BOREALIS: Building Block for Sealed Bid Auctions on Blockchains

Erik-Oliver Blass
Airbus, Munich
erik-oliver.blass@airbus.com

Florian Kerschbaum
University of Waterloo
florian.kerschbaum@uwaterloo.ca

August 21, 2020

Abstract

We focus on securely computing the ranks of sealed integers distributed among n parties. For example, we securely compute the largest or smallest integer, the median, or in general the k^{th}-ranked integer. Such computations are a useful building block to securely implement a variety of sealed-bid auctions. Our objective is efficiency, specifically low interactivity between parties to support blockchains or other scenarios where multiple rounds are time-consuming. Hence, we dismiss powerful, yet highly-interactive MPC frameworks and propose BOREALIS, a special-purpose protocol for secure computation of ranks among integers. BOREALIS uses additively homomorphic encryption to implement core comparisons, but computes under distinct keys, chosen by each party to optimize the number of rounds. By carefully combining cryptographic primitives, such as ECC Elgamal encryption, encrypted comparisons, ciphertext blinding, secret sharing, and shuffling, BOREALIS sets up systems of multi-scalar equations which we efficiently prove with Groth-Sahai ZK proofs. Therewith, BOREALIS implements a multi-party computation of pairwise comparisons and rank zero-knowledge proofs secure against malicious adversaries. BOREALIS completes in at most 4 rounds which is constant in both bit length ℓ of integers and the number of parties n. This is not only asymptotically optimal, but surpasses generic constant-round secure multi-party computation protocols, even those based on shared-key fully homomorphic encryption. Furthermore, our implementation shows that BOREALIS is very practical. Its main bottleneck, ZK proof computations, is small in practice. Even for a large number of parties ($n = 200$) and high-precision integers ($\ell = 32$), computation time of all proofs is less than a single Bitcoin block interval.

1 Introduction

Sealed-bid auctions [15, 19] are commonly used when the true valuation of items is sought, i.e., to minimize strategic behavior of bidders. There are many types of sealed-bid auctions currently used in practice: first-price auctions, second-price (Vickrey) auctions, reverse auctions, uniform multi-unit auctions and many more. They differ in the ranks of parties winning the bid and the rank of the bid paid by auction winners.

Yet, all auctions share as a core building block the ability to compute ranks of sealed bids. In this paper, we focus on securely computing ranks of sealed bids in a distributed system. A secure and distributed computation of bid rankings and therewith sealed-bid auctions allows replacing trust in an auctioneer by a cryptographic protocol on top of, e.g., blockchains. For the implementation of auctions, we envision the following high-level protocol.

First, all parties securely compare their bids without revealing them. Then each party proves in zero-knowledge to the others whether its integer is k^{th} ranked, i.e., whether its rank among the set of n bids is k or not. For example, in a first-price auction, the winner would prove that its integer is ranked 1 and then disclose its integer. In a reverse auction, the winner proves that its integer is ranked n and discloses its integer. In a second-price auction, the winning party proves its integer being ranked 1, and the party with the second ranked integer proves rank 2 and discloses its integer. In a uniform multi-unit auction, parties with ranks 1, 2, and so forth subsequently prove their integers' ranks and disclose integers. Various other auction types can be constructed in this manner. Realizing comparisons and proofs above as a building block secure against fully-malicious adversaries is technically challenging. For example, one challenge is that rank computation must guarantee output delivery (in case of honest majority). That is, we must deal with, e.g., malicious parties aborting the protocol and not proving their rank after learning partial results of the auction.

Note that this building block of pairwise comparisons and rank computation is of general, independent interest in a wider set of problems in distributed databases [2, 16, 29, 41, 52].

We consider the problem of (integer) rankings on blockchains, because blockchains are becoming a popular platform for auctions. Users are increasingly migrating auctions to the blockchain [6, 49], since the immutable history of a blockchain provides an automatic advantage in addition to their distributed nature and therewith the lack of a trusted auctioneer. However, running an auction on top of today's blockchains such as Bitcoin or Ethereum also brings another technical challenge. Blockchains typically have large block interval times up to several minutes. Party interaction using the blockchain, e.g., to broadcast or send a message on the blockchain, is therefore expensive in terms of latency. Any protocol for securely comparing integers with high interactivity, i.e., a large number of rounds, implies an equally large number of blockchain blocks and would quickly become useless for many scenarios. So, the goal is a protocol for securely comparing n integers with low latency.

Designing secure computation protocols with low latency is technically challenging, as generic techniques for multi-party computation [5, 9, 27, 35] induce a large number of communication rounds. In general, the number of rounds is linear in the (multiplicative) depth of the circuit computed by the parties. While there exists recent research focusing on constant-round protocols [43, 44], based on the technique in [8], these works still require a considerable number of rounds, namely at least 9, and moreover expensive fully or somewhat-homomorphic encryption (SHE).

This paper: We design BOREALIS ("Building blOck foR sEALed bId auctionS"), a special-purpose protocol for securely computing ranks of n integers distributed among up to n parties. We distinguish between information each party participating in the protocol learns about bids and information disclosed on the blockchain, also available to and verifiable by parties not participating in the protocol, but only observing the blockchain, e.g., the seller in the auction.

More formally, we consider the following problem: given a sequence of n integers $(v_i)_{i=1,\ldots,n}$ of ℓ bits each where each v_i is held by a different party P_i, our goal is to securely compare the integers and compute the rank of $(v_i)_{i=1,\ldots,n}$, such that P_i can later prove (in zero-knowledge) whether it holds the k^{th}-ranked integer from that sequence. Hence, each party can publicly prove whether it won the auction or not without revealing any additional information. BOREALIS supports parties with multiple input integers each by simulating additional parties for each integer. BOREALIS is secure against malicious adversaries and guarantees output delivery as long as the majority of integers comes from honest parties. The actual value of the k^{th}-ranked integer or auxiliary values, such as the amount of items bid on in multi-unit auctions, can also be computed simply by provably decrypting an integer held by P_i.

Given the large variety of building blocks for secure computation, a new efficient solution requires careful design. We explain our objectives and justify our design decisions in Section 1.1. The practical efficiency of our protocol is due to optimized cryptographic engineering. We use a number of ingredients, such as Groth and Sahai [36]'s framework to realize our zero-knowledge proofs (Section 5). We also provide security definition (Section 3), security proof (Section 6), and a practical evaluation (Section 7).

1.1 Design Choices

The main idea to realize above sealed-bid auctions is to devise a new comparison mechanism which allows a party to securely compute and prove that its integer is ranked k^{th} among all integers. Thus, if the k^{th}-ranked bid wins the auction or multiple differently ranked bids win the auction as in the case of multi-unit auctions, corresponding parties can use our new comparison mechanism as a building block and prove winning bids. That is, we focus on computing pairwise comparisons of $n \geq 2$ integers held by different parties, but without revealing significantly more than the rank of each integer. In particular, we do not want to reveal the exact integer values. Our design objectives are:

- Security against *malicious* adversaries, assuming honest majority of parties.
- Practical efficiency for a large number, e.g., dozens, of participating parties and the *minimum number of rounds*. A low number of rounds implies low latency and allows deploying our solution in scenarios where rounds are costly such as with blockchains.

To hide integer values of parties while comparing, our first design decision is to choose additively homomorphic encryption. Multi-party computation (MPC), even constant round MPC [8, 43, 44], requires many rounds of interaction and expensive SHE. For example, Lindell et al. [43] need 16 rounds of interaction and $O(n^3)$ encryptions. Alternatively, Lindell et al. [44] need 9 rounds and $O(n^2)$ SHE encryptions, but additionally the SHE evaluation of a circuit with multiplicative depth 4. See Fig. 1 in [44] for a comparison. In contrast, our approach with additively homomorphic encryption allows for efficient comparisons non-interactively in one round which is optimal.

1.1.1 Key Distribution

When using homomorphic encryption, there are two options regarding keys used and their distribution. Either, in option 1, all parties encrypt their integers and compute under a joint public key with a threshold shared private key. Alternatively, in option 2, each party chooses its own private, public key pair. With option 1, computation and zero-knowledge (ZK) proofs to achieve malicious security are simple. However, one needs to securely generate a distributed key, a threshold shared private key, which is expensive. Secure distributed key generation requires at least two additional rounds of interaction during the distribution phase in case no party cheats. In case a party cheats, additional rounds are required, cf. Gennaro et al. [34].

With option 2 (which we choose), computing comparisons requires (re-)encrypting one party's integer with the key of the other party. This makes ZK proofs complex, since we need to prove that a homomorphic comparison computation has been performed correctly, including (re-)encryption. That is, we must prove correctness of the homomorphic computation without revealing the input, in particular the ciphertext of one party's integer under the other party's public key. Revealing this ciphertext to the other party would obviously imply that the other party learns the corresponding integer. On the positive side, we do not need distributed key generation for a shared private key. Instead, we use a variation of verifiable secret sharing based on [50] during the first round of the main comparison protocol. This saves us two rounds of interaction.

Furthermore, our key insight is that when using an Elgamal-based variation of Damgård et al.'s technique [24, 25] (called DGK henceforth) for homomorphically comparing integers, we can use efficient elliptic curve Elgamal encryption in one single elliptic curve group for all parties. The main advantage when operating within one single group is that we can then construct for all parties Groth and Sahai [36] proofs to elegantly prove correctness of re-encryptions, comparisons, and integer shuffling. This leads to protocol BOREALIS which is not only secure against malicious parties, but also practically efficient: it requires one round for integer commitments, one round for comparison computations, and one round for proving which party holds the k^{th}-ranked integer.

1.1.2 Circuit Evaluation

To compare input integers of all n participants, we implement a circuit. This circuit consists of a sequence of pairwise comparisons. A single comparison of a pair of two arbitrary length integers can be efficiently implemented using unbounded fan-in gates in a circuit of multiplicative depth 2. The second level gate can be implemented using shuffling of ciphertexts with bit plaintexts, since it is a logical "or" with at most one true integer. The DGK technique realizes such a comparison circuit based on additively homomorphic encryption by scalar multiplication with one party's plaintext integer.

To compute the index of the k^{th}-ranked integer using a sequence of pairwise comparisons, one could perform n comparisons using a selection algorithm, but this circuit has multiplicative depth n. One could also sort the n integers [3] using a (partial) sorting network. However, resulting circuits would still have $\Omega(\log n)$ multiplicative depth and hence would either require more rounds of interaction or a homomorphic encryption scheme that efficiently supports $\log n$ consecutive multiplications and even more complex ZK proofs.

The choice we make is a compromise. We perform $O(n^2)$ comparisons, $n-1$ per party, but we perform them in parallel, in only one round, and including ZK proofs. This has the downside that each party P_i learns whether their v_i is smaller than another party's v_j. Yet, this allows us to support a very wide variety of sealed-bid auctions while performing heavy computations (comparisons) only once, since parties learn the ranks of their integers and can subsequently reveal them when necessary on the blockchain. Hence, we can divide the auction protocol into secure comparisons and proofs of ranks in zero-knowledge. As a consequence, we achieve our main design objectives: we provide security against malicious adversaries, and we obtain asymptotically optimal $O(1)$ latency which is low in practice (total of 3 rounds, 4 rounds if a party aborts).

1.2 Blockchain

For the purpose of this paper, a blockchain realizes a secure public ledger. Parties append transactions to the ledger, verifiable by everybody after one blockchain block interval latency. Transactions are signed with the originator's private key for authenticity and stored immutably. Based on the concept of transactions, blockchains allow storing custom bit strings in the ledger, e.g., Bitcoin's `OP_RETURN` opcode or a trivial mailbox smart contract in Ethereum. Therewith, a blockchain provides a reliable, authenticated broadcast channel for arbitrary data. Also, knowledge of a party's public key enables personal messages by encrypting with the public key and broadcasting the ciphertext.

Caveats: Note that, in practice, limits apply to the length of data stored per transaction. For example, `OP_RETURN` accepts bit strings up to 40 Byte length per transaction. So, longer messages must be split in multiple transactions. For simplicity, we assume that parties store (long) messages in a public bulletin board and use the blockchain only to store the messages' hash. We also stress that proof-of-work-based consensus in blockchains is not fork-free. The current block might become invalid in the future, after another χ blocks, if the blockchain agrees on another fork. However assuming honest majority, the probability that the current block becomes invalid after χ blocks is negligible in χ [33, 53]. In practice, parties often wait χ additional blocks until they accept the current block ($\chi = 6$ with Bitcoin).

1.3 Related Work

Secure computation of the k^{th}-ranked integer was introduced by Aggarwal et al. [1] as an important primitive for operations in distributed databases. It was used prominently in, e.g., data mining applications [41], data anonymization [2], social network analysis [16], decision tree learning [29], and top-k queries [52]. The protocol by Aggarwal et al. [1] was also one of the first sub-linear computation complexity MPC protocols. It requires only $O(\log k)$ comparisons to compute the k^{th}-ranked element in the two-party setting and $O(\ell)$ comparisons in the multi-party setting. However, it also requires $O(\log k)$ or $O(\ell)$ rounds, respectively. High round complexity has motivated the research presented in this paper, since there is a need to enable this important functionality in scenarios where rounds have high latency such as with blockchains.

A primitive used by any protocol for secure computation of (the index of) the k^{th}-ranked integer is secure integer comparison. Secure integer comparison can be either implemented using generic secure computation, but many special protocols improving the efficiency have been developed. Protocols for secure computation using homomorphic encryption have been developed by Garay et al. [32], for information-theoretic secure computation by Damgård et al. [26], and improved by Nishide and Ohta [48] and Catrina and De Hoogh [21]. Kolesnikov et al. [37] developed an improved circuit which can be used to optimize performance in various secure computation protocols. Fischlin [31] developed a protocol specifically for (somewhat-)homomorphic encryption. This protocol has been further refined by Damgård et al. [25] which is the comparison protocol BOREALIS is based on.

Secure sealed-bid auctions are deployed in the real-world [15]. For a survey on secure auctions, see [19]. Naor et al. [47] developed a secure auction protocol based on two servers. Cachin [20] developed a secure auction protocol using an oblivious third party which is also the setup in Damgård et al. [25]'s protocol. Brandt [18] developed an interactive protocol requiring only a constant number of rounds, but requires unary bid encoding and has later been shown to additionally require expensive zero-knowledge proofs [28]. Improved protocols for Vickrey (second price) auctions have been developed by Lipmaa et al. [45] and Suzuki and Yokoo [51]. BOREALIS can be used to implement secure auctions on the blockchain in a constant number of (three) rounds, using binary bid encoding and highly practical ZK proofs. The advantage of an unforgeable history using a blockchain for auctions has been demonstrated before by researchers [14], real-world auctions [49], and start-ups [6].

Some work has investigated the relation between MPC and blockchains. Kosba et al. [38] developed secure and private smart contracts in Hawk, a system which requires a manager overseeing all parties' input. Generic secure computation has been implemented on the blockchain by Andrychowicz et al. [4] and Zyskind et al. [54]. Both approaches require a number of rounds depending on the circuit depth, but achieve a notion of fairness. Fairness can also be achieved in off-chain multi-party protocols using incentives, e.g., by crypto-currencies, or trusted hardware on the blockchain [11, 12, 23, 39, 40]. In contrast, BOREALIS assumes honest majority, but optimizes efficiency. It requires only a constant number (three) of rounds, we have implemented it in software, and no party needs to reveal its input.

2 Preliminaries

Let $\{P_1, \ldots, P_n\}$ be a set of parties. Each party P_i has an ℓ bit integer v_i as input.

Groth and Sahai Proof Systems: To prove that v_i is the k^{th}-ranked integer among all inputs, this paper sets up systems of equations and proves their correctness in zero-knowledge using Groth and Sahai's framework [36]. While Groth and Sahai define ZK proofs in multiple different settings, we focus on the case of proving validity of systems of equations over bilinear symmetric external Diffie-Hellman (SXDH) groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, \mathcal{P}_1, \mathcal{P}_2)$. Here, $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_t$ are groups of prime order p, and $\mathcal{P}_1$ and $\mathcal{P}_2$ generate $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. Let λ be the security parameter and $|p| \in \mathsf{poly}(\lambda)$. Function $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ is a bilinear map.

We choose prime order SXDH bilinear groups, as the decisional Diffie-Hellman (DDH) assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$. Therefore, we will be able to use additively homomorphic Elgamal encryption over elements in $\mathbb{G}_1$. Moreover, there are efficient implementations of Type-3 elliptic curves available which realize SXDH groups [46]. For more details about parameters of our implementation, we refer to Section 7.

With Groth and Sahai's framework [36] defined over SXDH groups, we prove validity of systems of equations in ZK. Using Groth and Sahai's notation, we set ring $\mathcal{R} = \mathbb{Z}_p$ and modules $A_1 = \mathbb{G}_1, A_2 = \mathbb{Z}_p$, $A_T = \mathbb{G}_1$. This allows proving equations of multi-scalar multiplications over $\mathbb{G}_1$ of type

$$\vec{y} \cdot \vec{\gamma_1} + \vec{\gamma_2} \cdot \vec{x} + \vec{x} \cdot \Gamma \cdot \vec{y} = t, \tag{1}$$

where $t \in \mathbb{G}_1, \gamma_1 \in \mathbb{G}_1^n, \gamma_2 \in \mathbb{Z}_p^m$, and $\Gamma \in \mathbb{Z}_p^{m \times n}$ are publicly known (called constants). The secret witnesses (called variables) in such proofs are $\vec{x} \in \mathbb{G}_1^m$ and $\vec{y} \in \mathbb{Z}_p^n$. Roughly speaking, we can prove equations combining secret elements from $\mathbb{G}_1$ with public elements from $\mathbb{Z}_p$, public elements from $\mathbb{G}_1$ with secret elements from $\mathbb{Z}_p$, and secret elements from both $\mathbb{G}_1$ and $\mathbb{Z}_p$. We denote both types of commitments, commitments to integers $x \in \mathbb{Z}_p$ and commitments to points $x \in \mathbb{G}_1$, simply by $\mathsf{Com}(x)$. We also simplify generation of a (random) common reference string in the SXDH setting by hashing the latest λ block hashes of the blockchain and use that as input to a PRG. For details on commitments and CRS requirements in the SXDH setting, see § 9 in [36].

Additively Homomorphic Elgamal: As the DDH assumption holds in elliptic curve point group $\mathbb{G}_1$, we can use additively homomorphic Elgamal encryption. For private key $sk \xleftarrow{\$} \mathbb{Z}_p$, let $pk = sk \cdot \mathcal{P}_1$ be the public key. To encrypt plaintext $m \in \mathbb{Z}_p$, randomly choose $r \xleftarrow{\$} \mathbb{Z}_p$ and compute ciphertext $c = E_{pk}(m) = (r \cdot \mathcal{P}_1, r \cdot pk + m \cdot \mathcal{P}_1)$. In this paper, we will write $c[0]$ for the left-hand part $r \cdot \mathcal{P}_1$ of ciphertext c and $c[1]$ for c's right-hand part $r \cdot pk + m \cdot \mathcal{P}_1$. To decrypt c, first compute $m \cdot \mathcal{P}_1 = c[1] - sk \cdot c[0]$ and then solve the elliptic curve discrete logarithm problem (ECDLP) to get m. Due to the computational hardness of ECDLP, m can be recovered only for small values of m. Yet, as we will see, in this paper it will be sufficient to check whether $m = 0$, which is easy. We have $m = 0$, *iff* $c[1] - sk \cdot c[0] = \mathcal{O}$, the point at infinity. Note the additively homomorphic property of this encryption: for ciphertexts $c_1 = (r_1 \cdot \mathcal{P}_1, r_1 \cdot pk + m_1 \cdot \mathcal{P}_1)$ and $c_2 = (r_2 \cdot \mathcal{P}_1, r_2 \cdot pk + m_2 \cdot \mathcal{P}_1)$, decrypting ciphertext $(c_1[0] + c_2[0], c_1[1] + c_2[1])$ results in $(m_1 + m_2) \cdot \mathcal{P}_1$ and therewith $m_1 + m_2$.

Long-Term Key Pairs: For each party P_i, let $sk_i^{\mathsf{lt}} \in \mathbb{Z}_p$ be P_i's long term private key and $pk_i^{\mathsf{lt}} = sk_i^{\mathsf{lt}} \cdot \mathcal{P}_1$ be P_i's long term public key. Assume all parties know other parties' long-term public keys.

3 Security Definition

We define security following the standard ideal vs. real world paradigm. First, we specify an ideal functionality $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$ of our protocol to compute the index of the k^{th}-ranked integer, see Algorithm 1. This functionality encompasses both steps in an auction protocol: the comparison of integers and the proof of rank k. For any specific auction type, these two steps should be appropriately adapted.

In this ideal functionality, a trusted third party TTP receives all input integers v_i from all parties P_i. If a malicious party P_i submits an invalid input $v_i = \perp$, then v_i is excluded from the computation. The TTP then computes results $\gamma_{\hat{i},\hat{j}}$ of comparisons between integers from parties $P_{\hat{i}}$ with valid integers $v_{\hat{i}} \neq \perp$. The k^{th}-ranked integer is v_ι with index ι. Parties $P_{\hat{i}}$ who submitted a valid integer $v_{\hat{i}} \neq \perp$ receive from the TTP the result of each comparison between their integer $v_{\hat{i}}$ and all other integers $v_{\hat{j}}$. Finally, the TTP sends to everybody via broadcast on the blockchain index ι of the k^{th}-ranked integer v_ι and the result of the comparisons between each bid $v_{\hat{j}} \neq \perp$ and v_ι.

Note that if and only if a (malicious) party P_i submits a valid integer v_i, then v_i is included in the computation of TTP. Assuming the blockchain is a broadcast channel, we also guarantee delivery of TTP's output.

```
for $i = 1$ to $n$ do
    $P_i \to TTP$: $v_i \in \{\bot, 0, \ldots 2^\ell - 1\}$
end
// Let $\hat{I} = \{i | v_i \neq \bot\}, \hat{n} = |\hat{I}|$.
// $\forall \hat{i}, \hat{j} \in \hat{I}, \hat{j} \neq \hat{i}$ : Let $\gamma_{\hat{i},\hat{j}} = 1$, if $v_{\hat{i}} > v_{\hat{j}}$ and $\gamma_{\hat{i},\hat{j}} = 0$ otherwise.
// Let $\iota$ be the index of $k^{\text{th}}$-ranked integer $v_\iota \in \hat{I}$.
foreach $\hat{i} \in \hat{I}$ do
    $TTP \to P_{\hat{i}} : \{\gamma_{\hat{i},\hat{j}} | \hat{j} \in \hat{I} \land \hat{j} \neq \hat{i}\}$
end
// Via broadcast (on blockchain)
$TTP \to \star : (\iota, \{\gamma_{\iota,\hat{j}} | \hat{j} \in \hat{I}\})$;
```

Algorithm 1: Ideal Functionality $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$

We stress that functionality $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$ reveals more than achievable by generic MPC. Each party $P_{\hat{i}}$ learns whether another party $P_{\hat{j}}$'s input $v_{\hat{j}}$ is less or greater than $v_{\hat{i}}$ and less or greater than k^{th} integer v_ι. However, neither the actual values of $v_{\hat{j}}$ or v_ι nor results of other parties' comparisons are disclosed. While $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$'s security is weaker than general MPC, the key advantage of $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$ is that it is generic among many auction types and enables us to still implement an efficient protocol with an optimal number of rounds, i.e., low latency on the blockchain. In addition, we expect the additional leakage compared to MPC to be acceptable in many real-world scenarios.

We consider a static, active adversary $\mathcal{A}$ that controls up to $\tau < \frac{n}{2}$ parties. All attacks admissible in the real implementation of the protocol correspond to an attack in the ideal world implementation using a trusted third party. The following Theorem 1 summarizes our main contribution.

Theorem 1. *If adversary $\mathcal{A}$ is static, active, and controls up to $\tau < \frac{n}{2}$ parties P_i, then protocol BOREALIS securely implements functionality $\mathcal{F}_{k^{th}\text{-}Ideal}$.*

4 BOREALIS Description

Before presenting technical details, we start by giving a high-level overview over BOREALIS' intuition and its main concepts in Section 4.1. To ease understanding, we then present core comparisons by an example walk-through with just two parties and two bit input integers in Section 4.2. We finally give full and formal details of BOREALIS with pseudo-code in Section 4.3.

To furthermore ease exposition, our protocol presentation below assumes existence of multiple new ZK proofs. In Section 5, we present formal details on how we generate these proofs. Our last simplification is that, for now, we pretend that integers are pairwise different. Later, in Section 4.4 we will explain how to enforce distinct input integers with little overhead.

4.1 High-Level Overview

BOREALIS' main idea for sealed-bid auctions is to employ a new comparison mechanism as a building block. Comparisons allow party P_i with the k^{th}-ranked integer to prove this fact to other parties in ZK. As specified by the auction, parties use the comparison mechanism and prove that their integers have certain ranks. In the following, we focus only on the core comparison mechanism, where eventually P_i proves rank k. The extension how multiple parties prove their ranks will become obvious later.

So, assume that n parties have agreed to jointly compute the index of the k^{th} ranked integer of their input integers on a blockchain. Each party P_i has input integer $v_i \in \mathbb{N}, |v_i| = \ell$. We denote v_i's bit representation by $v_i = v_{i,\ell} \dots v_{i,1}$, i.e., $v_{i,1}$ is the least significant bit of v_i.

In BOREALIS' first round, each party P_i encrypts each bit $v_{i,j}$ with additively homomorphic Elgamal encryption and their own public key pk_i. Party P_i publishes ciphertexts on the blockchain.

In the second round, each party P_i homomorphically evaluates a DGK comparison circuit with ciphertexts from other parties P_j using their own v_i as input. Results of these homomorphic evaluations are ℓ ciphertexts for each of the $n-1$ other parties. Party P_i publishes these ℓ ciphertext on the blockchain.

In the third and final round, each party P_i decrypts all ℓ ciphertexts of each of the $n-1$ other parties. For the ℓ decrypted evaluations of another party P_j, P_i determines whether $v_i < v_j$ as follows. If exactly one of the ℓ evaluations decrypts to 0, then $v_i < v_j$, otherwise $v_i \geq v_j$. The one party P_ι with the k^{th} integer v_ι has $n-k-1$ comparisons $v_\iota < v_j$ and k comparisons $v_\iota \geq v_j$. Party P_ι announces ι on the blockchain (and reveals v_ι if required by the auction).

Technical Challenges: While the above protocol overview seems straightforward, it is obviously insecure. To protect against malicious adversaries, one has to, e.g., prove correctness of DGK evaluations on the blockchain. The proof of correctness, however, must be in ZK not to leak details about an input v_i. Along the same lines, second round comparisons require blinding and shuffling of input. Blinding and shuffling requires (non-trivial) correctness proofs which must be in ZK, too. Finally, we have to cope with malicious parties aborting the protocol. Our main contribution is thus to solve these technical challenges and enable maliciously-secure computation of the index of the k^{th} integer in $O(1)$ rounds of interaction.

We now present a simplified (two party, two bit) version of BOREALIS' core technique, maliciously secure integer comparison.

4.2 Two Party, Two Bit Walk-Through

Assume two parties P_1 = Alice and P_2 = Bob. Let Alice's input integer be $v_a = v_{a,2}v_{a,1}$ and Bob's be $v_b = v_{b,2}v_{b,1}$. Alice and Bob want to compute a generic comparison, i.e., whether $v_a < v_b$. In the clear they would compute

$$\begin{aligned} c_1 &= v_{a,1} - v_{b,1} + 1 + v_{a,2} + v_{b,2} - 2 \cdot v_{a,2} \cdot v_{b,2} \\ c_2 &= v_{a,2} - v_{b,2} + 1. \end{aligned} \tag{2}$$

We know $v_a < v_b$, *iff* either c_1 or c_2 equals zero. Note that both c_1 and c_2 cannot be 0 at the same time, see DGK [24].

To protect against fully-malicious adversaries, our idea is to evaluate DGK in the encrypted domain and prove evaluation correctness in ZK. We set up a system of equations of multi-scalar multiplications and prove them in Groth and Sahai's framework.

A small technicality arises from the fact that computation of c_1 in Equations (2) is not mod 2, but in the integers. To avoid "wrap-around", we require input bit length ℓ to be less than group order p. With $|p| \in \mathsf{poly}(\lambda)$ being a security parameter, this always holds.

4.2.1 First Round

Below, we refer to multiple new ZK proofs. Technical details about computing these proofs are in Section 5. Let Alice's public key be $pk_A = sk_A \cdot \mathcal{P}_1$ with private key $sk_A \in \mathbb{Z}_p$. Bob's public key is $pk_B = sk_B \cdot \mathcal{P}_1$ with private key $sk_B \in \mathbb{Z}_p$.

Alice computes Groth and Sahai commitments for $v_{a,1}, v_{a,2}, sk_A$ and randomly chosen $r_A, r'_A, R_A, R'_A \in \mathbb{Z}_p, \beta_A \in \{0,1\}$. The exact meaning of each variable will become clear below. She publishes commitments on the blockchain together with encryptions

$$\begin{aligned} E_{pk_A}(v_{a,1})[0] =& r_A \cdot \mathcal{P}_1 \\ E_{pk_A}(v_{a,1})[1] =& r_A \cdot pk_A + v_{a,1} \cdot \mathcal{P}_1 \\ E_{pk_A}(v_{a,2})[0] =& r'_A \cdot \mathcal{P}_1 \\ E_{pk_A}(v_{a,2})[1] =& r'_A \cdot pk_A + v_{a,2} \cdot \mathcal{P}_1 \end{aligned} \tag{3}$$

She also computes a ZK proof that $E_{pk_A}(v_{a,1})$ and $E_{pk_A}(v_{a,2})$ are encryptions of $v_{a,1}$ and $v_{a,2}$, i.e., she proves Equations (3). In addition, she prepares a ZK proof that $v_{a,1}, v_{a,2}$, and β_A are bits. Alice publishes ZK proofs on the blockchain.

Similarly, Bob commits to $v_{b,1}, v_{b,2}, sk_B, r_B, r'_B, R_B, R'_B \in \mathbb{Z}_p, \beta_B \in \{0,1\}$ and computes

$$\begin{aligned} E_{pk_B}(v_{b,1})[0] =& r_B \cdot \mathcal{P}_1 \\ E_{pk_B}(v_{b,1})[1] =& r_B \cdot pk_B + v_{b,1} \cdot \mathcal{P}_1 \\ E_{pk_B}(v_{b,2})[0] =& r'_B \cdot \mathcal{P}_1 \\ E_{pk_B}(v_{b,2})[1] =& r'_B \cdot pk_B + v_{b,2} \cdot \mathcal{P}_1 \end{aligned}$$

and publishes everything together with corresponding ZK proofs on the blockchain. This concludes the first round.

As one can see, Bob performs the exact same computation as Alice using his input. Thus, in the following more involving computation of the comparison circuit, we just describe Alice's computation and remark that Bob performs the same, but uses his input.

4.2.2 Second Round

Alice sees Bob's ciphertexts $E_{pk_B}(v_{b,1}), E_{pk_B}(v_{b,2})$ on the blockchain. She now computes c_1, c_2 in the encrypted domain, i.e., an encrypted DGK evaluation of Bob's ciphertexts $E_{pk_B}(v_{b,1}), E_{pk_B}(v_{b,2})$ with Alice's input $v_{a,1}, v_{a,2}$:

$$\begin{aligned} c_1[0] =& - E_{pk_B}(v_{b,1})[0] + E_{pk_B}(v_{b,2})[0] \\ & - 2 \cdot v_{a,2} \cdot E_{pk_B}(v_{b,2})[0] \\ c_1[1] =& v_{a,1} \cdot \mathcal{P}_1 - E_{pk_B}(v_{b,1})[1] + \mathcal{P}_1 + v_{a,2} \cdot \mathcal{P}_1 \\ & + E_{pk_B}(v_{b,2})[1] - 2 \cdot v_{a,2} \cdot E_{pk_B}(v_{b,2})[1] \\ c_2[0] =& - E_{pk_B}(v_{b,2})[0] \\ c_2[1] =& v_{a,2} \cdot \mathcal{P}_1 - E_{pk_B}(v_{b,2})[1] + \mathcal{P}_1 \end{aligned} \tag{4}$$

Alice could send c_1, c_2 to Bob by publishing them on the blockchain, and Bob could then decrypt them. If one of them decrypts to 0, Bob would know $v_a < v_b$. However with this approach, Bob would derive more information about v_a than just whether $v_a < v_b$. Bob would learn *which* of the two ciphertexts decrypts to zero, so would know which bit in v_a differs from its corresponding one in v_b. Moreover, Bob would learn the exact integer of Alice's DGK evaluation for each input bit. As evaluation takes place in the integers, see Equations (2), Bob would learn the exact number of bits differing between v_a and v_b.

To remedy both issues, Alice blinds and shuffles ciphertexts c_1, c_2 before sending to Bob. The purpose of blinding is that encryptions of 0 still decrypt to 0, but encryptions of anything non 0 do not decrypt. Shuffling ciphertexts will hide the position of a potential 0.

Blinding: Alice blinds c_1, c_2 to c'_1, c'_2 by multiplying each part of an Elgamal ciphertext with (previously committed) $R_A, R'_A \in \mathbb{Z}_p$:

$$\begin{aligned} c'_1[0] &= R_A \cdot c_1[0] \quad & c'_1[1] &= R_A \cdot c_1[1] \\ c'_2[0] &= R'_A \cdot c_2[0] \quad & c'_2[1] &= R'_A \cdot c_2[1] \end{aligned} \tag{5}$$

If a ciphertext c_i encrypts $m = 0$, then $c_i = (R_A \cdot r \cdot \mathcal{P}_1, R_A \cdot r \cdot pk_B)$ for some r, and Bob can decrypt immediately. For $m \neq 0$, $c_i = (R_A \cdot r \cdot \mathcal{P}_1, R_A \cdot r \cdot sk_B \cdot \mathcal{P}_1 + R_A \cdot m \cdot \mathcal{P}_1)$, and Bob cannot decrypt due to the size of R_A and the ECDLP. Note that our blinding resembles the blinding by Damgård et al. [24]. Their specific encryption and blinding operate in an RSA group $\mathbb{Z}_{n=p \cdot q}$, but our variation above targets additively homomorphic Elgamal encryption over elliptic curves to prove Groth and Sahai equations.

Shuffling: Using the β_A, Alice shuffles c'_1, c'_2 to C_1, C_2:

$$\begin{aligned} C_1[0] &= \beta_A \cdot c'_1[0] + (1-\beta_A) \cdot c'_2[0] \\ C_1[1] &= \beta_A \cdot c'_1[1] + (1-\beta_A) \cdot c'_2[1] \\ C_2[0] &= (1-\beta_A) \cdot c'_1[0] + \beta_A \cdot c'_2[0] \\ C_2[1] &= (1-\beta_A) \cdot c'_1[1] + \beta_A \cdot c'_2[1] \end{aligned} \tag{6}$$

So if $\beta_A = 0$, Alice flips ciphertexts.

She now computes Groth and Sahai commitments for $c_1[0]$, $c_1[1]$, $c_2[0]$, $c_2[1]$, $c'_1[0], c'_1[1], c'_2[0], c'_2[1]$ and publishes these commitments together with $C_1[0]$, $C_1[1]$, $C_2[0]$, $C_2[1]$ on the blockchain. Alice also computes ZK correctness proofs for Equations (4), Equations (5), and Equations (6) and publishes proofs on the blockchain. This concludes Alice's second round in BOREALIS.

Bob computes the same steps using his input and publishes commitments, ciphertexts, and proofs on the blockchain accordingly.

4.2.3 Third Round

In the third and last round, Bob observes Alice's data from above on the blockchain. We now describe how Bob proves whether $v_a < v_b$ based on this data. Again, Alice will do the same, using Bob's blockchain data and her own input.

First, Bob verifies whether Alice's commitments, Groth and Sahai ZK proofs, and ciphertexts C_1, C_2 match. If so, Bob decrypts ciphertexts C_1, C_2. Each ciphertext is either an encryption of 0, i.e., $C_i = (r \cdot \mathcal{P}_1, r \cdot sk_B \cdot \mathcal{P}_1)$ or an encryption of some $m \neq 0$, i.e., $C_i = (r \cdot \mathcal{P}_1, r \cdot sk_B \cdot \mathcal{P}_1 + m \cdot \mathcal{P}_1)$.

Bob now publishes his decrypted integers and proves correct decryption as follows. Bob computes

$$C_{\text{final},1} = sk_B \cdot C_1[0] \quad C_{\text{final},2} = sk_B \cdot C_2[0] \tag{7}$$

and publishes $C_{\text{final},1}$, $C_{\text{final},2}$, and a ZK proof of Equations (7) on the blockchain. Knowledge of $C_{\text{final,i}}$ allows everybody (including Alice) to verify whether m_i is 0, just by computing $m_i = C_i[1] - C_{\text{final,i}}$. If exactly one of m_i is 0, then everybody knows that $v_a < v_b$. So, the proof of correct decryption is simply a proof of correctness of Equations (7): correctly multiplying the left-hand side of the Elgamal ciphertext with Bob's private key. This allows Alice to then decrypt C_i by herself.

There is, however, yet another caveat. As Alice could undo her permutation of the second round (and her blinding), she would learn the position of the 0 and therewith the exact bit differing between her and Bob's integers. To remedy, Bob also blinds C_1, C_2, shuffles them, and proves correctness as in the second round before computing the C_{final}. This concludes the third round.

Discussion: Note that in this special case of two parties there cannot be a honest majority, so $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$ security against malicious adversaries is not achievable. Consider, e.g., the case of Alice aborting already after the first round: it will be impossible for Bob to output the result of the comparison. Still, we present the case of two parties here, as it helps understanding the main comparison for the case of $n > 2$ parties.

Along the same lines, it is actually unnecessary in the case of only two parties to run the third round and publish the outcome of evaluations. Both Alice and Bob already know after the second round whether $v_a < v_b$. Still, we include the description of the third round here, as it is crucial for security in the case of $n > 2$ parties where a minority of $\tau < \frac{n}{2}$ can be fully malicious. The idea later in Section 4.3 will be that the party P_ι with the k^{th}-ranked input integer v_ι will prove that it has the k^{th}-ranked integer, and all other parties will prove that they do not have the k^{th}-ranked integer. We will cope with malicious parties aborting the protocol or cheating in their ZK proofs by revealing their input integers.

4.3 Full Details

We now present BOREALIS' full details for an arbitrary number of parties $n > 2$ and arbitrary integer bit length $\ell \geq 1$. We institute two major changes to the simplified two-party, two-bit protocol. For $n > 2$ parties, we can achieve malicious security, if the majority of parties is honest. First, we verifiably secret share each party's private key during the first round, using the blockchain as a broadcast channel. Second, we append another round on demand. If a malicious party P_i is aborting the protocol at any time or caught cheating in their ZK proofs, (honest) parties agree to run another round. In this round, parties will re-assemble shares of P_i's secret key and reveal P_i's input integer. Thereby, we determine the party with the k^{th}-ranked integer, even if this integer comes from a malicious P_i.

4.3.1 First Round

Algorithm 2 presents details for BOREALIS' first round. The first step in this first round is, for each party, to generate and secret share a fresh session key. While there exists a large body of work on efficient (publicly) verifiable secret sharing (VSS), we use a new variation of Schoenmakers [50]'s solution due to its simplicity and efficiency. We briefly summarize our variation in Appendix A and only state its main property here.

Let $\mathsf{VSS}(t, n, \mathbb{G}_1, pk_1^{\mathsf{lt}}, \ldots, pk_n^{\mathsf{lt}})$ be a verifiable secret sharing scheme. Parameter n denotes the total number of parties, t the number of parties required to reconstruct a secret, $\mathbb{G}_1$ a group where the DDH holds, and $pk_1^{\mathsf{lt}}, \ldots, pk_n^{\mathsf{lt}}$ the parties long-term public keys. Note that public keys are of type $pk_i^{\mathsf{lt}} = sk_i^{\mathsf{lt}} \cdot \mathcal{P}_1$ where $\mathcal{P}_1$ generates $\mathbb{G}_1$ and $sk_i^{\mathsf{lt}} \in \mathbb{Z}_p$ is the private key. As one can see, VSS accepts exactly BOREALIS' long-term public keys as input. The output of VSS is a random private key $sk \in \mathbb{Z}_p$, internal commitments $\mathcal{C}$, encryptions $\mathcal{Y}_j$ of shares under the other parties P_js' public keys, and a ZK proof $\mathsf{Proof}_{\mathsf{VSS}}$ proving consistency of the shares (Appendix A).

So, each party P_i invokes VSS, gets private session key sk_i, and computes public session key $pk_i = sk_i \cdot \mathcal{P}_1 \in \mathbb{G}_1$. P_i also generates random strings r_j

// Let P_i's long term public key be pk_i^{lt}. Let $\eta = \ell \cdot \log \ell - \frac{\ell}{2}$.

1 **foreach** party $P_i, 1 \leq i \leq n$ **do**

2 $(sk_i, \mathcal{C}_0, \ldots, \mathcal{C}_{\tau-1}, \mathcal{Y}_1, \ldots, \mathcal{Y}_n, \mathsf{Proof}_{\mathsf{VSS},i}) \leftarrow \mathsf{VSS}(\tau - 1, n, \mathbb{G}_1, pk_1^{\mathsf{lt}}, \ldots, pk_n^{\mathsf{lt}})$;

3 $pk_i = sk_i \cdot \mathcal{P}_1$;

4 $\{r_{i,1}, \ldots, r_{i,\ell}\} \xleftarrow{\$} \mathbb{Z}_p^{\ell}$;

5 $\{(R_{i,1,1}, \ldots, R_{i,1,\ell}), \ldots, (R_{i,n-1,1}, \ldots, R_{i,n-1,\ell})\} \xleftarrow{\$} \mathbb{Z}_p^{(n-1)\cdot\ell}$;

6 $\{(\beta_{i,1,1}, \ldots, \beta_{i,1,\eta}), \ldots, (\beta_{i,n-1,\ell}, \ldots, \beta_{i,n-1,\ell})\} \xleftarrow{\$} \{0,1\}^{(n-1)\cdot\eta}$;

7 publish $pk_i, \mathsf{Com}(sk_i), \mathsf{Proof}_{\mathsf{KeyECDLP},i}, \mathcal{C}_0, \ldots, \mathcal{C}_{\tau-1}$, $\mathcal{Y}_1, \ldots, \mathcal{Y}_n, \mathsf{Proof}_{\mathsf{VSS},i}, \mathsf{Com}(v_{i,1}), \ldots, \mathsf{Com}(v_{i,\ell}), \mathsf{Proof}_{\mathsf{Bit},i,1}, \ldots, \mathsf{Proof}_{\mathsf{Bit},i,\ell}$, $\mathsf{Com}(r_{i,1}), \ldots, \mathsf{Com}(r_{i,\ell}), c_{i,1} = E_{pk_i}(v_{i,1}), \ldots, c_{i,\ell} = E_{pk_i}(v_{i,\ell}), \mathsf{Proof}_{\mathsf{Enc},i,1}$, $\ldots, \mathsf{Proof}_{\mathsf{Enc},i,\ell}, \mathsf{Com}(R_{i,1,1}), \ldots, \mathsf{Com}(R_{i,1,\ell}), \ldots, \mathsf{Com}(R_{i,n-1,\ell})$, $(\mathsf{Com}(\beta_{i,1,1}), \ldots, \mathsf{Com}(\beta_{i,1,\eta})), (\mathsf{Com}(\beta_{i,n-1,1}), \ldots, \mathsf{Com}(\beta_{i,n-1,\eta}))$, $(\mathsf{Proof}_{\mathsf{Bit},i,1,1}, \ldots, \mathsf{Proof}_{\mathsf{Bit},i,1,\eta}), \ldots, (\mathsf{Proof}_{\mathsf{Bit},i,n-1,1}, \ldots, \mathsf{Proof}_{\mathsf{Bit},i,n-1,\eta})$ on blockchain;

8 **end**

Algorithm 2: BOREALIS' first round

for use in encryption, random strings R_j for blinding, and random bits β_j for shuffling, lines 2 to 6 in Algorithm 2. P_i publishes this information together with corresponding ZK proofs of correctness on the blockchain, see Line 7. In detail, P_i publishes:

- public key pk_i, a commitment to private key sk_i, ZK correctness proof $\mathsf{Proof}_{\mathsf{KeyECDLP}}$,
- VSS commitments $\mathcal{C}$, sk_i's encrypted shares $\mathcal{Y}$, and the VSS ZK proof of correctness,
- Groth and Sahai commitments to each bit $v_{i,j}$ and ZK proofs $\mathsf{Proof}_{\mathsf{Bit}}$ which prove that each $v_{i,j}$ is a bit; random strings $r_{i,j}$ used for encryptions of bits and additively homomorphic Elgamal encryptions $E_{pk_i}(v_{i,j}) = (r_{i,j} \cdot \mathcal{P}_1, r_{i,j} \cdot pk_i + v_{i,j} \cdot \mathcal{P}_1)$ of each bit; ZK proofs of correctness $\mathsf{Proof}_{\mathsf{Enc}}$ for each bit,
- commitments to $(n-1) \cdot \ell$ random R used later during blinding,
- commitments to $(n-1) \cdot (\ell \cdot \log \ell - \frac{\ell}{2})$ bits β used later during shuffling, and corresponding $(n-1) \cdot (\ell \cdot \log \ell - \frac{\ell}{2})$ ZK proofs $\mathsf{Proof}_{\mathsf{Bit}}$ that the βs are bits.

Note that all P_i perform their computations and publish their output in parallel at the same time. Therefore, the first round requires one block latency.

4.3.2 Second Round

Both the second and third round start by parties verifying ZK proofs. To keep exposition clean, we defer details about handling invalid ZK proofs as well as parties aborting protocol execution to Section 4.4. In the following, assume that proofs are successfully verified.

After verifying ZK proofs, the main part of the second round starts (Algorithm 3). Each party P_i homomorphically computes a DGK comparison for each other party. Specifically, P_i computes for each other party's integer v_j and

foreach party $P_i, 1 \leq i \leq n$ **do**
  foreach $j \neq i$ **do**
    for $u = 2$ **to** ℓ **do**
      // Compute XORs
      $w_u = E_{pk_j}(v_{j,u})[0] - 2 \cdot v_{i,u} \cdot E_{pk_j}(v_{j,u})[0]$;
      $W_u = v_{i,u} \cdot \mathcal{P}_1 + E_{pk_j}(v_{j,u})[1] - 2 \cdot v_{i,u} \cdot E_{pk_j}(v_{j,u})[1]$;
    end
    for $u = 1$ **to** ℓ **do**
      // Ciphertexts $c_{i,j,u} = (c_{i,j,u}[0], c_{i,j,u}[1])$
      $c_{i,j,u}[0] = -E_{pk_j}(v_{j,u})[0] + \sum_{\delta=u+1}^{\ell} w_\delta$;
      $c_{i,j,u}[1] = v_{i,u} \cdot \mathcal{P}_1 - E_{pk_j}(v_{j,u})[1] + \mathcal{P}_1 + \sum_{\delta=u+1}^{\ell} W_\delta)$;
      publish $\mathsf{Com}(c_{i,j,u}[0]), \mathsf{Com}(c_{i,j,u}[1]), \mathsf{Proof}_{\mathsf{DGK},i,j,u}$ on blockchain;
      // Blinded ciphertexts $c'_{i,j,u}$
      $c'_{i,j,u} = (R_{i,j,u} \cdot c_{i,j,u}[0], R_{i,j,u} \cdot c_{i,j,u}[1])$;
      publish $\mathsf{Com}(c'_{i,j,u}[0]), \mathsf{Com}(c'_{i,j,u}[1]), \mathsf{Proof}_{\mathsf{Blind},i,j,u}$ on blockchain;
    end
    // Shuffled ciphertexts $C_{i,j,u}$
    // Let $\eta = \ell \cdot \log \ell - \frac{\ell}{2}$
    $(C_{i,j,1}, \ldots, C_{i,j,\ell}, \mathsf{Proof}_{\mathsf{Shuffle},i,j,1}, \ldots, \mathsf{Proof}_{\mathsf{Shuffle},i,j,\eta}) = \mathsf{Benes}(\beta_{i,j,1}, \ldots, \beta_{i,j,\eta}, c'_{i,j,1}, \ldots, c'_{i,j,\ell})$;
    publish all ℓ ciphertexts $C_{i,j,u}$ and all η proofs $\mathsf{Proof}_{\mathsf{Shuffle},i,j,u}$ on blockchain;
  end
end

Algorithm 3: BOREALIS' second round

bit indices u expression

$$c_u = v_{i,u} - v_{j,u} + 1 + \sum_{\delta=u+1}^{\ell} v_{i,\delta} \oplus v_{j,\delta}.$$

For any two bits $v_{i,u}$ and $v_{j,u}$ at index u, c_u becomes 0 *iff* all bits "left" of index u are equal (sum of XORs is 0) and $v_{i,u} = 0$ and $v_{j,u} = 1$. So, $v_j > v_i$. Observe that there can be either one or no index u with $c_u = 0$.

We substitute $v_{i,u} \oplus v_{j,u}$ by $v_{i,u} + v_{j,u} - 2 \cdot v_{i,u} \cdot v_{j,u}$ and evaluate c_u in the encrypted domain. Variable w_u is an XOR used for the left-hand side of the evaluated ciphertext $c_{i,j,u}[0]$, and W_u is the XOR used in the right-hand side $c_{i,j,u}[1]$.

As discussed in the case of two parties, P_i needs to hide the exact results of the DGK expressions in order to hide its integer from P_j. Hence, P_i blinds and shuffles the c_u ciphertexts. The purpose of blinding is to hide non 0 integer values, and shuffling hides the position of a 0.

P_i blinds a homomorphic DGK evaluation c to c' by multiplying with previously committed random R, so encryptions of 0 decrypt, but anything non 0 does not due to the ECDLP. To prove correctness of homomorphic evaluations and blinding, P_i publishes commitments to the $c_{i,j,i}$ with corresponding ZK proofs $\mathsf{Proof}_{\mathsf{DGK}}$ of correctness and commitments to the $c'_{i,j,u}$ with correctness proofs $\mathsf{Proof}_{\mathsf{Blind}}$ on the blockchain.

The last step for P_i in the second round is to shuffle blinded ciphertexts c', see Line 14. To shuffle, BOREALIS employs a Beneš [10] permutation network using a call to function Benes. This function takes as input the $\eta = \ell \cdot \log \ell - \frac{\ell}{2}$ random, previously committed bits β and ℓ blinded ciphertexts c'. Internally, Benes sets up a ℓ input, ℓ output Beneš permutation and uses the β to implement internal crossbar switches. The output of Benes is a random (up to the β) permutation $C_1, \ldots, C_\ell$ of blinded ciphertexts c' together with η ZK proofs of correctness $\mathsf{Proof}_{\mathsf{Shuffle}}$ of correct crossbar switches. See Section 5.4 for more details on this proof. Finally, P_i publishes all ℓ blinded, shuffled ciphertexts C and all η proofs $\mathsf{Proof}_{\mathsf{Shuffle}}$ on the blockchain. Again, all parties compute and publish on the blockchain in parallel.

4.3.3 Third Round

In the third round, party P_i decrypts its ciphertexts C. Recall that for each other party P_j, P_i can decrypt ℓ ciphertexts $C_{j,i,1}, \ldots, C_{j,i,\ell}$. If exactly one decrypts to $\mathcal{O}$, then P_i knows $v_j < v_i$; if none decrypts to $\mathcal{O}$, then $v_j \geq v_i$. Note that it is impossible that more than one ciphertexts coming from one party P_j decrypt to $\mathcal{O}$ [24]. To simplify notation, we briefly introduce the following definition.

Definition 1 ($\mathcal{O}$- and $\not{\mathcal{O}}$-ciphertext sequences)**.** *A ciphertext sequence* $(C_{j,i,1}, \ldots, C_{j,i,\ell})$ *is called* $\mathcal{O}-$*ciphertext sequence* iff *exactly one* $C_{j,i,u}$ *decrypts to* $\mathcal{O}$*. Otherwise, this sequence is called* $\not{\mathcal{O}}$*-ciphertext sequence.*

Decrypting all $C_{j,i,u}$, party P_i computes its integer v_i's rank κ_i as follows. If there are $\kappa_i - 1$ sequences which are $\mathcal{O}$-ciphertext sequences and $n - \kappa_i$ sequences which are $\not{\mathcal{O}}$-ciphertext sequences, then v_i is ranked κ_i^{th} integer. This is the starting point of Algorithm 4, where each party P_i will now prove that their integer v_i is *either* the k^{th} integer ($\kappa_i = k$) *or* larger than the k^{th} integer ($\kappa_i > k$) *or* less than the k^{th} integer ($\kappa_i < k$).

To enable proofs in ZK, P_i first shuffles all $n-1$ ciphertexts $C_{j,i,u}$ using Beneš permutation networks, see Line 2. Note that P_i shuffles $n-1$ items, each being a sequence of ℓ shuffled ciphertexts. This is done by using a function called Benes^*. Besides a ZK proof $\mathsf{Proof}_{\mathsf{Shuffle}^*}$ of correct shuffle, Benes^* outputs a blinded, permuted sequence $C'_{j,i,u}$ of input plaintexts $C_{j,j,u}$.

Without loss of generality, assume that the first $\kappa_i - 1$ sequences of ciphertexts C' are $\mathcal{O}$-ciphertext sequences, and the remaining $n - \kappa_i$ are $\not{\mathcal{O}}$-ciphertext sequences. Party P_i now proves its integer v_i to be ranked κ_i^{th} integer by decrypting C' and proving correct decryption with $\mathsf{Proof}_{\mathsf{Decrypt}}$. Specifically,
Line 4: if $\kappa_i = k$, then P_i will prove in ZK that from the $n-1$ ciphertext sequences $C_{j,i,u=1,\ldots,\ell}$, encrypted for its public key, there are $k-1$ sequences which are $\mathcal{O}$-ciphertext sequences, and $n-k$ are $\not{\mathcal{O}}$-ciphertext sequences.
Line 11: if $\kappa_i < k$, then P_i will prove in ZK that there are $n-k+1$ sequences which are $\not{\mathcal{O}}$-ciphertext sequences.

// $\eta = (n-1) \cdot \log n - 1 - \frac{n-1}{2}$

1 **foreach** party $P_i, 1 \leq i \leq n$ **do**

2 $\{(C'_{1,i,1}, \ldots, C'_{1,i,\ell}), \ldots, (C'_{n-1,i,1}, \ldots, C'_{n-1,i,\ell}), \mathsf{Proof}_{\mathsf{Shuffle}^*}\} \leftarrow \mathsf{Benes}^*(\{(C_{j,i,1}, \ldots, C_{j,i,\ell})\}_{\forall j \neq i}, \{(R_{j,i,1}, \ldots, R_{j,i,\ell})\}_{\forall j \neq i})$;

3 publish $(C'_{1,i,1},\ldots,C'_{1,i,\ell}),\ldots,(C'_{n-1,i,1},\ldots,C'_{n-1,i,\ell})$, $\mathsf{Proof}_{\mathsf{Shuffle}^*}$ on blockchain;

// Let v_i be ranked as $\kappa_i^{\mathtt{th}}$ integer, let $(C'_{1,i,1},\ldots,C'_{1,i,\ell}),\ldots,(C'_{\kappa_i-1,i,1},\ldots,C'_{\kappa_i-1,i,\ell})$ be the $\mathcal{O}$-ciphertext sequences and $(C'_{\kappa_i,i,,1},\ldots,C'_{\kappa_i,i,\ell}),\ldots,(C'_{n-1,i,1},C'_{n-1,i,\ell})$ be the $\not{\mathcal{O}}$-ciphertext sequences

4 **if** $\kappa_i = k$ **then**

5 **for** $j = 1$ **to** $n-1$ **do**

6 **for** $u = 1$ **to** ℓ **do**

7 $C_{\mathsf{final},j,i,u} = sk_i \cdot C'_{j,i,u}[0]$;

8 publish $C_{\mathsf{final},j,i,u}, \mathsf{Proof}_{\mathsf{Decrypt},j,i,u}$ on blockchain;

9 **end**

10 **end**

11 **else if** $\kappa_i < k$ **then**

12 **for** $j = \kappa_i$ **to** $\kappa_i + n - k$ **do**

13 **for** $u = 1$ **to** ℓ **do**

14 $C_{\mathsf{final},j,i,u} = sk_i \cdot C'_{j,i,u}[0]$;

15 publish $C_{\mathsf{final},j,i,u}, \mathsf{Proof}_{\mathsf{Decrypt},j,i,u}$ on blockchain;

16 **end**

17 **end**

18 **else if** $\kappa_i > k$ **then**

19 **for** $j = 1$ **to** k **do**

20 **for** $u = 1$ **to** ℓ **do**

21 $C_{\mathsf{final},j,i,u} = sk_i \cdot C'_{j,i,u}[0]$;

22 publish $C_{\mathsf{final},j,i,u}, \mathsf{Proof}_{\mathsf{Decrypt},j,i,u}$ on blockchain;

23 **end**

24 **end**

25 **end**

26 **end**

Algorithm 4: BOREALIS' third round

Line 18: if $\kappa_i > k$, then P_i will prove in ZK that there are k sequences which are $\mathcal{O}$-ciphertext sequences.

Proving decryption is simply multiplying the left-hand side of an Elgamal ciphertext with the secret key. This allows anyone to derive the plaintext. As shown in Algorithm 4, P_i publishes proofs and shuffled ciphertexts on the blockchain. Again, all parties compute and publish on the blockchain in parallel within the same round.

4.4 Additional Techniques

4.4.1 Revealing Malicious Input and Optional Fourth Round

Malicious parties can produce invalid ZK proofs or simply abort protocol execution at any time. We now present how BOREALIS handles such malicious behavior and distinguish between two cases.

First case: A malicious party P_i aborts protocol execution during the first round, before publishing valid ZK proofs $\mathsf{Proof}_{\mathsf{VSS},i}$, $\mathsf{Proof}_{\mathsf{Bit},\{1,\ldots,\ell\}}$, and $\mathsf{Proof}_{\mathsf{Enc},\{1,\ldots,\ell\}}$ on the blockchain, or party P_i has published in the first round invalid ZK proofs $\mathsf{Proof}_{\mathsf{VSS},i}$, $\mathsf{Proof}_{\mathsf{Bit},\{1,\ldots,\ell\}}$, $\mathsf{Proof}_{\mathsf{Enc},\{1,\ldots,\ell\}}$. BOREALIS treats this case as if P_i would have submitted an invalid input integer $v_i = \perp$. Subsequently, in the second and third round, all parties will ignore P_i's input to the blockchain. The index of the k^{th}-ranked integer will be computed among all integers, excluding integer v_i.

Second case: A malicious party publishes valid proofs $\mathsf{Proof}_{\mathsf{VSS},i}$, $\mathsf{Proof}_{\mathsf{Bit},\{1,\ldots,\ell\}}$, and $\mathsf{Proof}_{\mathsf{Enc},\{1,\ldots,\ell\}}$ in the first round. This case is more subtle, because BOREALIS will now compute the index of the k^{th}-ranked integer including P_i's input v_j. The general idea is that, if a malicious P_i aborts or produces an invalid proof in one round, then the other parties will recover P_i's previously shared private key sk_i in the following round ([50], see Appendix A). Therewith, the other parties can decrypt v_i and compute the index of the k^{th}-ranked integer. More specifically:

P_i produces invalid proofs or aborts in the first round: The other parties recover sk_i and v_i in the second round and then re-run Algorithm 3 with P_i's input in the clear in the third round. Each party P_j publishes a DGK evaluation of their encrypted input with v_i, blinds evaluations, and shuffles encrypted bits. Party P_j publishes a decryption of DGK in case they need another witness to show that v_j is the k^{th} integer or greater or less than the k^{th} integer.

P_i publishes invalid proofs or aborts during the second round: The other parties reconstruct sk_i and learn v_i in the third round. Honest parties agree to run a fourth round where they compute DGK encrypted evaluations with v_i and open as described above. BOREALIS then concludes after a total of four rounds.

P_i publishes invalid proofs or aborts during the third round: Then, P_i has already published correct DGK evaluations in the second round. The other parties recover sk_i in the fourth round and decrypt P_i's DGK evaluations. Each party P_j knows for each other party $P_{j'}$ whether $v_{j'} < v_i$. Using this information, together with $P_{j'}$'s output from the third round, P_j can decide by itself whether $v_{j'}$ is the k^{th} integer or greater or less.

We stress that BOREALIS computes index ι of the k^{th} integer v_ι, even if v_ι is a malicious party's input and multiple malicious parties abort or publish invalid proofs. As all malicious parties' integers are revealed, these integers can be ordered, and they are compared to the other (honest) parties' input. So, the index of the k^{th} integer is always found.

4.4.2 Enforcing unique input integers

So far, we have assumed that for any pair of integers v_i and v_j, we have $v_i \neq v_j$. As a consequence, either $v_i < v_j$ or $v_j < v_i$, and our specific approach to prove the k^{th} element in Round 3 is correct. However for any pair of integers $v_i = v_j$, both P_i and P_j will get a $\not\oslash$-ciphertext sequence when comparing with each other. The additional $\not\oslash$-ciphertext sequence violates correctness of our approach of computing and proving the rank by counting the number of $\mathcal{O}$- and $\not\oslash$-ciphertext sequences. Party P_i will estimate v_i's rank κ_i off by one, denying computation of the k^{th}-ranked integer. To mitigate, we enforce that any two integers become different as follows. Any two public keys pk_i and pk_j are different with probability $1 - \mathsf{negl}(\lambda)$. If we interpret public keys as bit strings, we can order them lexicographically and thus each party P_i's public key pk_i is assigned a unique number ID_i from $\{1, \ldots, n\}$. The idea is now to extend each party's integer representation $v_i = v_{i,\ell} \ldots v_{i,1}$ by $\lceil \log n \rceil$ bits to $v_i = v_{i,\ell} \ldots v_{i,1} ID_{i,\lceil \log n \rceil} \ldots ID_{i,1}$, where the $\lceil \log n \rceil$ least significant bits are the bit representation of ID_i. This guarantees different input integers with overwhelming probability.

Note that it is not required to add complex ZK proofs to the first round, where parties prove that the least significant bits of their integer are indeed the ID. As we encrypt bitwise and IDs are publicly known, each party P_i agrees to encrypt the ID bits of their integer and compute corresponding Groth and Sahai commitments using fixed, publicly known random coins. This allows for automatic verification by all parties.

4.4.3 Optionally revealing integers

In addition to computing index ι of the k^{th}-ranked integer, parties can also compute integers, e.g. v_ι's actual value. Party P_ι will publish v_ι (or any other plaintext of a committed ciphertext) together with proofs $\mathsf{Proof}_{\mathsf{Decrypt}}$ of correctly decrypting the ciphertext $(c_{\iota,1,\ldots,\ell})$ at the end of Round 3, Algorithm 4.

5 Zero-Knowledge Tools

In the following, we present details about the various ZK proofs used within the paper. The framework by Groth and Sahai [36] allows proving multi-scalar equations (see Equation 1) in ZK. So, for each proof we want to provide, we reformulate all properties to prove as a set of such multi-scalar equations. We then prove each set of equations using the approach in [36]. If not stated differently, the output of our proofs below is simply the output of corresponding Groth and Sahai proofs of our equations.

5.1 Proving a Bit ($\mathsf{Proof}_{\mathsf{Bit}}$)

Party P proves that a previously committed integer $v \in \mathbb{Z}_p$ is either $0 \in \mathbb{Z}_p$ or $1 \in \mathbb{Z}_p$ by showing that $v \cdot (1 - v) = 0$, i.e., $v = v^2$. Using additively homomorphic

Elgamal encryption $c = E_{pk}(v)$, P will show plaintext equivalence of ciphertexts c and $c' = v \cdot c$ in ZK. However, P cannot simply multiply c with secret v and publish result c', as this would leak whether or not $v = 0$. Therefore, the idea is to randomize c' at the same time as multiplying it by v. P chooses random $r \xleftarrow{\$} \mathbb{Z}_p$ and computes $c'[0] = v \cdot c[0] + r \cdot \mathcal{P}_1, c'[1] = v \cdot c[1] + r \cdot pk_P$ for public key pk_P. The remaining plaintext equivalence proof is rather standard and proves an ECDLP, namely P proves that $sk_P \cdot (c'[0] - c[0]) = c'[1] - c[1]$ for private key sk_P.

So, applying the Groth and Sahai framework, P proves the following three multi-scalar equations over $\mathbb{G}_1$.

1. Correctness of $c'[0]$

   secret: $y_1 = v, y_2 = r$
   public: $\gamma_{1,1} = c[0], \gamma_{1,2} = \mathcal{P}_1, t = c'[0]$

2. Correctness of $c'[1]$

   secret: $y_1 = v, y_2 = r$
   public: $\gamma_{1,1} = c[1], \gamma_{1,2} = pk_P, t = c'[1]$

3. c, c' plaintext equivalence

   secret: $y = sk_P$
   public: $\gamma_1 = c'[0] - c[1], t = c'[1] - c[1]$

$\mathsf{Proof}_{\mathsf{Bit}}$ comprises the Groth and Sahai proofs for these three equations together with c' and commitment $\mathsf{Com}(r)$.

5.2 Proving Encryption ($\mathsf{Proof}_{\mathsf{ECDLP}}, \mathsf{Proof}_{\mathsf{Enc}}$)

Party P proves that previously committed input integer v (for example: a bit) and its public key matches ciphertext c. The following equations and all remaining ones in this section are Groth and Sahai's multi-scalar equations over $\mathbb{G}_1$.

First, P proves that their private key sk_P matches their public key pk_P. This is just a ZK proof of knowledge of exponent for ECDLP.
secret: $y = sk_P$ public: $\gamma_1 = \mathcal{P}_1, t = pk_P$
P now proves correctness of encryption.

1. Correctness of $E_{pk_P}(v)[0]$

   secret: $y = r$
   public: $\gamma_1 = \mathcal{P}_1, t = E_{pk_P}(v)[0]$

2. Correctness of $E_{pk_P}(v)[1]$

   secret: $y_1 = r, y_2 = v$
   public: $\gamma_{1,1} = pk_P, \gamma_{1,2} = \mathcal{P}_1, t = E_{pk_P}(v)[1]$

5.3 Proving DGK ($\mathsf{Proof}_{\mathsf{DGK}}$)

P_i proves that secret ciphertexts $c_1, \ldots, c_\ell$ are encrypting DGK with their secret input $v_{i,1}, \ldots, v_{i,\ell}$ and P_j's public ciphertexts $E_{pk_{P_j}}(v_{j,1}), \ldots, E_{pk_{P_j}}(v_{j,\ell})$. To prove DGK in the clear, remember that P_i would have to show for $u \in \{1, \ldots, \ell\}$ that $v_{i,u} - v_{j,u} + 1 + \sum_{\delta=u+1}^{\ell} v_{i,\delta} \oplus v_{j,\delta} = v_{i,u} - v_{j,u} + 1 + \sum_{\delta=u+1}^{\ell}(v_{i,\delta} + v_{j,\delta} - 2 \cdot v_{i,\delta} \cdot v_{j,\delta})$. In our Elgamal-encrypted domain, we therefore have:

$$\begin{aligned}
c_u[0] =& - E_{pk_{P_j}}(v_{j,u})[0] + \sum_{\delta=u+1}^{\ell} (E_{pk_{P_j}}(v_{j,\delta})[0] \\
& - 2 \cdot v_{i,\delta} \cdot E_{pk_{P_j}}(v_{j,\delta})[0]) \\
c_u[1] =& v_{i,u} \cdot \mathcal{P}_1 - E_{pk_{P_j}}(v_{j,u})[1] + \mathcal{P}_1 + \sum_{\delta=u+1}^{\ell} (v_{i,\delta} \cdot \mathcal{P}_1 \\
& + E_{pk_{P_j}}(v_{j,\delta})[1] - 2 \cdot v_{i,\delta} \cdot E_{pk_{P_j}}(v_{j,\delta})[1]).
\end{aligned}$$

We rearrange both equations and get

$$c_u[0] + \sum_{\delta=u+1}^{\ell} 2 \cdot v_{i,\delta} \cdot E_{pk_{P_j}}(v_{j,\delta})[0] = -E_{pk_{P_j}}(v_{j,u})[0] + \sum_{\delta=u+1}^{\ell} E_{pk_{P_j}}(v_{j,\delta})[0]$$

$$\begin{aligned}
c_u[1] - v_{i,u} \cdot \mathcal{P}_1 - \sum_{\delta=u+1}^{\ell} v_{i,\delta} \cdot \mathcal{P}_1 + \sum_{\delta=u+1}^{\ell} 2 \cdot v_{i,\delta} \cdot E_{pk_{P_j}}(v_{j,\delta})[1] \\
= -E_{pk_{P_j}}(v_{j,u})[1] + \mathcal{P}_1 + \sum_{\delta=u+1}^{\ell} E_{pk_{P_j}}(v_{j,\delta})[1].
\end{aligned}$$

Note that the right-hand sides contain only public information, while the left-hand sides contain secret information. We therefore derive Groth and Sahai's representation as follows:

1. Correctness of $c_u[0]$
   secret: $x = c_u[0], y_{u+1} = v_{i,u+1}, \ldots, y_\ell = v_{i,\ell}$
   public: $\gamma_{1,u+1} = 2 \cdot E_{pk_{P_j}}(v_{j,u+1})[0], \ldots, \gamma_{1,\ell} = 2 \cdot E_{pk_{P_j}}(v_{j,\ell})[0], \gamma_2 = 1, \Gamma = 0, t = -E_{pk_{P_j}}(v_{j,u})[0] + \sum_{\delta=u+1}^{\ell} E_{pk_{P_j}}(v_{j,\delta})[0]$

So, the multi-scalar equations are of type $\sum_{l=u+1}^{\ell} \gamma_{1,l} \cdot y_l + \gamma_2 \cdot x = t$.

2. Correctness of $c_u[1]$
   secret: $x = c_u[1], y_u = v_{i,u}, y'_{u+1} = v_{i,u+1}, \ldots, y'_\ell = v_{i,\ell}, y''_{u+1} = v_{i,u+1}, \ldots, y''_\ell = v_{i,\ell}$
   public: $\gamma_{1,u} = -\mathcal{P}_1, \gamma'_{1,u+1} = -\mathcal{P}_1, \ldots, \gamma'_{1,\ell} = -\mathcal{P}_1, \gamma''_{1,u+1} = 2 \cdot E_{pk_{P_j}}(v_{j,u+1})[1], \gamma''_{1,\ell} = 2 \cdot E_{pk_{P_j}}(v_{j,\ell})[1], \gamma_2 = 1, \Gamma = 0, t = -E_{pk_{P_j}}(v_{j,u})[1] + \mathcal{P}_1 + \sum_{\delta=u+1}^{\ell} E_{pk_{P_j}}(v_{j,\delta})[1]$

Here, multi-scalar equations are of type $\gamma_{1,u} \cdot y_u + \sum_{l=u+1}^{\ell} \gamma'_{1,l} \cdot y'_l + \sum_{l=u+1}^{\ell} \gamma''_l \cdot y'' + \gamma_2 \cdot x = t$.

5.4 Proving Permutation Networks ($\mathsf{Proof}_{\mathsf{Blind}}$, $\mathsf{Proof}_{\mathsf{Shuffle}}$, $\mathsf{Proof}_{\mathsf{Shuffle}^*}$)

A *crossbar switch* is a simple operator with two inputs i_1, i_2 and two outputs o_1, o_2. The switch either assigns output o_1 to i_1 and o_2 to i_2, or the other way around $o_1 = i_2$ and $o_2 = i_1$. Basically, the switch flips the input or not. Crossbar switches are building blocks for permuting larger input sequences.

To randomly permute an input of n elements, we construct a Beneš [10] permutation network PN_n out of crossbar switches. The idea to permute n elements is to recursively use 2 permutation networks $\mathsf{PN}_{\frac{n}{2}}$, each for $\frac{n}{2}$ elements. More specifically, the n elements of the input are grouped by two and input to $\frac{n}{2}$ crossbar switches. One output of each switch is routed to the first permutation network $\mathsf{PN}_{\frac{n}{2}}$, and the other output is routed to the second $\mathsf{PN}_{\frac{n}{2}}$ permutation network. The output of the two $\mathsf{PN}_{\frac{n}{2}}$ permutation networks is then connected to a final sequence of $\frac{n}{2}$ crossbar switches. That is, the outputs of the first $\mathsf{PN}_{\frac{n}{2}}$ permutation network are routed to the first inputs of the $\frac{n}{2}$ switches, and the outputs of the second $\mathsf{PN}_{\frac{n}{2}}$ permutation network are routed to the second inputs of switches. The recursion ends with PN_2 permutation networks which are again crossbar switches.

To permute an input sequence of n elements, a Beneš permutation network requires $n \cdot \log n - \frac{n}{2}$ crossbar switches.

5.4.1 Zero-Knowledge Proof Setup

Party P_i wants to prove in ZK correctness of an n element shuffle. Specifically in this paper, P_i has as an input a sequence of n additively homomorphic Elgamal ciphertexts $c_1, \ldots, c_n$ and outputs a re-encrypted shuffle $C_{\pi(1)}, \ldots, C_{\pi(n)}$. Note that in contrast to the typical scenario where both sequences of ciphertexts c_j and C_j are public, we are targeting the situation where inputs c are private, i.e., P_i has only published commitments $\mathsf{Com}(c_j)$ to them, and just the C are public.

P_i proves in two steps: first, P_i computes blinded versions c'_j, publishes commitments $\mathsf{Com}(c'_j)$ to them, and proves in ZK that the c'_j behind these commitments are blinded versions of the c_j.

The second step is then to randomly shuffle the c'_j and prove correctness of the shuffle by using a permutation network. The idea here is that for any n, the recursive layout of a permutation network is fixed. So, for a party P_i to prove correctness of an n element shuffle in ZK, it is sufficient to prove correctness of all $n \cdot \log n - \frac{n}{2}$ internal crossbar switches.

5.4.2 Proving Blinding ($\mathsf{Proof}_{\mathsf{Blind}}$)

First, we will use the previously introduced *blinding* of ciphertexts instead of re-encryption. Otherwise, decryption will leak details about the DGK evaluation.

Let c' be a blinded additively homomorphic Elgamal ciphertext of c computed as above. Commitments $\mathsf{Com}(c), \mathsf{Com}(c')$ for ciphertexts and a commitment $\mathsf{Com}(R)$ for a random string R have been published. P_i proves in ZK that ciphertext c' is a blinded version of ciphertext c_1. The Groth and Sahai representation is:

1. Correctness of $c'[0]$
   secret: $x_1 = c'[0], x_2 = c[0], y = R$,
   public: $\gamma_1 = \mathcal{O}, \gamma_{2,1} = 1, \gamma_{2,2} = 0, \Gamma = \left(\begin{smallmatrix} 0 & -1 \\ 0 & 0 \end{smallmatrix}\right), t = \mathcal{O}$

2. Correctness of $c'[1]$
   secret: $x_1 = c'[1], x_2 = c[1], y = R$,
   public: $\gamma_1 = \mathcal{O}, \gamma_{2,1} = 1, \gamma_{2,2} = 0, \Gamma = \left(\begin{smallmatrix} 0 & -1 \\ 0 & 0 \end{smallmatrix}\right), t = \mathcal{O}$

Note that Γ is non-zero, so our multi-scalar equations combine secret elements from $\mathbb{G}_1$ and $\mathbb{Z}_p$. Party P_i also has to prove that $R \neq 0$, but this is straightforward: for another (encrypted) random $\hat{R}$, P_i reveals $R \cdot \hat{R}$, which has to be $\neq 0$, and proves correctness of this product with a $\mathsf{Proof}_{\mathsf{ECDLP}}$.

5.4.3 Proving a Crossbar Switch ($\mathsf{Proof}_{\mathsf{Shuffle}}$)

P_i constructs a single ZK crossbar switch with inputs i_1, i_2 and outputs o_1, o_2. In our case, the inputs are two additively homomorphic Elgamal ciphertexts c'_1, c'_2 which the switch will randomly permute and then output as C_1, C_2. Using the Groth and Sahai framework, P_i will prove in ZK that the two output ciphertexts C_1, C_2 are a permutation of input c'_1, c'_2. As output C_1, C_2 of one crossbar switch serves as input for two other crossbar switches in a permutation network, neither c'_1, c'_2 (output of blinding above) nor C_1, C_2 are public. Instead, P_i only publishes commitments to them. Only the last sequence of crossbar switches reveals output ciphertexts which another party P_j can finally decrypt.

We realize a crossbar switch by flipping input depending on secret random bit $\beta \in \{0, 1\}$. To be able to use β, P_i first publishes commitment $\mathsf{Com}(\beta)$ and proves that $\beta \in \{0, 1\}$, see Section 5.1. Specifically, P_i proves that C_1, C_2 is a permutation of secret c'_1, c'_2:

$$\begin{aligned} C_1[0] &= \beta \cdot c'_1[0] + (1-\beta) \cdot c'_2[0] \\ C_1[1] &= \beta \cdot c'_1[1] + (1-\beta) \cdot c'_2[1] \\ C_2[0] &= (1-\beta) \cdot c'_1[0] + \beta \cdot c'_2[0] \\ C_2[1] &= (1-\beta) \cdot c'_1[1] + \beta \cdot c'_2[1] \end{aligned}$$

Observe for this shuffle trick $C_1 = \beta \cdot c'_1 + (1-\beta) \cdot c'_2 = \beta \cdot c'_1 + c'_2 - \beta \cdot c'_2$. Therewith, we can now derive the Groth and Sahai representation:

1. Correctness of $C_1[0]$
   secret: $x_1 = c'_1[0], x_2 = c'_2[0], x_3 = C_1[0], y_1 = \beta$
   public: $\gamma_1 = \mathcal{O}, \gamma_{2,1} = 0, \gamma_{2,2} = 1, \gamma_{2,3} = -1, \Gamma = \left(\begin{smallmatrix} 1 & -1 & 0 \end{smallmatrix}\right), t = \mathcal{O}$

2. Correctness of $C_1[1]$
   secret: $x_1 = c'_1[1], x_2 = c'_2[1], x_3 = C_1[1], y_1 = \beta$
   public: $\gamma_1 = \mathcal{O}, \gamma_{2,1} = 0, \gamma_{2,2} = 1, \gamma_{2,3} = -1, \Gamma = \left(\begin{smallmatrix} 1 & -1 & 0 \end{smallmatrix}\right), t = \mathcal{O}$

3. Correctness of $C_2[0]$
secret: $x_1 = c'_1[0], x_2 = c'_2[0], x_3 = C_2[0], y_1 = \beta$,
public: $\gamma_1 = \mathcal{O}, \gamma_{2,1} = 1, \gamma_{2,2} = 0, \gamma_{2,3} = -1, \Gamma = (\begin{smallmatrix} -1 & 1 & 0 \end{smallmatrix}), t = \mathcal{O}$

4. Correctness of $C_2[1]$
secret: $x_1 = c'_1[1], x_2 = c'_2[1], x_3 = C_2[1], y_1 = \beta$,
public: $\gamma_1 = \mathcal{O}, \gamma_{2,1} = 1, \gamma_{2,2} = 0, \gamma_{2,3} = -1, \Gamma = (\begin{smallmatrix} -1 & 1 & 0 \end{smallmatrix}), t = \mathcal{O}$

Equations above are for proving a single crossbar switch. Proof $\mathsf{Proof}_{\mathsf{Shuffle}}$ for n ciphertexts is simply the concatenation of proofs of the $n \cdot \log n - \frac{n}{2}$ crossbar switches in the permutation network. The last sequence of crossbar switches in the permutation network outputs shuffled ciphertexts: in the equations above, there will be no secret $x_3 = C$, but instead public t will be C.

Function Benes we use in Section 4.3 to shuffle ciphertexts of v_i's bits outputs both shuffled ciphertexts C_j and the proof of shuffle for the whole permutation ($n \cdot \log n - \frac{n}{2}$ proofs of crossbar switches).

5.4.4 Proving a Shuffle of Ciphertext Sequences ($\mathsf{Proof}_{\mathsf{Shuffle}^*}$)

With function Benes, we generate and prove a shuffle of ℓ ciphertexts which are encryptions of bits $v_{i,j}$. We extend the idea behind this proof to also prove the shuffle of $n-1$ DGK evaluations in Round 3 (Algorithm 4). With this shuffle (called $\mathsf{Shuffle}^*$), P_i shuffles sequence $\mathcal{C} = \{(C_{j,i,1}, \ldots, C_{j,i,\ell})\}_{\forall j \neq i}$. Sequence $\mathcal{C}$ comprises $n-1$ elements which are sub-sequences, each of ℓ ciphertexts. The shuffle shuffles *both* indices j *and* positions of encrypted bits for each $C_{j,i,u}$. So, $\mathsf{Shuffle}^*$ outputs $\mathcal{C}' = \{(C_{\pi(j),i,\pi'_j(1)}, \ldots, C_{\pi(j),i,\pi'_j(\ell)})\}_{\forall j \neq i}$ for randomly chosen permutations π and π'_j.

To shuffle $\mathcal{C}$ using a Beneš permutation network, our idea is, first, to treat a sequence of ℓ ciphertexts $(C_{j,i,1}, \ldots, C_{j,i,\ell})$ as a single input to a crossbar switch. Again, each crossbar switch will flip its two inputs, two sequences of ℓ ciphertexts each, depending on a single random bit β as above. If $\beta = 0$, the switch swaps the two sequences. Before the actual shuffle, we need to blind each ciphertext $C_{j,i,u}$ by multiplying both sides of the ciphertext with a random $R_{j,i,u} \xleftarrow{\$} \mathbb{Z}_p$.

Function Benes^* takes as an input sequence $\mathcal{C}$ and outputs blinded, shuffled sequence $\mathcal{C}'$ together with a ZK proof of correctness $\mathsf{Proof}_{\mathsf{Shuffle}^*}$. We construct $\mathsf{Proof}_{\mathsf{Shuffle}^*}$ as a simple concatenation of proofs $\mathsf{Proof}_{\mathsf{Blind}}$ and then $(n-1) \cdot \log(n-1) - \frac{n-1}{2}$ $\mathsf{Proof}_{\mathsf{Shuffle}}$ for the individual crossbar switches of the permutation network. Therewith, we realize and prove correctness of the first permutation π and blinding of all ciphertexts. We then also shuffle and prove positions of encrypted bits within sequences $(C_{\pi(j),i,1}, \ldots, C_{\pi(j),i,\ell})$ using $n-1$ different Beneš permutation networks of ℓ inputs, each. Thus, we generate and prove correctness of permutations π'_j. The output of function Benes^* includes the $n-1$ proofs of an ℓ input Beneš permutation network for that, too.

To enable verification of $\mathsf{Proof}_{\mathsf{Shuffle}^*}$, note that we need to publish commitments to all β and all random $R_{j,i,u}$ on the blockchain. We also prove on the blockchain that the β are bits using $\mathsf{Proof}_{\mathsf{Bit}}$ and prove correct blinding using $\mathsf{Proof}_{\mathsf{Blind}}$. To help readability, we have omitted repeating details of these proofs in Algorithm 4.

5.4.5 Arbitrary n

While the above standard Beneš permutation networks require the input set size to be a power of 2, there exist extensions for arbitrary sizes. They are efficient and require only up to $\lfloor n \cdot \log n - \frac{n}{2} \rfloor$ crossbar switches [22]. So, we can shuffle without putting constraints on bit length ℓ or the number of parties n.

5.5 Proving Decryption $\mathsf{Proof}_{\mathsf{Decrypt}}$

P_i proves that a ciphertext $C_{\text{final}} = C[0] \cdot sk_i$ for the left-hand side of another ciphertext C and its secret key sk_i. Again, this is just a Groth and Sahai proof of knowledge of exponent for ECDLP.

1. C_{final} secret: $y = sk_i$ public: $\gamma_1 = C[0], t = C_{\text{final}}$

6 Security Analysis

We prove Theorem 1 using a simulation-based proof in the hybrid model [42]. In the hybrid model, simulator $\mathcal{S}$ generates messages of honest parties interacting with malicious parties and the trusted third party TTP, but we treat ZK proofs ([36, 50]) as oracle functionalities. Simulator $\mathcal{S}$ does not use inputs of honest parties (except for forwarding to the TTP which does not leak any information), so the protocol does not reveal any information except the result, i.e., the output of the TTP. Messages generated by $\mathcal{S}$ must be indistinguishable from messages in the real execution of BOREALIS.

Proof of Theorem 1. Let P be the set of all Parties and $\overline{P}$ be the parties controlled by adversary $\mathcal{A}$. We prove

$$IDEAL_{\mathcal{F}_{k^{\text{th}}\text{-Ideal}},\mathcal{S}}(v_1, \ldots, v_n) \equiv REAL_{\Pi_{\mathsf{BOREALIS}},\mathcal{A}}(v_1, \ldots, v_n).$$

I) In the first round of the protocol, malicious parties $P_{\bar{i}}$ commit to their input, including their public key $pk_{\bar{i}}$, an encryption of $E_{pk_{\bar{i}}}(v_{\bar{i},j})$ and ZK proofs of proper integer encryption ($\mathsf{Proof}_{\mathsf{Bit}}$ and $\mathsf{Proof}_{\mathsf{Enc}}$) and correct VSS shares $\mathcal{Y}$ $\mathsf{Proof}_{\mathsf{VSS}}$. If verification of either of the ZK proofs $\mathsf{Proof}_{\mathsf{Bit}}, \mathsf{Proof}_{\mathsf{Enc}}, \mathsf{Proof}_{\mathsf{VSS}}$ fails, we treat the value $v_{\bar{i}} = \perp$, since it is non-recoverable by honest parties and exclude $P_{\bar{i}}$ from further participation in the protocol. If the verification succeeds, $\mathcal{S}$ extracts $v_{\bar{i}}$ from the oracle functionality of the ZK proof $\mathsf{Proof}_{\mathsf{Enc}}$ and sends it to the TTP. Since $\mathcal{S}$ forwards the honest parties' input to TTP, it receives the output $OUTPUT$ of $\mathcal{F}_{k^{\text{th}}\text{-Ideal}}$ from the TTP. If the verification of any further ZK proof, in this or a subsequent step, fails, we invoke an input recovery protocol for $v_{\bar{i}}$ using VSS shares $\mathcal{Y}$. $\mathcal{S}$ can simulate messages from honest parties, since they are either semantically-secure ciphertexts under honest parties' keys, computationally-hiding commitments or ZK proofs.

II) In the second round, malicious parties $P_{\bar{i}}$ need to prove correctness of their messages in ZK. As before, we invoke an input recovery protocol on failure.

$\mathcal{S}$ simulates messages from the honest parties as follows. The ciphertexts under the malicious parties' keys are simulated with random plaintexts which match comparison results from $OUTPUT$; they are the comparison results between $v_{\bar{i}}$ and v_j from the honest parties encrypted under the malicious parties' public keys $pk_{\bar{i}}$. The remainder are ZK proofs or computationally-hiding commitments.

III) In the third round, malicious parties $P_{\bar{i}}$ need to again prove the correctness of their messages in ZK. $\mathcal{S}$ simulates messages from the honest parties, since revealed plaintext information, ι, and the comparison of v_j from the honest parties to v_ι can be derived from $OUTPUT$. $\mathcal{S}$ then simulates the corresponding ZK proofs and computationally-hiding commitments. □

7 Evaluation

In each round of BOREALIS, each party's computation time is dominated by preparing and publishing Groth and Sahai ZK proofs. To indicate BOREALIS' practicality in real-world scenarios, we have therefore implemented and benchmarked the ZK proofs of this paper. The source code is available for download at [17]. In the following, our goal is estimating for up to which number of suppliers n and bit length ℓ BOREALIS is practical. That is, for which values of n, ℓ each party's total computation time is below Bitcoin's or Ethereum's block intervals.

All proofs are implemented on top of Bazin [7]'s general framework for Groth and Sahai proofs. For its underlying cryptographic primitives, this framework employs the MIRACL library [46]. Our benchmarks were run with Fp254BNb, i.e., a standard 128 bit security Barreto-Naehrig Type-3 elliptic curve, Ate pairing, and SHA256 as hash function. Being a Type-3 curve, the SXDH assumption holds. Benchmarks were performed on a Linux laptop with 2.20 GHz Intel i7-6560U CPU. Table 1, which we have moved to Appendix A due to space constraints, summarizes benchmark results. For each ZK proof, we measure proof computation time, averaged over 100 runs, and total proof size. Note that our CPU features 4 cores, so we can independently compute 4 ZK proofs at the same time. Also note that total time and size of a Groth and Sahai system of equations is linear in the number of equations and variables (cf. Figure 3 in [36]). So, total time and size for each party in each round is a simple linear combination of the individual proofs from Table 1.

First round: P_i computes one $\mathsf{Proof}_{\mathsf{ECDLP}}$ for their private key, ℓ $\mathsf{Proof}_{\mathsf{Bit}}$ for its own input integer v_i, $(n-1) \cdot (\ell \cdot \log \ell - \frac{\ell}{2})$ $\mathsf{Proof}_{\mathsf{Bit}}$ for the β, and ℓ $\mathsf{Proof}_{\mathsf{Enc}}$ to prove correct encryption. Our variation of Schoenmakers's $\mathsf{Proof}_{\mathsf{VSS}}$ corresponds to one $\mathsf{Proof}_{\mathsf{ECDLP}}$.

Second round: P_i computes $\ell \cdot (n-1)$ $\mathsf{Proof}_{\mathsf{DGK}}$. Yet, computation time of $\mathsf{Proof}_{\mathsf{DGK}}$ itself increases linearly in ℓ. Table 1 shows computation time for a 2 bit proof, so we have to multiply this computation time by $\frac{l}{2}$ to estimate time for arbitrary ℓ. So, P_i is busy computing $\frac{\ell^2}{2} \cdot (n-1)$ times $\mathsf{Proof}_{\mathsf{DGK}}$ of Table 1. In addition, P_i computes $(n-1) \cdot (\ell \cdot \log \ell - \frac{\ell}{2})$ $\mathsf{Proof}_{\mathsf{Shuffle}}$.

Third round: Here, P_i computes a single $\mathsf{Proof}_{\mathsf{Shuffle}^*}$. This comprises $\ell \cdot (n-1) \cdot (\log(n-1) - \frac{n-1}{2})$ $\mathsf{Proof}_{\mathsf{Shuffle}}$ to shuffle all length ℓ ciphertext sequences, $(n-1) \cdot \log(n-1) - \frac{n-1}{2}$ $\mathsf{Proof}_{\mathsf{Bit}}$ for the individual crossbar switches, $(n-1) \cdot \ell$ $\mathsf{Proof}_{\mathsf{Blind}}$ to blind all ciphertexts, and then $(n-1) \cdot (\ell \cdot \log \ell - \frac{\ell}{2})$ $\mathsf{Proof}_{\mathsf{Shuffle}}$ plus $(n-1) \cdot (\ell \cdot \log \ell - \frac{\ell}{2})$ $\mathsf{Proof}_{\mathsf{Bit}}$ for the $n-1$ permutation networks. Finally, P_i computes up to ℓ $\mathsf{Proof}_{\mathsf{Decrypt}}$.

Due to $\mathsf{Proof}_{\mathsf{DGK}}$ and $\mathsf{Proof}_{\mathsf{Shuffle}^*}$, computation times in the second and third rounds are significantly higher than in the first round. As $\mathsf{Proof}_{\mathsf{DGK}}$ computation time is quadratic in ℓ, either the second or the third round take longest. Therefore, Figure 1 depicts the maximum time of these two rounds for various combinations of the number of parties n and typical bit lengths ℓ. Both axes are scaled logarithmically. The figure also shows block interval times for Ethereum (≈ 15 s [30]) and Bitcoin (≈ 10 min [13]).

In scenarios with low resolution integers ($\ell = 8$), our prototypical, non-optimized implementation of BOREALIS is very practical, supporting several hundreds of parties ($n \approx 800$) with Bitcoin, and $n \approx 30$ parties with Ethereum. Even in the other extreme with fine-grained, high precision integers ($\ell = 32$), BOREALIS remains practical and copes with ≈ 200 parties for Bitcoin. Only in the worst-case situation with $\ell = 32$ bit and Ethereum's low block interval times, our implementation is practical for only a small number of parties, e.g., n up to 8.

8 Conclusion

In this paper, we have built BOREALIS, a generic, efficient building block for sealed bid auctions on blockchains. We perform secure comparisons of integer bids and enable ZK proofs of the k^{th}-rank of an integer among n parties. BOREALIS completes in a constant number of only 3 (in case of malicious behavior 4) rounds. Moreover, its computational requirements suit several dozens of parties and large integers. Such a low number of rounds permits running more complex auctions on blockchains. We achieve these properties by carefully engineering several cryptographic primitives, like ECC Elgamal, homomorphic comparisons, and Groth and Sahai proofs.

Acknowledgments: This work was partially supported by NSERC grants RGPIN-05849, RGPAS-507908, and DGDND-00085.

References

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k^{th}-ranked element. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 40–55, 2004.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *International Conference on Database Theory*, pages 246–258, 2005.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. An $o(n \log n)$ sorting network. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 1–9, 1983.

[4] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 443–458, 2014.

[5] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016.

[6] Auctionity. The world's largest blockchain auction house for NFTs, 2019. `https://www.auctionity.com/`.

[7] R. Bazin. Efficient implementation of the Non-Interactive Zero-Knowledge proofs invented by Groth and Sahai , 2016. `https://github.com/baz1/GS-NIZK`.

[8] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, 1988.

[10] V.E. Beneš. Optimal rearrangeable multistage connecting networks. *The Bell System Technical Journal*, 43(4):1641–1656, 1964. ISSN 0005-8580.

[11] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *International Cryptology Conference*, pages 421–439, 2014.

[12] I. Bentov, R. Kumaresan, and A. Miller. Instantaneous decentralized poker. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 410–440. Springer, 2017.

[13] Bitinfocharts. Bitcoin (BTC) price stats and information, 2019. `https://bitinfocharts.com/bitcoin/`.

[14] E.-O. Blass and F. Kerschbaum. Strain: A secure auction for blockchains. In *European Symposium on Research in Computer Security*, pages 87–110, 2018.

[15] P. Bogetoft, D.L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J.D. Nielsen, J.B. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[16] F. Bonchi, C. Castillo, A. Gionis, and A. Jaimes. Social network analysis and mining for business applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):22, 2011.

[17] BOREALIS. Source code, 2019. `https://github.com/e-o-blass/BOREALIS-src`.

[18] F. Brandt. Fully Private Auctions in a Constant Number of Rounds. In *Proceedings of the 7th International Conference on Financial Cryptography, FC 2003*, pages 223–238, 2003.

[19] F. Brandt. Auctions. In Burton Rosenberg, editor, *Handbook of Financial Cryptography and Security.*, pages 49–58. Chapman and Hall/CRC, 2010.

[20] C. Cachin. Efficient Private Bidding and Auctions with an Oblivious Third Party. In *Conference on Computer and Communications Security, Singapore*, pages 120–127, 1999.

[21] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *International Conference on Security and Cryptography for Networks*, pages 182–199. Springer, 2010.

[22] C. Chang and R.G. Melhem. Arbitrary Size Benes Networks. *Parallel Processing Letters*, 7(3):279–284, 1997.

[23] A.R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 719–728, 2017.

[24] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *Information Security and Privacy, 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, pages 416–430, 2007.

[25] I. Damgård, M. Geisler, and M. Krøigaard. A correction to 'efficient and secure comparison for on-line auctions'. *IJACT*, 1(4):323–324, 2009.

[26] I. Damgård, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pages 285–304. Springer, 2006.

[27] I. Damgård, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662, 2012.

[28] J. Dreier, J.-G. Dumas, and P. Lafourcade. Brandt's fully private auction protocol revisited. *Journal of Computer Security*, 23(5):587–610, 2015.

[29] F. Emekçi, O.D. Sahin, D. Agrawal, and A. El Abbadi. Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering*, 63(2):348–361, 2007.

[30] Etherscan. Ethereum Average BlockTime Chart, 2019. `https://etherscan.io/chart/blocktime`.

[31] M. Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *Cryptographers' Track at RSA Conference*, pages 457–471, 2001.

[32] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342, 2007.

[33] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, 2015.

[34] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[35] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, 1987.

[36] J. Groth and A. Sahai. Efficient Non-interactive Proof Systems for Bilinear Groups. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 415–432, 2008. The full version of the paper is available at `http://eprint.iacr.org/2007/155`.

[37] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *International Conference on Cryptology and Network Security*, pages 1–20. Springer, 2009.

[38] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858, 2016.

[39] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 30–41, 2014.

[40] R. Kumaresan, V. Vaikuntanathan, and P.N. Vasudevan. Improvements to secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 406–417, 2016.

[41] Y. Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. 2005.

[42] Y. Lindell. How To Simulate It – A Tutorial on the Simulation Proof Technique. Cryptology ePrint Archive, Report 2016/046, 2016. `http://eprint.iacr.org/2016/046`.

[43] Y. Lindell, B. Pinkas, N.P. Smart, and A. Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Annual Cryptology Conference*, pages 319–338, 2015.

[44] Y. Lindell, N.P. Smart, and E. Soria-Vazquez. More Efficient Constant-Round Multi-party Computation from BMR and SHE. In *Theory of Cryptography*, pages 554–581, 2016.

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

[46] MIRACL. Cryptographic SDK, 2018. `https://github.com/miracl/MIRACL`.

[47] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[48] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *International Workshop on Public Key Cryptography*, pages 343–360, 2007.

[49] Reuters. Ukrainian ministry carries out first blockchain transactions, 2017. `https://www.reuters.com/article/us-ukraine-blockchain/ukrainian-ministry-carries-out-first-blockchain-transactions-idUSKCN1BH2ME`.

[50] B. Schoenmakers. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 148–164, 1999.

[51] K. Suzuki and M. Yokoo. Secure generalized vickrey auction using homomorphic encryption. In *International Conference on Financial Cryptography*, pages 239–249. Springer, 2003.

[52] J. Vaidya and C. Clifton. Privacy-preserving top-k queries. In *International Conference on Data Engineering*, pages 545–546, 2005.

[53] M. Vukolic. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In *Open Problems in Network Security*, pages 112–125, 2015.

[54] G. Zyskind, O. Nathan, and A. Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

A Verifiable Secret Sharing

We briefly summarize our variation of Schoenmakers [50]'s scheme for verifiable secret sharing. Let P_i be the dealer, the party which wants to verifiably share a fresh, randomly generated private key $sk_i \in \mathbb{Z}_p$. The core modification to Schoenmakers's scheme is that each party P_j receives an Elgamal encrypted version of their share in addition to commitments as follows.

Distribution: Let $\mathcal{P}_1$ be the generator of our group $\mathbb{G}_1$ in which the DDH holds. P_1 randomly selects $sk_i \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ as (session) private key and computes public key $pk_i = sk_i \cdot \mathcal{P}_1$. Moreover, P_i computes a degree $\frac{n}{2} - 1$ polynomial $f(x) = \sum_{j=0}^{\frac{n}{2}-1} \alpha_j \cdot x^j$ with $\alpha_0 = sk_i$ and all other coefficients α_j random from $\mathbb{Z}_p$.

Then, P_i publishes on the blockchain: pk_i, commitments to all of f's coefficients $C_u = \alpha_u \cdot \mathcal{P}_1, 0 \leq u \leq \frac{n}{2} - 1$, and a ZK proof $\mathsf{Proof}_{\mathsf{ECDLP}}$ that sk_i is indeed the DLOG of pk_i. For each party P_j, P_i also selects another $r_j \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and publishes Elgamal encryption $Y_j = E_{pk_j^{\mathsf{lt}}}(f(j)) = (r_j \cdot \mathcal{P}_1, r_j \cdot pk_j^{\mathsf{lt}} \oplus f(j))$ on the blockchain.

To verify its share, each party P_j decrypts Y_j and gets $f(j)$. Now, each P_j computes $\sum_{u=0}^{\frac{n}{2}-1} j^u \cdot C_u$ and checks whether this equals $f(j) \cdot \mathcal{P}_1$. If the check fails, P_j publishes $f(j)$ and $sk_j^{\mathsf{lt}} \cdot Y_j[0]$ on the blockchain together with a $\mathsf{Proof}_{\mathsf{ECDLP}}$ to prove correct multiplication and therewith decryption. If this proof is correct, P_j ignores P_i for the rest of the protocol. If P_j's proof of correct decryption is wrong, all parties exclude P_j (and they could try recovering sk_j). If P_i's initial $\mathsf{Proof}_{\mathsf{ECDLP}}$ is wrong, P_i is excluded, too.

Reconstruction: In case a party P_i's key has to be recovered, all other parties P_j publish their share $f(j)$ together with $sk_j^{\mathsf{lt}} \cdot Y_j[0]$ and $\mathsf{Proof}_{\mathsf{ECDLP}}$ to prove correct decryption. Honest parties use correct $f(j)$s to interpolate f and finally compute P_i's secret key $sk_i = f(0)$. As we assume a honest majority of at least $\frac{n}{2}$ honest parties, they will be able to interpolate degree $\frac{n}{2} - 1$ polynomial f.

Table 1: Resources per party

Proof Type	Time (ms)	Size (Byte)	Number of invocations per party
$\mathsf{Proof}_{\mathsf{Bit}}$	5.90	489	$O(n \cdot (l \cdot \log l + \log n))$
$\mathsf{Proof}_{\mathsf{ECDLP}}$	1.78	163	$O(1)$
$\mathsf{Proof}_{\mathsf{Enc}}$	3.92	326	$O(\ell)$
$\mathsf{Proof}_{\mathsf{DGK}}$ per bit	8.24	914	$O(n \cdot \ell^2)$
$\mathsf{Proof}_{\mathsf{Blind}}$ per bit	11.0	786	$O(n \cdot \ell)$
$\mathsf{Proof}_{\mathsf{Shuffle}}$ per two ciphertexts	29.4	1308	$O(n \cdot \ell \cdot (\log \ell + \log n))$
$\mathsf{Proof}_{\mathsf{Decrypt}}$	1.78	196	$O(\ell)$

Following our notation in Algorithm 2, VSS outputs sk_i, encryptions $\mathcal{Y}_j$, and $\mathsf{Proof}_{\mathsf{VSS}}$ which is a $\mathsf{Proof}_{\mathsf{ECDLP}}$.

Our modification to [50] allows to share an element of $\mathbb{Z}_p$ instead of $\mathbb{G}_1$. At the same time, our scheme loses the property of public verifiability. That is, one party cannot automatically verify whether the dealer's output to another party is valid or not. However in our specific scenario, this is acceptable.

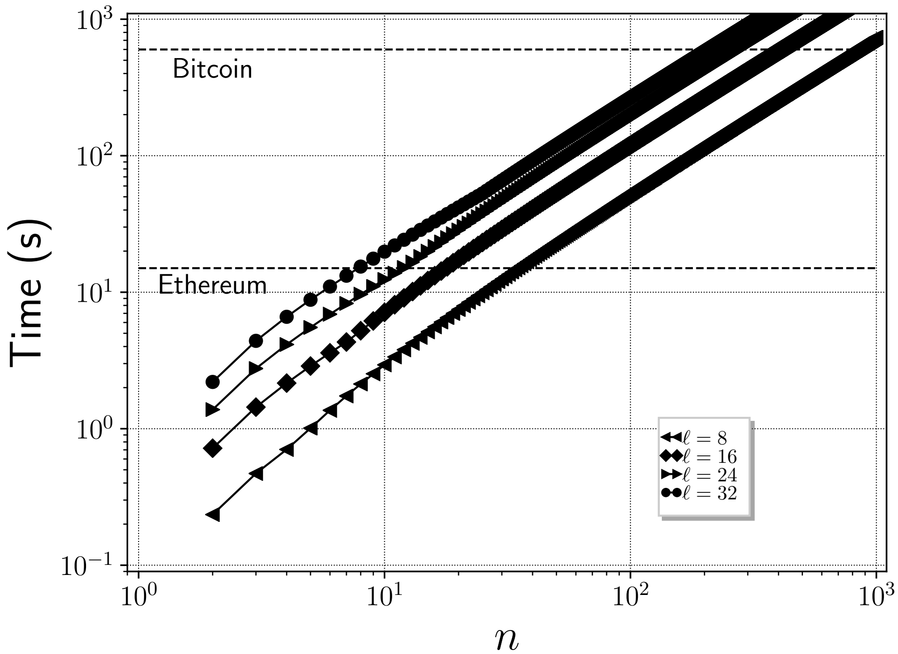

Figure 1: Maximum round computation time