j<i} x_j - \sum_{j>i} x_j}\right)^{x_i} = g^{x_i.(\sum_{j<i} x_j - \sum_{j>i} x_j)}$$

$$\implies \prod_{i=1}^{n} b_i = g^{\sum \sum_{j<i} x_i x_j - \sum \sum_{j>i} x_i x_j} = g^0 = 1 \text{ since, } \sum \sum_{j<i} x_i x_j - \sum \sum_{j>i} x_i x_j = 0$$

After all the V_is submit their encrypted ballot, we compute $B = \prod_{i=1}^{n} b_i$ to decide is there is any non-zero bid or not. We already have, if we have all 0-bids, then $B = 0$. If at least one input bid v_i is 1, then B would a random element in G, with the randomness of its coming from r_i. Basically, in the later case, $B \neq 1$ with very good probability. Noticing B, we can guess if there is a single non-zero input for v_i or not, which is enough to compute $\bigvee_{i=1}^{n} v_i$. Hence, the logical-OR of all input bits has been securely computed by all participants without revealing the value of each individual bit.

4.5 Differences between AV-net and Modified AV-net Protocols

.

- In AV-net protocol, we needed to choose one random value from $\mathbb{Z}_q$. Here we are choosing two random values $(x_i, r_i) \in \mathbb{Z}_p^2$. This requires more computation from each participant, but the (asymptotic) computation and communication complexity remains the same as AV-net.
- In AV-net, the 'veto' vote is encoded by raising a pre-defined generator to the power of a random variable, while in the modified veto protocol, the '1' vote is encoded by raising a random generator to the power of a pre-defined exponent x_i. This modification allows us to effectively integrate the veto protocol into the e-auction scheme as some zero-knowledge proofs will require proving the equality of the exponents.

5 SEAL Protocol

SEAL protocol is an auctioneer-free sealed-bid e-auction protocol, which operates in a decentralized setting, where bidders jointly compute the maximum bid while preserving the privacy of losing bids. As we mentioned earlier, no secret channel between participants are used and all operations are publicly verifiable. Upon learning the highest bid, the winner comes forward with a proof to prove that (s)he is the real winner. Any third party verifier can also verify if there is unique winner or there is a tie.

5.1 Basic Overview of the Protocol

A basic building block of the e-auction protocol is the Modified Anonymous Veto Network (AV-Net) Protocol [13], which securely computes the logical-OR of binary inputs without revealing each individual bit. Notably, Modified AV-Net protocol is a modification of Hao-Zielinski's AV-net protocol [9], which we described in the earlier section. We first describe the Modified AV-Net Protocol and then explain the way the main SEAL protocol has been sketched using this modified version of AV Net protocol.

Consider G to be the same group, which we assumed for the Modified AV-Net Protocol. As described, G is a finite cyclic group of p elements, in which the Decisional Diffie–Hellman problem is intractable. Say, g is a random generator of G. All computations in G are modular operations with respect to a prime modulus q.

Assume there are n many bidders, namely $V_1, V_2, \ldots, V_n$. Each bidder V_i put their respective bid p_i on a public bulletin board. The binary representation of each bid contains c bits. The bid price by bidder V_i is hence expressed in the binary form as $p_i = p_{i1} \mid\mid p_{i2} \mid\mid \ldots \mid\mid p_{ic}$, where $\mid\mid$ denotes concatenation and $p_{ij} \in \{0, 1\}$, with p_{i1} being the most significant bit and p_{ic} the least significant bit.

Aim of the protocol : The SEAL e-auction protocol consists of two phases. In the first phase *i.e.* the *Commit Phase*, n bidders $V_1, V_2, \ldots, V_n$ commit their respective bids $p_1, p_2, \ldots, p_n$ on a public bulletin board. In the second phase, all bidders jointly compute the maximum bid bit wise, starting from the most significant bit position on the left, without revealing the other bids. This phase can be divided into two stages for better explanation. SEAL protocol has c iterations. (c is number of bits in binary representation of bid-prices)

5.2 Phase 1 : Commit Phase

In this phase, for each $1 \leq i \leq n$, the bidder V_i commit their bid p_i to the public bulletin board, where binary representation of p_i is $p_{i1} \mid\mid p_{i2} \mid\mid \ldots \mid\mid p_{ic}$, as mentioned. In order to do this, V_i computes c many commitments ϵ_{ij}, each one for a bit of p_{ij} , for $j \in \{1, \ldots, c\}$. The process is as follows.

- For $1 \leq j \leq c$, the bidder V_i chooses $\alpha_{ij} \leftarrow_R \mathbb{Z}_q$ and $\beta_{ij} \leftarrow_R \mathbb{Z}_q$
- V_i computes c individual commitments $\varepsilon_{ij} = \langle g^{\alpha_{ij}\beta_{ij}} g^{p_{ij}}, g^{\alpha_{ij}}, g^{\beta_{ij}} \rangle$
- V_i also posts a NIZK proof of well formedness of each committed bit.

NIZKPs for Phase I.

Each bidder V_i also posts a NIZK proof of well formedness of each committed bit in the form of ϵ_{ij}, where NIZKP shows $p_{ij} \in \{0, 1\}$, without revealing the correct value. Suppose, each commitment of our scheme is of the form $\varepsilon = \langle g^{\alpha\beta} g^v, g^\alpha, g^\beta \rangle$, where v is the committed bit. Each bidder provides c many commitments, each for exactly one of the c many bits in the binary representation of the bid-price of that bidder. The construction of the NIZK proof of well-formedness of the commitment has two parts. First, given g^α and g^β, the bidder (*i.e.* the prover here) needs to prove knowledge of α and β using Schnorr's signature (well-formedness of public keys). Then the statement that the prover needs to show is that ε is well formed for $v \in \{0, 1\}$. We explain both the NIZKP in the following.

NIZKP of well-formedness of Public Keys.

We have seen that to form the commitments, the bidder needs to select α and β and publishes g^α and g^β on the public bulletin board. Now to prove the knowledge of α and β, the prover (*i.e.* the bidder) uses the following NIZKP using Schnorr's Signature.

Suppose, the prover V_i has x and she publishes $X = g^x$ publicly. Now she wants to prove the knowledge of $x = \log_g X$. In this proof, we need a random challenge of the obtained through feeding the commitment and all other available argument into a random oracle. We hereby introduce a Hash function $\mathcal{H}$, which takes certain inputs and outputs a random challenge z. Here we present the game based proof to understand the ZKP better. Note that, since we need a NIZKP, only change we shall make is that instead of keeping the game between Prover and verifier that at the last part, the prover will not send any information to verifier and the code itself checks if the verification equation holds or not.

`Prover`		`Verifier`
Chooses $r \leftarrow_R \mathbb{Z}_p$		
Computes $\varepsilon = g^r$		
Computes $ch = \mathcal{H}(g, g^r, g^x, i)$		
Computes $\rho = r - x \ . \ ch$		
	$\xrightarrow{\varepsilon,\ \rho,\ ch}$	
		$\varepsilon \stackrel{?}{=} g^\rho X^{ch}$

Correctness : The NIZKP is accepted if the verification equation holds true *i.e.* $\varepsilon = g^\rho X^{ch}$. The correctness proof lies to the fact that $\varepsilon = g^r = g^{\rho + x \ \cdot \ ch} = g^\rho (g^x)^{ch} = g^\rho X^{ch}$.

In the above ZKP, we call ch as challenge, ε as commitment and ρ as response. The prover needs to do just one exponentiation for generating the proof that contains one challenge, one commitment and one response, 3 parameters in total. The verifier needs to do 2 exponentiations to verify the proof.

NIZKP of well-formedness of Commitments.

We assumed that each commitment is of the form $\varepsilon = \langle g^{\alpha\beta}g^v, g^\alpha, g^\beta\rangle$, where v is the committed bit. Each bidder provides c such commitments. Now to prove that ε is well-formed *i.e.* $v \in \{0, 1\}$, the prover *i.e.* the bidder needs to show the following statement holds.

$$\sigma \equiv (\phi = g^{\alpha\beta} \wedge A = g^\alpha \wedge B = g^\beta) \vee (\phi = g^{\alpha\beta}g \wedge A = g^\alpha \wedge B = g^\beta)$$

Note that, depending upon $v = 0$ or $v = 1$, we have $g^{\alpha\beta}g^v = g^{\alpha\beta}$ or $g^{\alpha\beta} = g^{\alpha\beta}g$, respectively. Hence, only one of the statement can be true. WLOG we assume that the first statement is correct *i.e.* $\phi = g^{\alpha\beta} \wedge A = g^\alpha \wedge B = g^\beta$. So, the prover needs to provide a real proof for this statement and a simulated proof for the other statement $\phi = g^{\alpha\beta}g \wedge A = g^\alpha \wedge B = g^\beta$. the NIZKP to prove this is following. As similar to the above NIZKP, instead of keeping the game between Prover and verifier that at the last part, the prover will not send any information to verifier and the code itself checks if the verification equations hold or not. As before, $\mathcal{H}$ is publicly agreed secure Hash function.

Prover		Verifier
Chooses $r_1 \leftarrow_R \mathbb{Z}_p$		
Computes $\varepsilon_{11} = g^{r_1}$		
Computes $\varepsilon_{12} = (g^\beta)^{r_1}$		
Chooses $ch_2 \leftarrow_R \mathbb{Z}_p$		
Chooses $\rho_2 \leftarrow_R \mathbb{Z}_p$		
Computes $\varepsilon_{21} = g^{\rho_2}(g^\alpha)^{ch_2}$		
Computes $\varepsilon_{22} = (g^\beta)^{\rho_2}(\phi/g)^{ch_2}$		
Computes $ch = \mathcal{H}(g, g^{r_1}, g^v, i)$		
Computes $ch_1 = ch - ch_2$		
Computes $\rho_1 = r_1 - \alpha.ch_1$		
	$\xrightarrow{\varepsilon_{11},\ \varepsilon_{12},\ \varepsilon_{21},\ \varepsilon_{22},\ \rho_1,\ \rho_2,\ ch_1,\ ch_2}$	
		$\varepsilon_{11} \stackrel{?}{=} g^{\rho_1} \,.\, (g^\alpha)^{ch_1}$
		$\varepsilon_{12} \stackrel{?}{=} (g^\beta)^{\rho_1} \,.\, (\phi)^{ch_1}$
		$\varepsilon_{21} \stackrel{?}{=} g^{\rho_2} \,.\, (g^\alpha)^{ch_2}$
		$\varepsilon_{22} \stackrel{?}{=} (g^\beta)^{\rho_2} \,.\, (\phi/g)^{ch_2}$

Correctness : If all 4 relations hold then the proof is accepted. This works because

- $\varepsilon_{11} = g^{r_1} = g^{\rho_1+\alpha.ch_1} = g^{\rho_1}.(g^\alpha)^{ch_1}$
- $\varepsilon_{12} = (g^\beta)^{r_1} = (g^\beta)^{\rho_1+\alpha.ch_1} = (g^\beta)^{\rho_1}.(g^{\alpha\beta})^{ch_1} = (g^\beta)^{\rho_1}.(\phi)^{ch_1}$
- $\varepsilon_{21} = g^{\rho_2}.(g^\alpha)^{ch_2}$ and $\varepsilon_{22} = (g^\beta)^{\rho_2}.(\phi/g)^{ch_2}$, by construction.

The proof consists of 4 commitments, 2 challenges and 2 responses, making the space complexity equal to 8. The prover needs to do 6 exponentiations for generating the proof. The verifier needs to do 8 exponentiations for verifying them.

5.3 Phase 2 : Computing the Highest Bid

The second phase of the protocol is divided into two stages. As mentioned earlier, the Modified AV-Net protocol is used as a basic building block to compute the logical-OR of the input bits. Assume that, some bidder V_i bids a value of $p_i = p_{i1} \,||\, p_{i2} \,||\, \ldots \,||\, p_{ic}$, where $p_{ij} \in \{0, 1\}$. Say, for example some bid is of value 11, whose binary expression would be 01011, assuming $c = 5$.

Now, at each iteration of the SEAL protocol, each bidder V_i uses a bit d_{ij} (for $1 \leq j \leq c$) as the input to the mentioned Modified Anonymous Veto protocol, without revealing the committed bit. Now initially we consider $d_{ij} = p_{ij}$. All bidders can easily compute the logical OR of all d_{ij} bits for any of the bit positions using the described protocol.

Define $T_j = d_{1j} \vee d_{2j} \vee \ldots \vee d_{nj}$. We call a position j to be *Deciding Position* if $T_j = 1$ and *Non-Deciding Position* if $T_j \neq 1$. The first deciding position is called a *Junction*. Notably until the *Junction*, we keep $d_{ij} = p_{ij}$ and after *Junction*, the bidder uses $d_{ij} = p_{ij} \wedge d_{i\overleftarrow{j}}$, where $d_{i\overleftarrow{j}}$ is the bit in previous deciding position. Consider the following example with bid value $p_1 = 01011$. Here, Junction (**J**) position is the second position, hence $d_{10} = p_{10} = 0$ and $d_{11} = p_{11} = 1$. But, $d_{12} = p_{12} \wedge d_{1\overleftarrow{2}} = p_{12} \wedge d_{11} = 0$, $d_{13} = p_{13} \wedge d_{1\overleftarrow{3}} = p_{12} \wedge d_{12} = 0$ and finally, $d_{14} = p_{14} \wedge d_{1\overleftarrow{4}} = p_{12} \wedge d_{12} = 0$. We call it *Stage I* until the computation of the junction position is done, and the later stage is termed as *Stage II*.

	Decimal	*Binary*			**J**	**D**		**D**
Committed Bid	12	01100	0	1	1	0	0	
Submitted Bid			0	1	1	0	0	
Committed Bid	11	01011	0	1	0	1	1	
Submitted Bid			0	1	0	0	0	
Committed Bid	13	01101	0	1	1	0	1	
Submitted Bid			0	1	1	0	1	
Committed Bid	7	00111	0	0	1	1	1	
Submitted Bid			0	0	0	0	0	
Highest Bid	**13**	01101	**0**	**1**	**1**	**0**	**1**	

Index : **D** - Deciding position, **J** - Junction (1st Deciding) Position.

Stage I : Computation till Junction Position.

Bidders start from the most significant bit position *i.e.* $j = 1$ and move to the less significant bit positions bit-by-bit until they reach a junction, where Stage I terminates. As stated above, at the bit position j, bidders apply the Modified AV-Net protocol in following way with private binary inputs d_{ij} , for $1 \leq j \leq n$, with a ZKP that the input bit is the same as the committed one *i.e.* $d_{ij} = p_{ij}$.

Round 1.

- Each bidder V_i selects two private keys $x_{ij} \leftarrow_R \mathbb{Z}_p$ and $r_{ij} \leftarrow_R \mathbb{Z}_p$.
- V_i computes the public keys $X_{ij} = g^{x_{ij}}$ and $R_{ij} = g^{r_{ij}}$
- Each V_i publishes $Pub_{ij} = (X_{ij}, R_{ij})$ and NIZKPs for $x_{ij} = \log_g X_{ij}$ and $r_{ij} = \log_g R_{ij}$.

Round 2.

- Each V_i computes $Y_{ij} = g^{y_{ij}} = \prod_{k=1}^{i-1} g^{x_{kj}} / \prod_{k=i+1}^{n} g^{x_{kj}}$
- V_i already holds $p_{ij} \in \{0, 1\}$, corresponding to his choice of p_i
- V_i posts encrypted ciphertext, known as *cryptogram*

$$b_{ij} = \begin{cases} Y_{ij}^{x_{ij}} = g^{x_{ij}y_{ij}} & \text{if } p_{ij} = 0; \text{ (0-cryptogram)} \\ R_{ij}^{x_{ij}} = g^{x_{ij}r_{ij}} & \text{if } p_{ij} = 1; \text{ (1-cryptogram)} \end{cases}$$

on the bulletin board along with π_{ij}, a NIZKP of well-formedness of b_{ij}.

Tally for Stage I.

- Compute $\prod_{i=1}^{n} b_i$, the product of all the cryptograms.
- If $\prod_{i=1}^{n} b_i = 1$, the we conclude $T_j = 0$.
- If $\prod_{i=1}^{n} b_i \neq 1$, the we conclude $T_j = 1$.
- Hence, we get $T_1 T_2 \dots T_j$ as final output, where j is the *Junction* Position.

After the two rounds, all the bidders verify that the correctness of NIZKPs and compute T_j, the logical-OR of the input bits d_{ij} for the jth position. This logical-OR computation is realized by multiplying b_{ij}s and comparing the result against 1. Basically, if $\prod_{i=1}^{n} b_i = 1$, then all bit inputs were 0 and hence, $T_j = 0$ and if $\prod_{i=1}^{n} b_i \neq 1$, then at least one bit input was 1 and hence, $T_j = 1$. The explanation is similar to the one in the computation part of the Modified AV-Net protocol.

Finally, if $T_j = 0$ *i.e.* the bit position is non-deciding position, all bidders remain in *Stage 1* and move on to compute the logical-OR of the next bit position. If $T_j = 1$, which means the junction is reached and all bidders move to *Stage II*.

NIZKP for Stage I.

Note that, In *Round I*, every bidder posts $Pub_{ij} = (X_{ij}, R_{ij})$ and NIZKPs for $x_{ij} = \log_g X_{ij}$ and $r_{ij} = \log_g R_{ij}$. Both these NIZKPs are constructed based on Schnorr's Signature. In previous subsection, we discussed the same NIZKPs to prove the well-formedness of Public Keys. We have seen that how the bidder prove the knowledge of x while posting g^x on public bulletin.

In *Round II*, the bidder posts a NIZKP of the well-formedness of the encrypted cryptogram. For proving the well-formedness of b_{ij} in the *Stage I* of the protocol, the NIZK proof of $d_{ij} = p_{ij}$ basically needs to prove the logical statement $(d_{ij} = 0 \wedge p_{ij} = 0) \vee (d_{ij} = 1 \wedge p_{ij} = 1)$. Basically, in iteration j, we need to prove that

$$\sigma \equiv \big((b_{ij} = g^{x_{ij}y_{ij}} \wedge X_{ij} = g^{x_{ij}} \wedge Y_{ij} = g^{y_{ij}}) \bigwedge (c_{ij} = g^{\alpha_{ij}\beta_{ij}} \wedge A_{ij} = g^{\alpha_{ij}} \wedge B_{ij} = g^{\beta_{ij}})\big)$$

$$\bigvee \big((b_{ij} = g^{x_{ij}r_{ij}} \wedge X_{ij} = g^{x_{ij}} \wedge Y_{ij} = g^{r_{ij}}) \bigwedge (c_{ij} = g^{\alpha_{ij}\beta_{ij}} g \wedge A_{ij} = g^{\alpha_{ij}} \wedge B_{ij} = g^{\beta_{ij}})\big)$$

We can write the statement in following way for simplicity.

$$\sigma \equiv \big((B = g^{xy} \wedge X = g^x \wedge Y = g^y) \bigwedge (c = g^{\alpha\beta} \wedge \bar{A} = g^\alpha \wedge \bar{B} = g^\beta)\big)$$
$$\bigvee \big((B = g^{xr} \wedge X = g^x \wedge Y = g^r) \bigwedge (c = g^{\alpha\beta} g \wedge \bar{A} = g^\alpha \wedge \bar{B} = g^\beta)\big)$$

This is an instance of one-out-of-two statement. Hence, only one of the two constituent sub-statements can be true. We suppose the first sub-statement is true, i.e. $(B = g^{xy} \wedge X = g^x \wedge Y = g^y) \bigwedge (c = g^{\alpha\beta} \wedge \bar{A} = g^\alpha \wedge \bar{B} = g^\beta)$ holds. The ZKP is the following.

Prover		Verifier
Chooses $r_{11}, r_{12} \leftarrow_R \mathbb{Z}_p$		
Computes $\varepsilon_{11} = g^{r_{11}}$		
Computes $\varepsilon_{12} = g^{r_{12}}$		
Computes $\varepsilon_{13} = Y^{r_{11}}$		
Computes $\varepsilon_{14} = \bar{B}^{r_{12}}$		
Chooses $\rho_{21}, \rho_{22} \leftarrow_R \mathbb{Z}_p$		
Chooses $ch_2 \leftarrow_R \mathbb{Z}_p$		
Computes $\varepsilon_{21} = g^{\rho_{21}} X^{ch_2}$		
Computes $\varepsilon_{22} = g^{\rho_{22}} \bar{A}^{ch_2}$		
Computes $\varepsilon_{23} = R^{\rho_{21}} B^{ch_2}$		
Computes $\varepsilon_{24} = \bar{B}^{\rho_{22}} (c/g)^{ch_2}$		
$ch \leftarrow$ Grand Challenge of NIZKP		
Computes $ch_1 = ch - ch_2$		
Computes $\rho_{11} = r_{11} - x.ch_1$		
Computes $\rho_{12} = r_{12} - \alpha.ch_1$		
	$\xrightarrow{commitments,\ challenges,\ responses}$	
		$\varepsilon_{11} \stackrel{?}{=} g^{\rho_{11}} \,.\, X^{ch_1}$
		$\varepsilon_{12} \stackrel{?}{=} g^{\rho_{12}} \,.\, \bar{A}^{ch_1}$
		$\varepsilon_{13} \stackrel{?}{=} Y^{\rho_{11}} \,.\, B^{ch_1}$
		$\varepsilon_{14} \stackrel{?}{=} \bar{B}^{\rho_{12}} \,.\, c^{ch_1}$
		$\varepsilon_{21} \stackrel{?}{=} g^{\rho_{21}} \,.\, X^{ch_2}$
		$\varepsilon_{22} \stackrel{?}{=} g^{\rho_{22}} \,.\, \bar{A}^{ch_2}$
		$\varepsilon_{23} \stackrel{?}{=} R^{\rho_{21}} \,.\, B^{ch_2}$
		$\varepsilon_{24} \stackrel{?}{=} \bar{B}^{\rho_{22}} \,.\, (c/g)^{ch_2}$

As stated earlier, we make this NIZKP by involving no so-called Verifier and keeping the check of the verifying equations happen by the code itself. If the above 8 relations hold, the NIZK proof is authentic. A bidder needs to do 12 exponentiations for computing the above NIZK proof arguments. The proof itself consists of 8 commitments, two challenges and 4 responses. Hence, the space complexity of the proof is 14. Moreover, a verifier needs to do 16 exponentiations for checking all the arguments of this NIZK proof. Similarly a NIZK proof can be constructed if the second statement holds true.

Stage II : Computation after Junction Position

The later stage is almost the similar to the earlier one, except that d_{ij} is defined differently. Instead of using a bit that must be the same as the committed one, every bidder now uses $d_{ij} = p_{ij} \wedge d_{i\overleftarrow{j}}$, where $d_{i\overleftarrow{j}}$ is the bit that the bidder used in the previous deciding bit position. We assume that *Stage II* starts from the jth position and iterates towards the lest significant bit position until $j = c$. Therefore, the steps of *Round 1* are exactly same as the *Stage I*. The only difference in the *Round 2* occurs while calculating the encrypted cryptogram b_{ij}. We present the stage in the following.

Round 1.

- Each bidder V_i selects two private keys $x_{ij} \leftarrow_R \mathbb{Z}_p$ and $r_{ij} \leftarrow_R \mathbb{Z}_p$.
- V_i computes the public keys $X_{ij} = g^{x_{ij}}$ and $R_{ij} = g^{r_{ij}}$
- Each V_i publishes $Pub_{ij} = (X_{ij}, R_{ij})$ and NIZKPs for $x_{ij} = \log_g X_{ij}$ and $r_{ij} = \log_g R_{ij}$.

Round 2.

- Each V_i computes $Y_{ij} = g^{y_{ij}} = \prod_{k=1}^{i-1} g^{x_{kj}} / \prod_{k=i+1}^{n} g^{x_{kj}}$
- V_i already holds $p_{ij} \in \{0, 1\}$, corresponding to his choice of p_i
- V_i posts encrypted ciphertext, known as *cryptogram*

$$b_{ij} = \begin{cases} Y_{ij}^{x_{ij}} = g^{x_{ij}y_{ij}} & \text{if } p_{ij} \wedge d_{i\overleftarrow{j}} = 0; \text{ (0-cryptogram)} \\ R_{ij}^{x_{ij}} = g^{x_{ij}r_{ij}} & \text{if } p_{ij} \wedge d_{i\overleftarrow{j}} = 1; \text{ (1-cryptogram)} \end{cases}$$

  on the bulletin board along with π_{ij}, a NIZKP of well-formedness of b_{ij}.

Tally for Stage II.

- Compute $\prod_{i=1}^{n} b_i$, the product of all the cryptograms.
- If $\prod_{i=1}^{n} b_i = 1$, the we conclude $T_j = 0$.
- If $\prod_{i=1}^{n} b_i \neq 1$, the we conclude $T_j = 1$.
- Hence, we get $T_{j+1}T_{j+2}\ldots T_c$ as final output, where j is the *Junction* Position.

After the second round, one can check all NIZKPs and compute T_j. The bidders follow the same procedure to iterate through the rest of bit positions. The logical-OR outputs T_j from each of the c bit positions constitutes the binary representation of the highest bid, i.e. the highest bid is $T_1||T_2||\ldots||T_c$ in binary format.

Once, the highest bid is computed, the winning bidder V_w can come forward and prove that (s)he is indeed the real winner either by opening her commitments $\{\varepsilon_{wj} : 1 \leq j \leq c\}$ or by revealing the randomness $x_{wk} = \log_g X_{wk}$ used in the last deciding bit position allowing everyone else to decipher the cryptogram submitted by him/her in the iteration corresponding to the last deciding bit position. It is easy to see that only the winner(s) would submit 1-cryptogram(s) in the iteration corresponding to the last deciding bit position. Based on x_{wk}, everyone is able to verify if there is only one winner of if there is a tie.

NIZKP for Stage II.

Since the calculations in *Round I* is exactly similar to the *Round I* of *Stage I*, the same ZKPs would work here as well. In *Round II*, the bidder posts a NIZKP of the well-formedness of the encrypted cryptogram. For proving well-formedness of b_{ij}, we need to prove $d_{ij} = p_{ij} \wedge d_{i\overleftarrow{j}}$. For that, we need to show the logical statement $(d_{ij} = 0 \wedge (p_{ij} \wedge d_{i\overleftarrow{j}}) = 0) \vee (d_{ij} = 1 \wedge (p_{ij} \wedge d_{i\overleftarrow{j}}) = 1)$, which is equivalent to prove the following statement

$$(d_{ij} = 1 \wedge p_{ij} = 1 \wedge d_{i\overleftarrow{j}} = 1) \vee (d_{ij} = 0 \wedge p_{ij} = 0 \wedge d_{i\overleftarrow{j}} = 1) \vee (d_{ij} = 0 \wedge d_{i\overleftarrow{j}} = 0)$$

Basically, in iteration j, we need to prove this one-out-of-three logical statement.

$$\sigma \equiv \Bigg((b_{ij} = g^{x_{ij}r_{ij}} \wedge X_{ij} = g^{x_{ij}} \wedge R_{ij} = g^{r_{ij}}) \bigwedge (b_{ij'} = g^{x_{ij'}r_{ij'}} \wedge X_{ij'} = g^{x_{ij'}} \wedge R_{ij'} = g^{r_{ij'}})$$

$$\bigwedge (c_{ij} = g^{\alpha_{ij}\beta_{ij}} g \wedge A_{ij} = g^{\alpha_{ij}} \wedge B_{ij} = g^{\beta_{ij}}) \Bigg) \bigvee \Bigg((b_{ij} = g^{x_{ij}y_{ij}} \wedge X_{ij} = g^{x_{ij}} \wedge Y_{ij} = g^{y_{ij}})$$

$$\bigwedge (b_{ij'} = g^{x_{ij'}r_{ij'}} \wedge X_{ij'} = g^{x_{ij'}} \wedge R_{ij'} = g^{r_{ij'}}) \bigwedge (c_{ij} = g^{\alpha_{ij}\beta_{ij}} \wedge A_{ij} = g^{\alpha_{ij}} \wedge B_{ij} = g^{\beta_{ij}}) \Bigg)$$

$$\bigvee \Bigg((b_{ij} = g^{x_{ij}y_{ij}} \wedge X_{ij} = g^{x_{ij}} \wedge Y_{ij} = g^{y_{ij}}) \bigwedge (b_{ij'} = g^{x_{ij'}y_{ij'}} \wedge X_{ij'} = g^{x_{ij'}} \wedge Y_{ij'} = g^{y_{ij'}}) \Bigg)$$

We can write the statement in following way for simplicity.

$$\sigma \equiv \Bigg((B_i = g^{x_i r_i} \wedge X_i = g^{x_i} \wedge R_i = g^{r_i}) \bigwedge (B_j = g^{x_j r_j} \wedge X_j = g^{x_j} \wedge R_j = g^{r_j})$$

$$\bigwedge (C_i = g^{\alpha_i \beta_i} g \wedge A = g^{\alpha_i} \wedge B = g^{\beta_i}) \Bigg) \bigvee \Bigg((B_i = g^{x_i y_i} \wedge X_i = g^{x_i} \wedge Y_i = g^{y_i})$$

$$\bigwedge (B_j = g^{x_j r_j} \wedge X_j = g^{x_j} \wedge R_j = g^{r_j}) \bigwedge (C_i = g^{\alpha_i \beta_i} \wedge A = g^{\alpha_i} \wedge B = g^{\beta_i}) \Bigg)$$

$$\bigvee \Bigg((B_i = g^{x_i y_i} \wedge X_i = g^{x_i} \wedge Y_i = g^{y_i}) \bigwedge (B_j = g^{x_j y_j} \wedge X_j = g^{x_j} \wedge y_j = g^{y_j}) \Bigg)$$

The above statement is a one-out-of-3 logical statement. The NIZK proof for the above statement is constructed such that each bidder will be able to show the well-formedness of her cryptogram without revealing whether the cryptogram is an encryption of 0 or 1.

Since this is a one-out-of-3 NIZKP, basically we construct three cases, where in each case we assume one of these statements is correct. We here describe all these three cases and how the commitments, responses and challenges are formed by the Prover (*i.e.* the Bidder) for the NIZKP. After that, we shall describe the verifying equations. In *Case 1*, we assume that $(B_i = g^{x_i r_i} \wedge X_i = g^{x_i} \wedge R_i = g^{r_i}) \bigwedge (B_j = g^{x_j r_j} \wedge X_j = g^{x_j} \wedge R_j = g^{r_j}) \bigwedge (C_i = g^{\alpha_i \beta_i} g \wedge A = g^{\alpha_i} \wedge B = g^{\beta_i})$ is true. Similarly, the second and third cases are assuming the other statements are true, respectively. Overall it requires at most 28 exponentiations for computing the above zero knowledge proof. Also the space complexity of the proof is 27. The verifier needs to perform 32 exponentiations for verifying all the arguments in the given 16 equations.

`Case I`

Chooses $r_{11}, r_{12}, r_{13} \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{11} = g^{r_{11}}, \varepsilon_{12} = g^{r_{12}}, \varepsilon_{13} = g^{r_{13}}$

$\varepsilon'_{11} = R_i^{r_{11}}, \varepsilon'_{12} = R_j^{r_{12}}, \varepsilon'_{13} = B^{r_{13}}$

Chooses $\rho_{21}, \rho_{22}, \rho_{23} \leftarrow_R \mathbb{Z}_p$

Chooses $ch_2 \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{21} = g^{\rho_{21}} X_i^{ch_2}, \varepsilon_{22} = g^{\rho_{22}} X_j^{ch_2}, \varepsilon_{23} = g^{\rho_{23}} A^{ch_2}$

$\varepsilon'_{21} = Y_i^{\rho_{21}} B_i^{ch_2}, \varepsilon'_{22} = R_j^{\rho_{22}} B_j^{ch_2}, \varepsilon'_{23} = B^{\rho_{23}} C_i^{ch_2}$

Chooses $\rho_{31}, \rho_{32}, \rho_{33} \leftarrow_R \mathbb{Z}_p$

Chooses $ch_3 \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{31} = g^{\rho_{31}} X_i^{ch_3}, \varepsilon_{32} = g^{\rho_{32}} X_j^{ch_3}$

$\varepsilon'_{31} = Y_i^{\rho_{31}} B_i^{ch_3}, \varepsilon'_{32} = Y_j^{\rho_{32}} B_j^{ch_3}$

$ch \leftarrow$ Grand Challenge of NIZKP

$ch_1 = ch - ch_2 - ch_3$

$\rho_{11} = r_{11} - x_i.ch_1$

$\rho_{12} = r_{12} - x_j.ch_1$

$\rho_{13} = r_{13} - \alpha_i.ch_1$

`Case II`

Chooses $r_{21}, r_{22}, r_{23} \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{21} = g^{r_{21}}, \varepsilon_{22} = g^{r_{22}}, \varepsilon_{23} = g^{r_{23}}$

$\varepsilon'_{21} = Y_i^{r_{21}}, \varepsilon'_{22} = R_j^{r_{22}}, \varepsilon'_{23} = B^{r_{23}}$

Chooses $\rho_{11}, \rho_{12}, \rho_{13} \leftarrow_R \mathbb{Z}_p$

Chooses $ch_1 \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{11} = g^{\rho_{11}} X_i^{ch_1}, \varepsilon_{12} = g^{\rho_{12}} X_j^{ch_1}, \varepsilon_{13} = g^{\rho_{13}} A^{ch_1}$

$\varepsilon'_{11} = R_i^{\rho_{11}} B_i^{ch_1}, \varepsilon'_{12} = R_j^{\rho_{12}} B_j^{ch_1}, \varepsilon'_{13} = B^{\rho_{13}} (C_i/g)^{ch_1}$

Chooses $\rho_{31}, \rho_{32}, \rho_{33} \leftarrow_R \mathbb{Z}_p$

Chooses $ch_3 \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{31} = g^{\rho_{31}} X_i^{ch_3}, \varepsilon_{32} = g^{\rho_{32}} X_j^{ch_3}$

$\varepsilon'_{31} = Y_i^{\rho_{31}} B_i^{ch_3}, \varepsilon'_{32} = Y_j^{\rho_{32}} B_j^{ch_3}$

$ch \leftarrow$ Grand Challenge of NIZKP

$ch_2 = ch - ch_1 - ch_3$

$\rho_{21} = r_{21} - x_i.ch_2$

$\rho_{22} = r_{22} - x_j.ch_2$

$\rho_{23} = r_{23} - \alpha_i.ch_2$

`Case III`

Chooses $r_{31}, r_{32} \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{31} = g^{r_{31}}, \varepsilon_{32} = g^{r_{32}}$

$\varepsilon'_{31} = Y_i^{r_{31}}, \varepsilon'_{32} = Y_j^{r_{32}}$

Chooses $\rho_{11}, \rho_{12}, \rho_{13} \leftarrow_R \mathbb{Z}_p$

Chooses $ch_1 \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{11} = g^{\rho_{11}} X_i^{ch_1}, \varepsilon_{12} = g^{\rho_{12}} X_j^{ch_1}, \varepsilon_{13} = g^{\rho_{13}} A^{ch_1}$

$\varepsilon'_{11} = R_i^{\rho_{11}} B_i^{ch_1}, \varepsilon'_{12} = R_j^{\rho_{12}} B_j^{ch_1}, \varepsilon'_{13} = B^{\rho_{13}} (C_i/g)^{ch_1}$

Chooses $\rho_{21}, \rho_{22}, \rho_{23} \leftarrow_R \mathbb{Z}_p$

Chooses $ch_2 \leftarrow_R \mathbb{Z}_p$

$\varepsilon_{21} = g^{\rho_{21}} X_i^{ch_2}, \varepsilon_{22} = g^{\rho_{22}} X_j^{ch_2}, \varepsilon_{23} = g^{\rho_{23}} A^{ch_2}$

$\varepsilon'_{21} = Y_i^{\rho_{21}} B_i^{ch_2}, \varepsilon'_{22} = R_j^{\rho_{22}} B_j^{ch_2}, \varepsilon'_{23} = B^{\rho_{23}} C_i^{ch_2}$

$ch \leftarrow$ Grand Challenge of NIZKP

$ch_3 = ch - ch_1 - ch_2$

$\rho_{31} = r_{31} - x_i.ch_3$

$\rho_{32} = r_{32} - x_j.ch_3$

`Verifying Equations`

$\varepsilon_{11} \stackrel{?}{=} g^{\rho_{11}} \,.\, X_i^{ch_1}$

$\varepsilon_{12} \stackrel{?}{=} g^{\rho_{12}} \,.\, X_j^{ch_1}$

$\varepsilon_{13} \stackrel{?}{=} g^{\rho_{13}} \,.\, A^{ch_1}$

$\varepsilon'_{11} \stackrel{?}{=} R_i^{\rho_{11}} \,.\, B_i^{ch_1}$

$\varepsilon'_{12} \stackrel{?}{=} R_j^{\rho_{12}} \,.\, B_j^{ch_1}$

$\varepsilon'_{13} \stackrel{?}{=} B^{\rho_{13}} \,.\, (C_i/g)^{ch_1}$

$\varepsilon_{21} \stackrel{?}{=} g^{\rho_{21}} \,.\, X_i^{ch_2}$

$\varepsilon_{22} \stackrel{?}{=} g^{\rho_{22}} \,.\, X_j^{ch_2}$

$\varepsilon_{23} \stackrel{?}{=} g^{\rho_{23}} \,.\, A^{ch_2}$

$\varepsilon'_{21} \stackrel{?}{=} Y_i^{\rho_{21}} \,.\, B_i^{ch_2}$

$\varepsilon'_{22} \stackrel{?}{=} R_j^{\rho_{22}} \,.\, B_j^{ch_2}$

$\varepsilon'_{23} \stackrel{?}{=} B^{\rho_{23}} \,.\, C_i^{ch_2}$

$\varepsilon_{31} \stackrel{?}{=} g^{\rho_{31}} \,.\, X_i^{ch_3}$

$\varepsilon_{32} \stackrel{?}{=} g^{\rho_{32}} \,.\, X_j^{ch_3}$

$\varepsilon'_{31} \stackrel{?}{=} Y_i^{\rho_{31}} \,.\, B_i^{ch_3}$

$\varepsilon'_{32} \stackrel{?}{=} Y_j^{\rho_{32}} \,.\, B_j^{ch_3}$

5.4 Extension to Vickrey auction

We discussed the SEAL Protocol thoroughly in this section, which is developed for the first-price sealed-bid auction. A straightforward way to extend it to support second-price sealed-bid (i.e., Vickrey auction) works as follows. The protocol is first run to identify the highest bid and the winner, and then run the second time with the winner excluded to compute the second-highest bid. The bidder who commits the second-highest bid remains anonymous. The winner pays the second-highest bid in the end. However, in this protocol the highest bid will be revealed, which is not strictly necessary, and may cause some privacy concerns. Yet we can say that this can be extended to second-price sealed bid auction protocol due to it's privacy policy. We first explain the property and state the formal proof of that and then we discuss the way to extend the protocol.

An auction protocol is said to satisfy *inclusive privacy*, if each bidder V_i learns nothing more than their own input and the output of the function $f_{max}(p1, \ldots, pn)$. Consider C to be a set of colluding bidders and H be the rest of the bidders, *i.e.* $C \cup H$ is the full set of bidders. Let θ be the size of H and $h_i \in H$ for $i \in \{1, \ldots, \theta\}$. We assume C to be non-empty set. We say an auction protocol satisfies *exclusive privacy* if besides the maximum of all inputs and their own inputs, C learns nothing more than $f_{max}(p_{h_1}, \ldots, p_{h_\theta})$.

Theorem. SEAL scheme satisfies exclusive privacy.

Proof of Theorem. To prove this theorem, we first prove the following lemma. As stated above, we assume $C \neq \phi$ to be the set of colluding bidders and H be the set of honest bidders. Also, $|C \cup H| = n$ and $|H| = \theta$.

Lemma. Let d_{h_ij} be the bit corresponding to the cryptogram submitted in iteration j by V_{h_i} for $hi \in H, 1 \leq i \leq \theta$ and d_{c_ij} be the bit corresponding to the cryptogram submitted in iteration j by V_{c_i} for $c_i \in C, 1 \leq i \leq n - \theta$. Then, the set of colluding bidders can learn nothing more than $\bigvee_{i=1}^{\theta} d_{h_ij}$.

Proof of Lemma. Suppose in some iteration j, we have $K_j = \bigvee_{i=1}^{\theta} d_{h_ij} = 0$, which means anyone can learn that $d_{h_ij} = 0$ for all $h_i \in H$. Hence, we have to show that when $K_j = \bigvee_{i=1}^{\theta} d_{h_ij} = 1$, the colluding bidders will not learn any other information. In order for proving this fact it is sufficient to show the followings.

- The colluding bidders will not be able to distinguish between the two cases where $\bigvee_{i=1}^{\theta} d_{h_ij} = 1$, but the number of bidders who submitted 1-cryptogram is different.
- If two honest bidders who submitted different bits exchange their inputs, then this cannot be detected by the adversary.

We choose two scenarios in which a particular honest bidder submits different cryptograms *i.e.* in one scenario the bidder submits a 0-cryptogram and in the other one she submits a 1-cryptogram. We show that if the value of K_j is 1 in both the scenarios, then the two scenarios will be indistinguishable to the adversary (colluding bidders). Once we prove these results, they could be easily extended to show that the statement of the above lemma holds. Let us assume that the public keys used by the colluding bidders are $(X_{c_i}, R_{c_i}) = (g^{x_{c_i}}, g^{r_{c_i}})$, for each $c_i \in C$. Similarly, the public keys of the honest bidders will be (X_{h_i}, R_{h_i}), for each

$h_i \in H$. The cryptograms of the colluding bidders will be $g^{x_{c_i} z_{c_i}}$, where $z_{c_i} = y_{c_i}$ or $z_{c_i} = y_{c_i}$, depending upon the choice of bid. Let, $\phi = \{x_{c_i} \mid 1 \leq i \leq n - \theta\}$. Now, let us assume that one honest bidder V_{h_w} has submitted a 1-cryptogram in the form $g^{x_{h_w} r_{h_w}}$. As such we need to show that the colluding bidders will not be able to find whether or not there is another bidder $V_{h_t}, h_t \in H$, who also submitted a 1-cryptogram. If V_{h_t} submitted a 1-cryptogram, then her cryptogram will be $b_{h_t} = g^{x_{h_w} r_{h_w}}$, and if she submitted a 0-cryptogram, her cryptogram should look like $b'_{h_t} = g^{x_{h_t} r_{h_t}}$.

Since we assumed DDH assumption in this group, we conclude that [13] no adversary can distinguish between $A = (\phi, b_1, b_2, \ldots, b_{h_w}, \ldots, b_{h_t}, \ldots, b_n)$ and $B = (\phi, b_1, b_2, \ldots, b_{h_w}, \ldots, b'_{h_t}, \ldots, b_n)$. Hence, the colluding bidders will not be able to distinguish between two cases where the value of $K_j = \bigvee_{i=1}^{\theta} d_{h_i j}$ is 1, but the number of bidders V_{h_i}, $h_i \in H$ who submitted 1-cryptogram in iteration j is different. Let us assume $b_{h_e} = g^{x_{h_e} r_{h_e}}$ for $e \in \{w, t\}$. Now observe that

$$B = (\phi, b_1, b_2, \ldots, b_{h_w}, \ldots, b'_{h_t}, \ldots, b_n)$$

$$\overset{c}{\approx} (\phi, b_1, b_2, \ldots, b_{h_w}, \ldots, b_{h_t}, \ldots, b_n)$$

$$\overset{c}{\approx} (\phi, b_1, b_2, \ldots, b'_{h_w}, \ldots, b_{h_t}, \ldots, b_n)$$

Hence, the colluding adversary will not be able to distinguish between two cases where a pair of honest bidders exchange the value of their submitted bits whose values are complement to each other. This way anyone can prove that as long as at least one bidder submits a 1-cryptogram in any iteration, the colluding bidders will not be able to distinguish between the set of all possible cryptograms corresponding to all possible values of the bits submitted by honest bidders. What the colluding bidders learn is the logical-OR of all bits submitted by all honest bidders which is given by $K_j = \bigvee_{i=1}^{\theta} d_{h_i j}$. [*Proves the lemma.*]

Now note that, $d_{h_i j}$ is the bit corresponding to the cryptogram submitted in iteration j by V_{h_i}. This is equal to the actual bid value $b_{h_i j}$ only if the bidder V_{h_i} remains in the race (she will have to submit 0 if she has lost in the race as enforced by the ZKP in the equation $(d_{ij} = p_{ij} \wedge d_{i\overleftarrow{j}} = 1) \vee (d_{ij} = 0 \wedge d_{i\overleftarrow{j}} = 0)$). Assume the winner is decided at the β-th iteration, $1 \leq \beta \leq c$. If the colluding set do not contain the winner, the bit value K_j that they learn is the same as the j-th most significant bit in the highest bid. In other words, they learn nothing more than the highest bid of all bidders. However, if the colluding set contain the winner, they can learn $K_1 \mid\mid K_2 \mid\mid \ldots \mid\mid K_\beta$, which are the β most significant bits of the maximum bid of the non-colluding set H. The colluding set will learn the maximum bid of the non-colluding set in the worse case when $\beta = c$ (namely, the winner is only decided in the last bit iteration). Hence, the theorem is established. □

We now describe a more efficient and privacy-preserving method to support the Vickrey auction. In this method, the bit iterations will only need to be run once and the highest bid remains secret. As we will show in the proof of the earlier theorem, at each j-th bit iteration, every bidder V_k learns nothing more than $\bigvee_{i \in \{1,2,\ldots,n\} \setminus k} d_{ij}$. Therefore, at each bit iteration, the bidder who remains a winner can learn if she is the only winner in the race. Thus, if the bidder finds that she has submitted the sole one bit in that bit iteration and thus has become a confirmed winner, she declares herself as the winner and steps aside to let other bidders continue. Those losing bidders reset the output of that winning iteration to be 0 and make it a non deciding iteration. Losing bidders then continue executing the rest of the steps as specified in the main protocol. This would reveal the next highest bid, while hiding the $c - j$ least significant bits of the highest bid.

6 Implementation of SEAL Protocol

In this section, we thoroughly discuss about the implementation of the discussed SEAL protocol over a Test Ethereum Network. Three HTML5 / JavaScript pages are to be developed to provide a browser interface for all bidder interactions. The web browser interacts with an Ethereum daemon running in the background. We use the Remix Ethereum Platform to run the whole protocol. We use Solidity language to design the tally and the Zero-knowledge Proofs of the scheme. The protocol is executed in five stages, and requires bidder interaction in two rounds. We give an overview of the implementation in the following subsection. The code is supposed to achieve the following properties.

- All communication is public - no secret channels between bidders are required.
- The system is self-tallying - no tallying authorities are required.
- The bidder's privacy protection is maximum - only a full collusion that involves all other bidders in the election can uncover the bidder's secret bid.
- The system is dispute-free - everybody can check whether all bidders act according to the protocol, hence ensuring the the result of the auction protocol is publicly verifiable.

6.1 Design Rationale

6.1.1 Structure of Implementation

The smart contracts for the tally and the ZKPs are written in Ethereum's Solidity language, as stated. In the final protocol, we wish to keep only two smart contracts, namely the **auction contract** and the **cryptography contract**. The auction contract implements the auction protocol, controls the auction process and verifies all the zero knowledge proofs we have in the SEAL e-auction protocol. The second contract *i.e.* the cryptography contract distributes the code for creating the zero knowledge proofs. This provides all bidders with the same cryptography code that can be used locally without interacting with the Ethereum network. Also, we need to finally provide three HTML5 pages for the users.

1. **Auction Administrator** *admin.html*

   This page administers the auction. This includes establishing the list of eligible bidders, setting the auction question, and activating a list of timers to ensure the election progresses in a timely manner. The latter includes notifying Ethereum to begin registration, to close registration and begin the auction, and to close bidding and compute the tally.

2. **Bidder** *bid.html*

   This page is designed for the bidder. The bidder can register for an auction, and once registered, she must cast her bid.

3. **Observer** *livefeed.html*

   Anyone, even any third party observer, can watch the auction's progress consisting of the administrator starting and closing each stage and bidders registering and casting bids. The running tally is not computable.

6.1.2 Auction stages

The five stages of the auction are described below in brief. The smart contracts has a designated owner that represents the auction administrator. This administrator is responsible for authenticating the bidders with their user controlled account and updating the list of eligible bidders. A list of timers is enforced by the smart contract to ensure that the auction progresses in a timely manner. The contract only allows eligible bidders to register for an auction, and registered bidders to cast their respective bids. Furthermore, the contract can require each bidder to deposit ether upon registration, and automatically refund the ether when their bid is accepted into the Block-chain.

1. **SETUP Stage.** The auction administrator authenticates each bidder with their user-controlled account and updates the contract to include a whitelist of accounts as eligible bidders. She defines a list of timers to ensure that the auction progresses in a timely manner. The time stamp we wish to keep in our implementation are the followings.

   - $t_{finishRegistration}$: All bidders must register their public keys (X_i, R_i) before this time stamp.
   - $t_{beginAuction}$: The administrator must notify Ethereum to begin the process by this time.
   - $t_{finishCommit}$: All bidders must commit to their bid before this time.
   - $t_{finishBid}$: All bidders must submit their bid before this time.
   - π : A minimum length of time in which the commitment and bidding stages must remain active to give bidders sufficient time to bid.

   The administrator also sets the registration deposit, the auction question/statement, and finally, the administrator notifies Ethereum to transition from the SETUP to the SIGNUP stage.

2. **SIGNUP Stage.** All eligible bidders can choose to register for the bid after reviewing the auction statement and other parameters set by the administrator. To register, the bidder computes their public key and ZKPs. Both the keys and proof are sent to Ethereum alongside deposit ether. Ethereum does not accept any registrations after the scheduled time. The administrator is responsible for notifying Ethereum to transition from the SIGNUP to COMMIT stage. All bidder's reconstructed keys are computed by Ethereum during the transition.

3. **COMMIT Stage.** All bidders publish the commitments formed by them using their bid to the Ethereum blockchain. The contract automatically transitions to the BIDDING stage once the final commitment is accepted into the Block-chain.

4. **BIDDING Stage.** All bidders publish their (encrypted) bid and corresponding ZKP. The deposit is refunded to the bidder when their bid is accepted by Ethereum. The administrator notifies Ethereum to compute the tally once the final bid is cast.

5. **TALLY Stage.** The administrator notifies Ethereum to compute the tally. Ethereum computes the product of all ballots (encrypted bids) and declared the result in the way defined in the protocol.

SEAL e-auction protocol requires all the registered bidders to cast their bid to enable the tally calculation. The deposit in our implementation provides a financial incentive for registered bidders to bid. This deposit is returned to the bidder if they follow through with the SEAL protocol and do not drop out. The list of timestamps defined by the administrator determines if the bidder's deposit is forfeited or refunded. There are three refund scenarios if a deadline is missed in the protocol.

- Registered bidders can claim their refund if the auction does not begin by $t_{beginAuction}$.
- Registered bidders who have committed can claim their refund if not all registered bidders commit to their bid by $t_{finishCommit}$.
- Registered bidders can claim their refund if not all registered bidders cast their bid by $t_{finishBid}$.

As described earlier, the SEAL protocol has used a modified version of Hao-Zielinski's Anonymous Veto Protocol [9] as a building block for the scheme. Previously, this AV-Net protocol was adapted for another scheme for e-voting, namely *Anonymous voting scheme by two-round public discussion* [10] by Feng Hao, Peter Ryan, and Piotr Zielinski in 2010. The implementation idea is almost in line with the implementation of the Open Vote Network (based on paper Anonymous voting by two-round public discussion). Patrick McCorry has implemented [15] this scheme over test ethereum network. We wish to adapt the code for the scheme in favour of our SEAL Protocol. The detailed mechanism is also studied and adapted for our protocol through their paper, namely *A Smart Contract for Boardroom Voting with Maximum Voter Privacy* [28].

6.1.3 Overview of the Code Execution

The final protocol is supposed to consists of the whole description given above. In this project, we have calculated the backends of the whole protocol *i.e.* we have designed all the ZKPs needed for this calculations as well as the tally for the Commit phase and Stage I and Stage II calculations of the Computation Phase has been implemented. Also we have implemented the JAVA file to generate the nonces needed and the output of the JAVA file is to be uploaded by the bidders while registering to the process.

Furthermore, we need to design the HTML5 front end to finally run the protocol as desired. As of now, to check the correctness of the protocols we designed, we have generated test values through the protocol itself and checked if the ZKPs are working properly or not. Similarly, we have generated a dummy tally as well to check if the whole SEAL code can be checked working properly or not using small number of bidders and small size input bids. We have kept the number of bidders to be three and each bidder can submit bid of 5 bits.

During the final execution, we need to run Geth in the background. Then the admin would create three Bidder account and unlock them through terminal command. Now, the admin needs to transact the minimum ether needed for bidding to each Bidder from Admin account. We need to create a Charity account, which may contain any ether. Then, the admin compile the sol file for eAuction. On successful compilation, deploy the contract using the Admin account. Fix gas limit higher as required, gap to be 1 and keep the charity account address as created. On successful deployment, the admin copies the Ethereum Address of the newly created contract and put that into the correct positions within the code of Admin.html, Bidder.html and Livefeed.html.

In similar way, we compile and deploy the other sol file for the ZKPs. After deployment, the admin copies the address and put that into Admin.html and Bidder.html. Now, the admin opens the Admin page and three Bidder pages in web browser. The Admin log into (default passphrase is empty string) the system and update the list of eligible bidders into the system. Note that, the Charity address won't be a Bidder. Next Admin sets the timestamps, states the Auction statement and begin the Registration. Each eligible Bidder now uploads the text file, generated through the JAVA program and logs in the system using their Ethereum Address and passphrase (assigned when creating account) and registers themselves, for which they have to use some fixed amount of ether. Admin then finishes the Registration phase and the bidders are allowed to cast their bids. They cast their encrypted bids and once all the bidders are done, one can see the output of the program. Livefeed.html contains all the information as well. Hence, any third party observer can also check this.

Since the HTML files are yet to be finished, we in this project would show how to assign values and verify the correctness of the backend functions of this protocol. Also we shall assign proper values and compute a dummy auction protocol to have a glimpse of the output delivered through the final protocol.

6.2 Elliptic Curve and its Usage in Cryptology

6.2.1 Limitation of Finite Field Arithmetic

Finite field arithmetic plays the most important role in cryptography, as every hardness assumption of cryptography (DDH, factorization) follows from properties of finite field. Each of these hardness assumption comes with multiplication of large numbers and/or exponentiation of the finite field elements with large numbers. Both of these type of operations comes with a lot of computational overhead, and also the numbers go beyond the scope of computer memory. In fact, multiplication of two 256-bit numbers can lead us to use memory of size 512-bit, which may go beyond the scope of computer-calculation.

During the execution of a cryptographic algorithm, we have to use these calculations for multiple times. In finite field, if we want to keep the security intact, we have to use large memory space again and again, making the system inefficient and even slow. Whereas, if we try to make the system efficient, we shall have to provide small data, so that we don't have to deal much overhead. But this memory-efficiency trade-off will have an impact on the security. In fact, since this time key-space size reduces, brute-force attacks will be easier. This scenario leads us to change our computation mechanism.

6.2.2 Elliptic Curve Cryptology

- **Elliptic Curve over Reals.**

  Let $a, b \in \mathbf{R}$ such that $4a^3 + 27b^2 \neq 0$. Then a non-singular elliptic curve is the set of points $(x, y) \in \mathbf{R} \times \mathbf{R}$ that satisfy

  $$y^2 = x^3 + ax + b$$

  together with a special point $\mathcal{O}$, called the point at infinity.

  We can realize elliptic curves in finite field also. Surprisingly, elliptic curves replicate finite field multiplications and exponentiation in a nice way which we can use to establish security and hardness results of cryptology.

- **Elliptic Curves in Finite Fields.**

  Let $p > 3$ be a prime. The elliptic curve $y^2 = x^3 + ax + b$ over $\mathbf{Z}_p$ is the set of solutions $(x, y) \in \mathbf{Z}_p \times \mathbf{Z}_p$ of the congruence relation

  $$y^2 \cong x^3 + ax + b \mod p$$

  together with the point $\mathcal{O}$ at infinity.

- **Operations over Elliptic Curve.**

  The addition operation over an elliptic curve $\mathcal{E}$ follows certain conditions.
  For any $(x, y) \in \mathcal{E}$,

  1. $\mathcal{O} + (x, y) = (x, y) + \mathcal{O} = (x, y)$
  2. $(x, y) + (x, -y) = \mathcal{O} = (x, -y) + (x, y)$
  3. For $(x_1, y_1) \neq (x_2, y_2) \in \mathcal{E}$, let $\lambda = \frac{y_2 - y_1}{x_2 - x_1} (\mod p)$. Then
     $(x_1, y_1) + (x_2, y_2) = (\lambda^2 - x_1 - x_2 \mod p, \lambda(2x_1 + x_2 - \lambda^2) - y_1 \mod p)$
  4. $2(x_1, y_1) = (\lambda^2 - 2x_1 \mod p, \lambda(3x_1 - \lambda^2) - y_1 \mod p)$ where $\lambda = \frac{3x_1^2 + a}{2y_1}$

- **Interpretation of Finite Field Arithmetic in Elliptic Curve.**

  Let G_1, G_2, G_3 be three abelian groups. A *pairing* is a function $e : G_1 \times G_2 \to G_3$ such that for any $P_1, Q_1 \in G_1, P_2, Q_2 \in G_2$,

  $$e(P_1 + Q_1, P_2) = e(P_1, P_2)e(Q_1, P_2) \text{ and}$$

  $$e(P_1, P_2 + Q_2) = e(P_1, P_2)e(P_1, Q_2)$$

  Clearly, $e(aP, bQ) = e(P, Q)^{ab}$.

If we simply replace the groups G_1, G_2 with $\mathcal{E}$ and G_3 with $F_{q^{k*}}$, then we can understand the relation between elliptic curve and finite field.

6.2.3 Elliptic curves over Solidity Language

Elliptic curve arithmetic uses less overhead than finite field calculations, yet provides the same level of security. There are various publicly available elliptic curves to be used in solidity. To use a particular elliptic curve $y^2 = x^3 + ax + b \mod p$ for a finite field of size n, we need to specify the values of a, b, p, n and the generator G of the elliptic curve group. The existing elliptic curves can handle $192, 224, 256, 384$ or 512-bit inputs. Also, the elliptic curves can be either verifiably random, or Koblitz-type. In Koblitz curves, we can implement fast elliptic curve operations. We have used the *secp256k1* as our underlying Koblitz-elliptic curve.

The secp256k1 parameters. The elliptic curve domain parameters over $\mathbf{F}_p$ associated with a Koblitz curve *secp256k1* for handling 256-bit data are specified by the tuple $T = (p, a, b, G, n)$ where the size of finite field $\mathbf{F}_p$ is defined by:

$$p = 0 \times \ FFFFFFFF \ FFFFFFFF \ FFFFFFFF \ FFFFFFFF \ FFFFFFFF$$

$$FFFFFFFF \ FFFFFFFE \ FFFFFC2F$$

The elliptic curve $\mathcal{E}$ is $y^2 = x^3 + ax + b \mod p$ where

$a = 0 \times 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000$

$b = 0 \times 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000000\ \ 00000007$

The X and Y coordinates of G are

$$Gx = 0 \times 79BE667E\ \ F9DCBBAC\ \ 55A06295\ \ CE870B07\ \ 029BFCDB\ \ 2DCE28D9$$
$$59F2815B\ \ 16F81798$$
$$Gy = 0 \times 483ADA77\ \ 26A3C465\ \ 5DA4FBFC\ \ 0E1108A8\ \ FD17B448\ \ A6855419$$
$$9C47D08F\ \ FB10D4B8$$

Moreover, $n = 0 \times FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFE\ \ BAAEDCE6$
$$AF48A03B\ \ BFD25E8C\ \ D0364141$$

6.2.4 ECCMath and Secp256k1 Libraries

1. The *ECCMath* library contains `expmod` and `invmod` functions, which finds $b^e \mod m$ in square and multiply algorithm and $a^{-1} \mod p$ respectively. Moreover, to handle point at infinity, the elliptic curve is realized in a 3-dimensional space and Jacobian coordinates. After calculating, the result is projected in $\mathcal{E}$ using the subroutine `toZ1`.

2. The library *Secp256k1* contains all the elliptic curve operations. It also checks if a given point P is on the curve or not using the subroutines `onCurve` and `isPubKey`.

```
function onCurve(uint[2] P) internal constant returns (bool) {
        uint p = pp;
        if (0 == P[0] || P[0] == p || 0 == P[1] || P[1] == p)
            return false;
        uint LHS = mulmod(P[1], P[1], p);
        uint RHS = addmod(mulmod(mulmod(P[0], P[0], p), P[0], p), 7, p
    );
        return LHS == RHS;
     }
```

3. The functions `_add`, `_addMixed`, `_addMixedM` all are used to add two points on the given elliptic curve. The only difference between these 3 functions is in the input type. `_add` can add two points which are in Jacobian(3-d) format, whereas `_addMixed` adds a point in the 2-d elliptic curve to another Jacobian point. `_addMixedM` is same as `_addMixed`, but it adds the points internally and returns nothing.

```
function _add(uint[3] memory P, uint[3] memory Q) internal constant
    returns (uint[3] memory R) {
        if(P[2] == 0)
            return Q;
        if(Q[2] == 0)
            return P;
        uint p = pp;
        uint[4] memory zs; // Pz^2, Pz^3, Qz^2, Qz^3
        zs[0] = mulmod(P[2], P[2], p);
        zs[1] = mulmod(P[2], zs[0], p);
        zs[2] = mulmod(Q[2], Q[2], p);
        zs[3] = mulmod(Q[2], zs[2], p);
```

```
        uint[4] memory us = [
            mulmod(P[0], zs[2], p),
            mulmod(P[1], zs[3], p),
            mulmod(Q[0], zs[0], p),
            mulmod(Q[1], zs[1], p)
        ]; // Pu, Ps, Qu, Qs
        if (us[0] == us[2]) {
            if (us[1] != us[3])
                return;
            else {
                return _double(P);
            }
        }
        uint h = addmod(us[2], p - us[0], p);
        uint r = addmod(us[3], p - us[1], p);
        uint h2 = mulmod(h, h, p);
        uint h3 = mulmod(h2, h, p);
        uint Rx = addmod(mulmod(r, r, p), p - h3, p);
        Rx = addmod(Rx, p - mulmod(2, mulmod(us[0], h2, p), p), p);
        R[0] = Rx;
        R[1] = mulmod(r, addmod(mulmod(us[0], h2, p), p - Rx, p), p);
        R[1] = addmod(R[1], p - mulmod(us[1], h3, p), p);
        R[2] = mulmod(h, mulmod(P[2], Q[2], p), p);
    }
```

4. The function `_mul` takes a point P on the elliptic curve, an integer n and obtains nP on the elliptic curve. The return type is a Jacobian point. The logic of the calculation is exactly same as calculating nP in elliptic curve.

```
function _mul(uint d, uint[2] memory P) internal constant returns (
    uint[3] memory Q) {
        uint p = pp;
        if (d == 0) // TODO
            return;
        uint dwPtr; // points to array of NAF coefficients.
        uint i;

        // wNAF
        assembly
        {
                let dm := 0
                dwPtr := mload(0x40)
                mstore(0x40, add(dwPtr, 512)) // Should lower this.
            loop:
                jumpi(loop_end, iszero(d))
                jumpi(even, iszero(and(d, 1)))
                dm := mod(d, 32)
                mstore8(add(dwPtr, i), dm) // Don"t store as signed -
    convert when reading.
                d := add(sub(d, dm), mul(gt(dm, 16), 32))
            even:
                d := div(d, 2)
                i := add(i, 1)
                jump(loop)
            loop_end:
        }
    . . .
    . . .
    }
```

6.3 Generating Private and Public Keys

As we described earlier, in the SIGNUP Stage of the protocol, the bidder needs to register with their Auction Key. The Auction Key is nothing but the public keys (X_{ij}, R_{ij}), where $X_{ij} = g^{x_{ij}}$ and $R_{ij} = g^{r_{ij}}$. Now these x_{ij} and r_{ij} are chosen randomly from $\mathbb{Z}_p$. Since we assume that any bidder can bid for amount, which can be represented in c bits in binary representation, we can fairly claim that our protocol runs for c iterations and hence, for each iterations, we need to generate both x_{ij} and r_{ij}. We, for now, assume c to be 5. Hence, we need five pair of random numbers chosen from $\mathbb{Z}_p$ to serve as Private keys and generate (X_{ij}, R_{ij}) pairs for each bit of the bid. We create a JAVA code to generate such and the JAVA file outputs "*auction.text*", which contains all this numbers. The bidder submits this file in order to register herself for the process.

```
    String xij = "";String Xij = "";
    for (int i = 0; i < 5; i++) {
        pair = g.generateKeyPair();
        BigInteger v = ((ECPrivateKey) pair.getPrivate()).getD();
        ECPoint vG = ((ECPublicKey) pair.getPublic()).getQ();
        BigInteger _vx = vG.getAffineXCoord().toBigInteger();
        BigInteger _vy = vG.getAffineYCoord().toBigInteger();
        if(i == 0){
          xij = v.toString();
          Xij = _vx.toString();
          Xij = Xij + "," + _vy;
        }
        else{
          xij = xij + "," + v;
          Xij = Xij + "," + _vx + "," + _vy;
        }
    }
```

The above code-snippet explains the way we generate each x_{ij} and $X_{ij} = g^{x_{ij}}$. Also, we need to generate nonces for the ZKPs and we do generate, append to the same text file and submit to the code in the SIGNUP Stage only. In cryptography, a *Nonce* is an arbitrary number that can be used just once in a cryptographic communication. For a process with $c = 5$ iterations, we generate a total of 60 numbers and store them in the mentioned *auction.txt* file. The generated elements are the followings.

- Random α_{ij}, β_{ij} for $1 \leq j \leq c = 5$
- Private Keys x_{ij}, r_{ij} for $1 \leq j \leq c = 5$
- Public Keys X_{ij}, R_{ij} for $1 \leq j \leq c = 5$
- Random nonce r for the single ZKP.
- Random nonces for ZKP for well-formedness of commitments.
- Random nonces for 1-out-of-2 ZKP in Stage I.
- Random nonces for 1-out-of-3 ZKP in Stage II.

Also, we need to import some advanced libraries in order to run our protocol. For that, alongside the JAVA file, we have added another *auctioncodes.jar* file, which handles the libraries. For the final execution of the JAVA file, we can run the following command in order to use the JAR file to fetch the libraries in the JAVA code we mentioned.

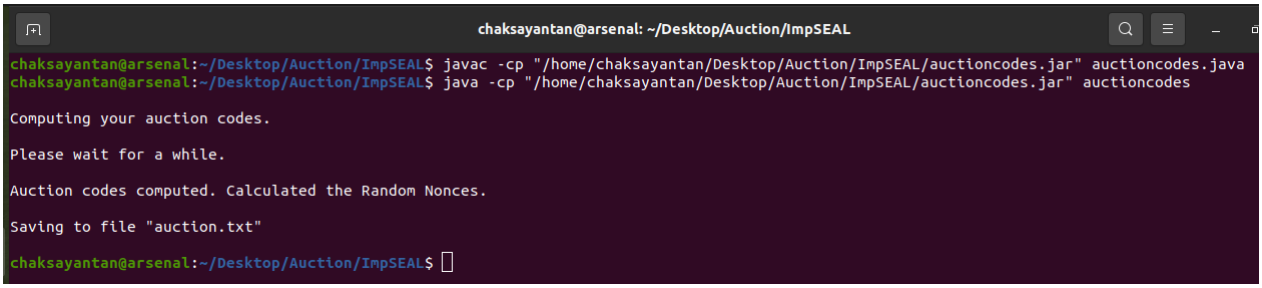

Figure 4: Generating Text file containing Random Nonces

6.4 Implementing Tally

6.4.1 Tally for Single Bit Case

We discussed in Section 5 that the SEAL protocol has two phases - Commit Phase and Computation Phase. In the commit phase, the bidder computes c many individual commitments. For each of the c many iteration of the protocol, the code basically generates $\varepsilon_{ij} = \langle g^{\alpha_{ij}\beta_{ij}} g^{p_{ij}}, g^{\alpha_{ij}}, g^{\beta_{ij}} \rangle$. The commitment function in the protocol does this (Fig 5). Since we are yet to finish designing the HTML5 front end, we cannot pass the values of $\alpha_{ij}, \beta_{ij}, p_{ij}$ and hence, for checking purpose we currently provide the internal inputs externally.

TallyStageOne, as the name suggests, does the calculations for the Stage I. Formally speaking, given one bit bids from different (three) bidders, it calculates the Logical-OR of the bids. The random nonces, as the commitment function, are to be given externally now but later on this shall be done internally once the front end is implemented. *TallyStageTwo* does the calculations for the Stage II phase of the protocol. *pi* denotes the bids from three different bidders and *di* here denoted the bid that the bidder used in the previous deciding bit position. As shown in the example $pi \ \wedge \ di = 0$ is to be treated as 0-cryptogram and $pi \ \wedge \ di = 1$ is to be treated as 1-cryptogram. Since, $pi \ \wedge \ di = 0$ for all three bits in the given example in the Fig. 5, clearly the output $T_j = 0$ is a correct output.

Note that, the reader is advised not to misunderstood the given input *pi* as the bid submitted by the i-th bidder. In the protocol we explained in the section 5.3., p_i stands for the bid of c-bits for the i-th candidate. Here, *pi* is the array of single-bit bids from three different bidders. With the notation we used earlier, we can think of *pi* here as $pi = [p_{0j}, p_{1j}, p_{2j}]$ where the logical-OR of these three bits will generate T_j. Surely, all-zero inputs will generate 0 output and any other input will yield 1 output. Similarly, in the right hand picture, the output is 1 if there exists at least one k such that both $pi[k] = 1$ and $di[k] = 1$.

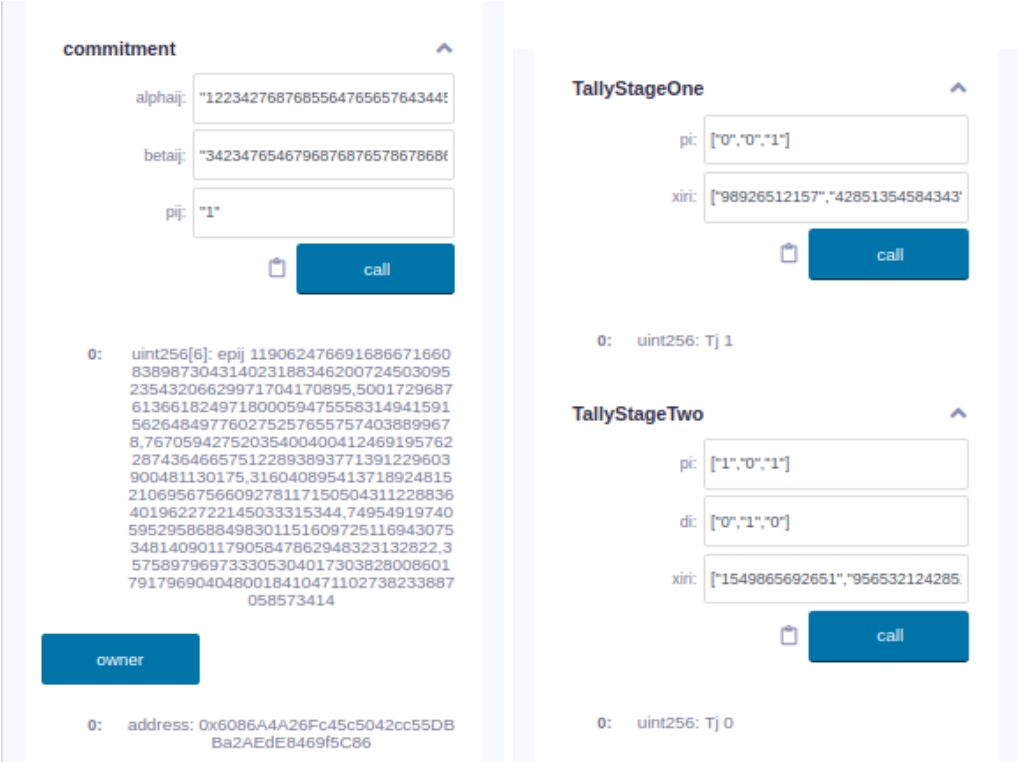

Figure 5: Tally for Commit Phase, Stage I and Stage II

6.4.2 Extending for Multiple Bits

In the section 6.1.3, we mentioned that for each iteration of the SEAL protocol, we basically do the tally separately. Hence, the functions described in the earlier sections shall be used in order to execute the whole protocol. yet we have implemented the computation using the scheme and checked if the protocol designed by us are working for proper inputs. As stated earlier, there are three bidders and each bidder is allowed to submit bids of five bits. The fourth input is nothing but x_{ij}, r_{ij} nonces, which shall be passed through the Java output file in the final protocol. The output of the protocol should be the highest bid value submitted in the auction, which we fairly claim is satisfying in the given three instances.

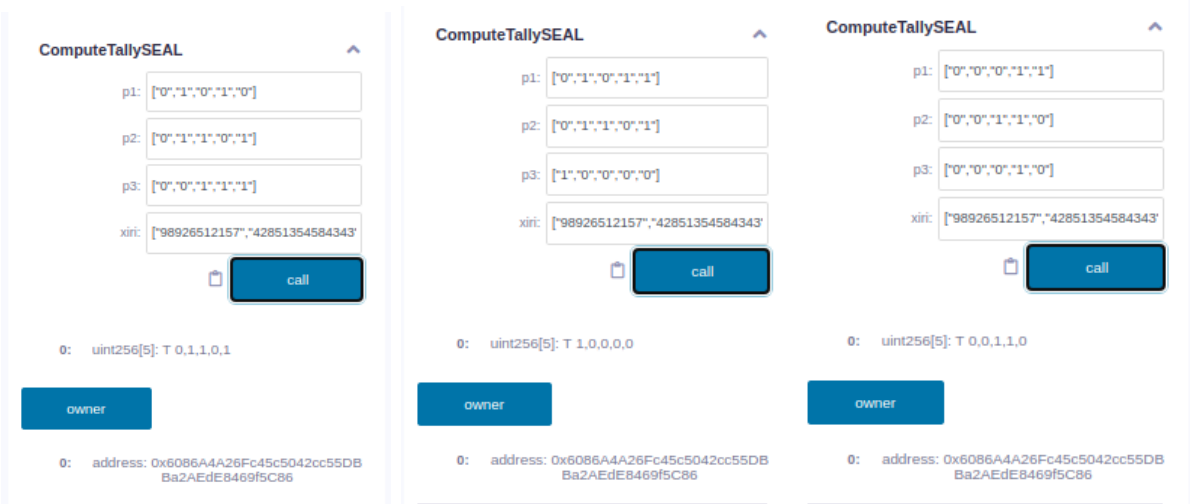

Figure 6: SEAL Protocol : Complete Tally Results

6.5 Implementing Zero-Knowledge Proofs

6.5.1 ZKP for Well-formedness of Public Keys

Since the SEAL protocol is getting executed without an auctioneer to actually judge the integrity of the protocol, we are depending upon the non-Interactive Zero-Knowledge proofs for the well-formedness of the Public Keys and others. We have seen in the protocol that Schnorr's ZKP has been used for multiple times for proving well-formedness of Public keys X_{ij} and R_{ij} and also to prove knowledge of α_{ij}, β_{ij} while publishing commitments. In this section, we discuss the implementation of Schnorr's ZKP for that.

The problem statement is to prove the knowledge of x while posting g^x in the public bulletin. In order to do that, we have created a sample example to check the correctness. The *createSample* function takes x as input and outputs g^x. Now recall that we have passed a random nonce r in the text file in SIGNUP phase, which shall work as an input of *createZKP* function along with x and g^x to provide us $\langle \varepsilon, \rho, ch \rangle$ as output. Now the *verifyZKP* function checks the verifying equation and outputs if the the ZKP is accepted or not.

Clearly the output of one function is used as input of other function here. We have kept the *public view* property on in these functions for checking purposes. If that is switched off, no observer can see the calculations.

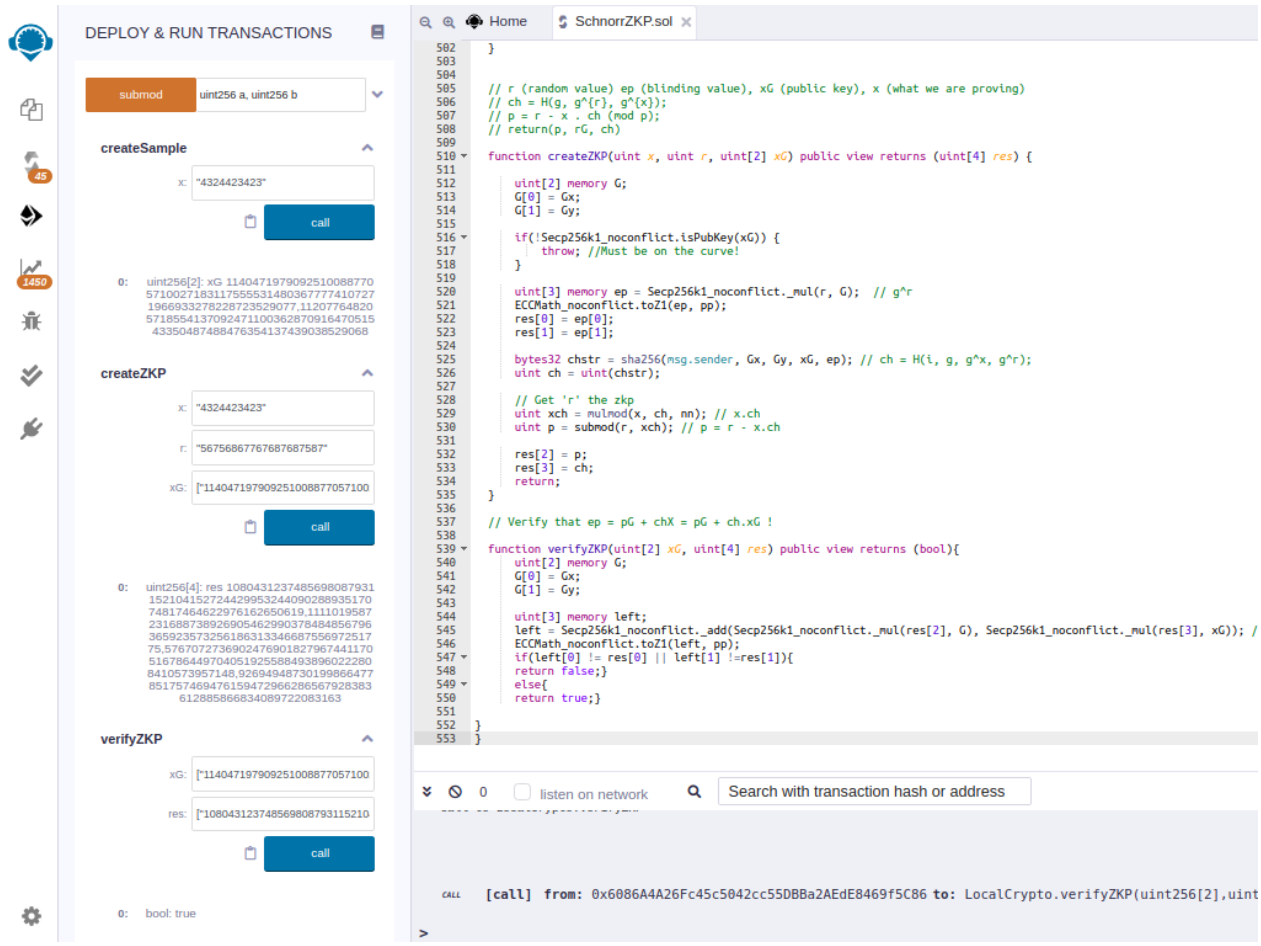

Figure 7: ZKP for Well-formedness of Public Keys

6.5.2 ZKP for Well-formedness of Commitments

This example is a classic example of one-out-of-two Zero-Knowledge Proof. Here, the bidder need to show the following statement holds in order to prove the well-formedness of the commitments.

$$\sigma \equiv (\phi = g^{\alpha\beta} \wedge A = g^{\alpha} \wedge B = g^{\beta}) \vee (\phi = g^{\alpha\beta} g \wedge A = g^{\alpha} \wedge B = g^{\beta})$$

Note that, depending upon $v = 0$ or $v = 1$, we have $g^{\alpha\beta} g^{v} = g^{\alpha\beta}$ or $g^{\alpha\beta} = g^{\alpha\beta} g$, respectively. Hence, only one of the statement can be true. WLOG we assume that the first statement is correct *i.e.* $\phi = g^{\alpha\beta} \wedge A = g^{\alpha} \wedge B = g^{\beta}$. So, the prover needs to provide a real proof for this statement and a simulated proof for the other statement $\phi = g^{\alpha\beta} g \wedge A = g^{\alpha} \wedge B = g^{\beta}$. Similarly, if the second statement holds, then the bidder needs to provide a real proof for the second statement and a simulated proof for the first.

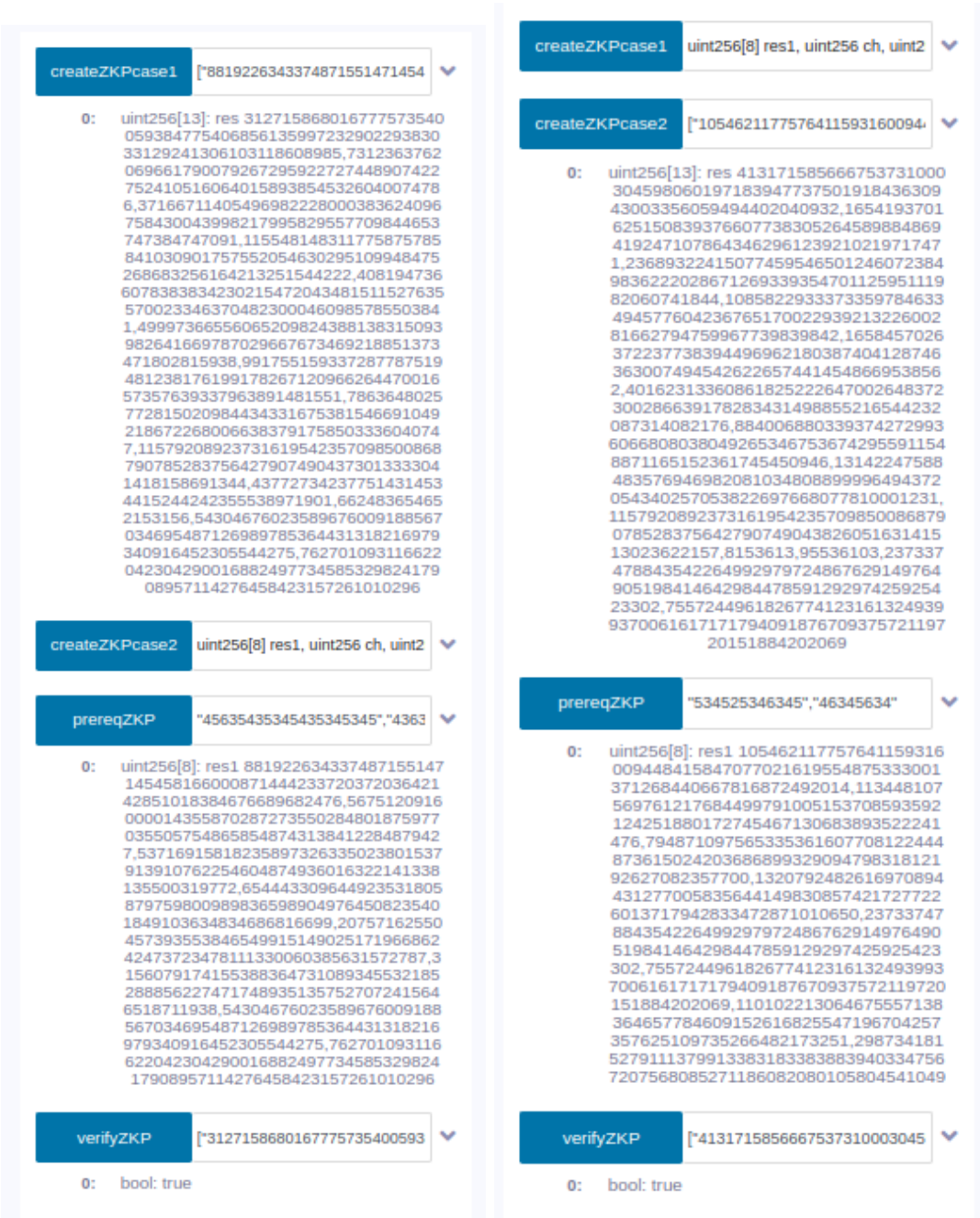

Figure 8: ZKP for Commitment Phase

The figure in the earlier page clearly showcases the two different cases. As before, we have feed the random nonces externally as input. In the picture on the left side, we feed the data generated in *prereqZKP* function and the random nonces in *createZKPcase1* assuming $v = 0$ and $g^{\alpha\beta}g^{v} = g^{\alpha\beta}$ and feed it's output in the *verifyZKP* function. Similar things have done in the right side picture assuming $v = 1$. In both cases, we check the same set of equations in the *verifyZKP* function, which are elaborated in the section 5.2.

6.5.3 ZKP for Computation Phase : Stage I-II

The final phase of the SEAL protocol is where we compute the highest bid from the bidders. We performed two different ZKPs for two different cases in the final phase of the protocol to prove the well-formedness of the cryptogram in Round II calculations. In order to do that, we basically check if $d_{ij} = p_{ij}$ holds or not in Stage I. Hence, the bidder need to prove that $(d_{ij} = 0 \wedge p_{ij} = 0) \vee (d_{ij} = 1 \wedge p_{ij} = 1)$. This is another example of 1-out-of-2 ZKP and the implementation is similar as above. In Fig. 9, we have shown the correctness for Case 1 only. Similarly, as shown in the ZKP for well-formedness of commitments, we can perform for Case 2 with not handling the *createZKPCase1* function. The following picture elaborates the idea.

Figure 9: ZKP for Stage I

The next ZKP, which is used in Stage II to prove $d_{ij} = p_{ij} \wedge d_{i\overleftarrow{j}}$ is an example of one-out-of-three ZKP, where the bidder need to provide the real proof of one case based on some assumption and the simulated proofs for the other two cases. We have provided an example where we show the real proof assuming case 3 is the real scenario. Here, the bidder posts a NIZKP of the well-formedness of the encrypted cryptogram. For proving well-formedness of b_{ij}, the bidder needs to prove $d_{ij} = p_{ij} \wedge d_{i\overleftarrow{j}}$. For that, she need to show the logical statement $(d_{ij} = 0 \wedge (p_{ij} \wedge d_{i\overleftarrow{j}}) = 0) \vee (d_{ij} = 1 \wedge (p_{ij} \wedge d_{i\overleftarrow{j}}) = 1)$, which is equivalent to prove the following statement $(d_{ij} = 1 \wedge p_{ij} = 1 \wedge d_{i\overleftarrow{j}} = 1) \vee (d_{ij} = 0 \wedge p_{ij} = 0 \wedge d_{i\overleftarrow{j}} = 1) \vee (d_{ij} = 0 \wedge d_{i\overleftarrow{j}} = 0)$.

Figure 10: ZKP for Stage II

6.6 Further Work Direction

As discussed in the section 6.1.3, we need to implement the HTML5 front end to finally achieve the goal we want to secure. For our experiment, where we are assuming three bidders with 5-bit bid value for each bidder, there shall be one admin.html to administrate the whole scheme and three bidder html files (one file which shall be used thrice) for registration and bidding purpose. Once these are done, we can have the whole protocol running in decentralized settings without any supervision of any trusted third party organisation, which we desired.

The whole back-end calculations are kept in separate files to easily understand the code and the verify them in test ethereum network. We believe that the screenshots used in this section would also help the reader to execute the programs easily. We have submitted the codes along with this report as attachments. We keep the codes in the following Google Drive link for easy access to public platforms.

`https://drive.google.com/drive/folders/1QNJ7digwAc83bG8tM5dyykS15UEnOpUX`

7 Final Notes

Auction, more generally, sealed-bid auction has been an integral part of trading goods for ages. It has been developed many fundamental aspects of the computer science subject as well. The privacy aspects of the problem and communication and computational complexity has raised interest among the computer scientists pretty well in the last few decades. The scope and reach of these auctions have been propelled by the Internet to a level beyond what the initial purveyors had anticipated. This is mainly because e-auction break down and remove the physical limitations of traditional auctions such as geography, presence, time, space, and a small target audience.

As a result, we have been encountered a lot of well-established protocols on e-auction which have been used in different areas. The protocol we mainly focused on, in this project, has been adapted from the protocol of anonymous e-voting protocol to achieve linear complexity as well. Now since the problem of e-auction is itself an instance of Secure Multi-party computation, we can fairly say that the the security aspects of the protocols will further be studied in coming years. As a result, we can further further encounter research works in the intersection parts of different topics of cryptology. For example, recent advances indicate that quantum computers may soon be reality. Motivated by this ever more realistic threat for existing classical cryptographic protocols, researchers have developed several schemes to resist "quantum attacks". In particular, for electronic voting, several e-voting schemes relying on properties of quantum mechanics have been proposed. We can certainly expect a new direction of research into the field of quantum e-auction as well in future.

Finally, to conclude this project, I must acknowledge the fact that the studying and started implementing of the backends of the SEAL protocol was indeed a challenging work. But the help from my supervisors and my seniors and friends made me cope up to overcome the burdens well. I have thoroughly enjoyed the work in last six months. We shall try to finish the front end using HTML5 so that we can have a proper implementation of the whole protocol, that could be handy for practical purposes.

References

[1] Chaum, D. The dining cryptographers problem: unconditional sender and recipient untraceability. *Journal of Cryptology* 1(1), 65–67 (1988). https://doi.org/10.1007/BF00206326

[2] Brandt, F. Secure and private auctions without auctioneers. *Technical Report FKI-245–02*. Institut fur Informatick, Technishce Universitat Munchen, 2002.

[3] Franklin, M K and Reiter, M K. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, vol. 22, no. 5, pp. 302–312, 1996.

[4] Sako, K. An auction protocol which hides bids of losers. *International Workshop on Public Key Cryptography*. Springer, 2000, pp. 422–432.

[5] Dining cryptographers problem. *Wikipedia*. Available at `https://en.wikipedia.org/wiki/Dining_cryptographers_problem`

[6] Diffie, W., and Hellman, M. E.: New Directions in Cryptography, *IEEE Transactions on Information Theory*, vol. 22, no.6, 644–654(1976).

[7] Bellare, Mihir and Rogaway, Phillip : Introduction to Modern Cryptography. Available at `https://web.cs.ucdavis.edu/ rogaway/classes/227/spring05/book/main.pdf`

[8] Boneh, D.: The decision Diffie-Hellman problem. In: Buhler, J.P. (ed.) ANTS 1998. LNCS, 1423, pp. 48–63. Springer, Heidelberg (1998)

[9] F. Hao and P. Zielinski, A 2-round anonymous veto protocol, *International Workshop on Security Protocols*. Springer, 2006, pp. 202–211.

[10] Hao, Feng and Ryan, Peter and Zielinski, Piotr. (2010). Anonymous voting by two-round public discussion. *Information Security, IET*. 4. 62 - 67. 10.1049/iet-ifs.2008.0127.

[11] Auction. *Wikipedia*. Available at `https://en.wikipedia.org/wiki/Auction`

[12] Auction. *Corporate Finance Institute Article*. Retrieved form CFI page `https://corporatefinanceinstitute.com/resources/knowledge/finance/auction/`

[13] S. Bag, F. Hao, S. F. Shahandashti and I. G. Ray, SEAL: Sealed-Bid Auction Without Auctioneers, *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2042-2052, 2020, doi: 10.1109/TIFS.2019.2955793.

[14] H. S. Galal and A. M. Youssef, "Verifiable sealed-bid auction on the ethereum blockchain," in 2018 Financial Cryptography and Data Security Workshops on Trusted Smart Contracts, 2018, pp. 265–278.

[15] Patrick McCorry. 2017. anonymousvoting (Open Vote Network). Retrieved from GitHub page `https://github.com/stonecoldpat/anonymousvoting`

[16] Hisham S. Galal. 2018. Verifiable Sealed-bid Auction on Ethereum Blockchain. Retrieved from GitHub page `https://github.com/HSG88/AuctionContract`

[17] Andreas Olofsson. 2016. *secp256k1 Implementation*. GitHub page link : `https://github.com/androlo/standard-contracts/blob/master/contracts/src/crypto/Secp256k1.sol`

[18] Andreas Olofsson. 2016. *ECCMath Implementation*. GitHub page `https://github.com/androlo/standard-contracts/blob/master/contracts/src/crypto/ECCMath.sol`

[19] Goldwasser, S. and Micali, S. and Rackoff, C. "The knowledge complexity of interactive proof systems" , *SIAM Journal on Computing*, 186–208, DOI :10.1137/0218012, 1989, ISSN 1095-7111

[20] Sherman, Alan T. and Javani, Farid and Zhang, Haibin and Golaszewski, Enis. On the Origins and Variations of Blockchain Technologies. *IEEE Security Privacy*. 72–77. arXiv:1810.06130. doi:10.1109/MSEC.2019.2893730. ISSN 1558-4046. S2CID 53114747.

[21] Nakamoto, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. Available at `http://www.bitcoin.org/bitcoin.pdf`

[22] Buterin, Vitalik. Ethereum Whitepaper. Originally published in 2013. Available at `https://ethereum.org/en/whitepaper/`

[23] Ethereum. *Wikipedia*. Available at `https://en.wikipedia.org/wiki/Ethereum`

[24] Naor, M. and Pinkas, B. and Sumner, R. Privacy preserving auctions and mechanism design. *EC*, vol. 99, pp. 129–139, 1999.

[25] Abe, M. and Suzuki, K. $(M + 1)$-st price auction using homomorphic encryption. *Public Key Cryptography*. vol. 2274. Springer, 2002, pp. 115–124.

[26] Montenegro, J. A. and Fischer, M. J. and Lopez, J. and Peralta, R. Secure sealedbid online auctions using discreet cryptographic proofs. *Mathematical and Computer Modelling*, vol. 57, no. 11, pp. 2583–2595, 2013.

[27] Lipmaa, H. and Asokan, N. and Niemi, V. Secure vickrey auctions without threshold trust. *International Conference on Financial Cryptography*. Springer, 2002, pp. 87–101.

[28] Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. *Financial Cryptography and Data Security*. 21st International Conference, volume 10322 of Lecture Notes in Computer Science, pages 357–375. Springer, 2017.