

Generic Constant-Round Oblivious Sorting Algorithm for MPC

Bingsheng Zhang

University of Tartu, Estonia
zhang@ut.ee

Abstract Various information-theoretically secure Multi-Party Computation (MPC) schemes have been proposed over some finite field \mathbb{F} or some finite ring \mathbb{R} . A function f that can be evaluated on MPC is usually represented by boolean or arithmetic circuits. In general, the function class that have constant-depth arithmetic circuit is studied. Additionally, some literatures show that one can represent any formulas and branching program by low-degree randomizing polynomials, which can be evaluated in constant rounds. However, these approaches have their limitations, and it is not easy to construct the optimal branching program for a complex function. Therefore, it is not obvious how to efficiently perform oblivious sort in constant rounds, but oblivious sort is one of the most important primitive protocols for MPC in practice. In this paper, we are going to show several constant-round 0-error oblivious sorting algorithms, together with some useful applications.

Keywords. Secure Multi-party Computation, Constant-round, Oblivious Sort, Information-theoretic Security

1 Introduction

In a secure Multi-Party Computation (MPC), n players want to compute an agreed function of their inputs in a secure way such that it guarantees the correctness of the output and the privacy of the players' inputs. Some players (parties) are dishonest or malicious. Let x_1, x_2, \dots, x_n and y_1, y_2, \dots, y_n to be the corresponding inputs and outputs of parties $\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_n$, respectively. A (randomized) function is defined as $f(x_1, x_2, \dots, x_n; [r]) = (y_1, y_2, \dots, y_n)$, where r is a uniformly random value unknown to all parties. Yao's millionaires' problem [Yao82] is a famous problem that used to introduce secure two- or multi-party computation and Secure Function Evaluation (SFE). Many kinds of solutions are proposed to solve these problems, such as Yao's garble circuit [Yao82]. A wide range of literatures offered solutions based on homomorphic encryption schemes, and a new type of crypto-computing method that based on Branching Program (BP) is introduced in 2007 [IP07]. Recently, Fully-Homomorphic Encryption (FHE) schemes enjoy their popularities, such as Gentry's FHE based on lattice [Gen09]. However, they are rarely used in practice

due to efficiency reasons. On the other hand, practical implementations of secure MPC have been developing recently [DGKN09,BLW08].

A typical MPC scheme, e.g.[RBO89], is based on (verifiable) linear secret sharing, and n parties have shared values x_1, x_2, \dots, x_n over some finite field \mathbb{F} or some finite ring \mathbb{R} . Without loss of generality, denote $\mathbb{K} = \mathbb{Z}_N$, then $f : \mathbb{K}^n \rightarrow \mathbb{K}^n$. The function f can be computed securely on the shares via a network with pairwise secure channels. Particularly, we are interested in constant-round information-theoretically secure MPC. In terms of efficiency, round complexity¹ is widely used for MPC performance evaluation, because the main costs for MPC algorithms are network delays, especially via asynchronous Internet. It is an open question to determine which functions can be efficiently computed with information-theoretic security on shares, using constant-round protocols. In general, the function class that have constant-depth arithmetic circuit is studied. In paper [CFIK03,IK02], the authors showed that one can represent any formulas and branching program by low-degree randomizing polynomials over some finite field \mathbb{F} or some finite ring \mathbb{R} , which can be evaluated in constant rounds. However, these approaches have their limitations and it is not easy to construct the optimal branching program for a complex function. On the other hand, papers [DFNT05,DFK⁺06] show how to do equality-check, comparison, bit-decomposition and unbounded fan-In symmetric functions in constant-round over finite field \mathbb{F} . But it is not obvious how to efficiently perform oblivious sort in constant rounds by using these elementary protocols. Nowadays, most oblivious sorting protocols use compare-swap block together with oblivious sorting networks, such as AKS sorting network [AKS83]. Very few literatures [CKMH07] proposed specifically designed oblivious sorting algorithm or comparator network for MPC. In this paper, we are going to propose several information-theoretically secure constant-round oblivious sorting algorithms for MPC, together with their applications in secure $(M + 1)$ st-price sealed-bid auction and obfuscated shuffle..

Sealed-bid auction is getting increasingly popular recently, for it preserves the privacy of losing bids after auction closes. It is proved that $(M + 1)$ st-price sealed-bid auction satisfies incentive compatibility, i.e. the dominant strategy is for a bidder to bid his/her true valuation [WWW98]. Assume there is a set of L single-unit bids, in which M are sell offers and $N = L - M$ are buy offers. $(M + 1)$ st-price auction rule says that the winner pays the $(M + 1)$ -th bid. Note that the $(M + 1)$ -th price is undefined if there is no buyer. For the case that $M = 1$, it is the same as the second-price auction. $(M + 1)$ st-price sealed-bid

¹ In general, the round complexity of MPC is proportional to the multiplicative depth of an arithmetic circuit to evaluate the function f .

auction problem is essentially finding the highest and the $(M + 1)$ -th highest value, which can be simply solved by oblivious sort.

Obfuscated shuffle problem has been thoroughly studied in the context of e-voting with mix-nets [Cha81,Wik04] and onion-routing [CL05,MTHK09]. These solutions propagate encrypted messages through a network of servers to achieve unlinkability at the endpoint. In our context, we need multi-party computation protocols that shuffle secret shared values. A protocol that runs between a list of machines $\mathcal{P}_1, \dots, \mathcal{P}_m$ perform a mix-net, and the parties $\mathcal{P}_1, \dots, \mathcal{P}_m$ are referred as mix servers. Input is a list of messages $\mathcal{M}_1, \dots, \mathcal{M}_n$, and output of the mix-net is a obfuscated permutation of the messages $\mathcal{M}_{\pi(1)}, \dots, \mathcal{M}_{\pi(n)}$, where π is unknown to everyone. As another application, we can assign each input message a random number, and the obfuscated shuffle can be achieved in constant rounds by sorting the messages according to their associated random numbers.

Notations Throughout the paper we use the following notations. The message space is denoted by \mathbb{K} or depending on a context. Denote input array size by n and number of parties by m . \mathcal{P}_i stands for the i -th party. Denote shares of value $\alpha \in \mathbb{K}$ as $[\alpha]$ and a share held by \mathcal{P}_i as $[\alpha]_i$. In some protocols, R is the range of numbers to be sorted, and particularly, we assume that the numbers are from $[0, R]$. Let $\text{COM}([\![x]\!], [\![y]\!])$ be the comparison circuit, s.t.

$$\text{COM}([\![x]\!], [\![y]\!]) = \begin{cases} [\![1]\!] & x \geq y \\ [\![0]\!] & x < y \end{cases}$$

Denote Split as the bit-decomposition protocol that decomposes a share value into a vector of shared bits. Although comparison can be used for equality check in general, we only want to check whether a shared value is equal to a public value, which can be implemented by unbounded fan-in AND circuit. (c.f. Sect.5, below) Let $\text{EQ}([\![x]\!], y)$ be equality-check circuit, s.t.

$$\text{EQ}([\![x]\!], y) = \begin{cases} [\![1]\!] & x = y \\ [\![0]\!] & x \neq y \end{cases}$$

Let τ_{ad} , τ_{mul} , τ_{and} , τ_{or} , τ_{eq} , τ_{com} and τ_{bd} be the round complexity of addition, multiplication, unbounded fan-in AND, unbounded fan-in OR, equality-check, comparison and bit-decomposition protocol of the underlying secure MPC, respectively.

Our Contributions We proposed several constant-round oblivious sorting algorithms for MPC with different properties. Table 1 illustrates the comparison between some widely used oblivious sorting algorithms and our schemes,

in terms of round complexity and communication & computation complexity. As applications, we also show how to use our sorting algorithms in secure $(M + 1)$ st-price sealed-bid auction protocols and mix-net from obfuscated shuffle.

Sorting Scheme	Rounds	Comm. Complexity	Comp. Complexity
AKS Sorting Network [AKS83]	$\mathcal{O}(\log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$ COM
Balanced Sort [DPSR83]	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(n \log^2 n)$ COM
Randomized Shellsort [Goo09]	$\mathcal{O}(\log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$ COM
Oblivious Sort for MPC [JKU11]	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(n \log^2 n)$ COM
Sect. 5	$\mathcal{O}(1)$	$\mathcal{O}(Rn)$	$\mathcal{O}(n + R)$ Split + $\mathcal{O}(Rn)$ AND
Sect. 6	$\mathcal{O}(1)$	$\mathcal{O}(Rn)$	$\mathcal{O}(Rn)$ COM
Sect. 7	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$ COM + $\mathcal{O}(n^2)$ EQ

Table 1. Comparison of well-known oblivious sorting networks and our three new sorting algorithms for MPC. R is the range of numbers. COM is comparison, Split is bit-decomposition, EQ is equality-check and AND is unbounded fan-in AND gate.

2 Related Works

A (multi-terminal) Binary Decision Diagram (BDD, also known as a branching program, [Weg00]) is a fanout-2 directed acyclic graph $(\mathcal{V}, \mathcal{E})$. A language has a polynomial-size branching program if and only if it belongs to the complexity class $\mathbf{L/poly}$ [Cob66], that is, if it can be decided by a nonuniform log-space Turing machine. In paper [IK02], the authors proposed a technique that representing any binary decision diagram by low-degree randomizing polynomials over field \mathbb{F} . And later, in paper [CFIK03], the authors generalized the randomizing polynomial approach to ring \mathbb{R} . For evaluating a binary decision diagram of size $\text{size}(P)$, the computation and communication complexities are around $\text{size}(P)^2$. However, this approach has its limitations, and it is not easy to construct the optimal branching program for a complex function.

As regarding to general oblivious sorting algorithms, research interests are focus on how to achieve asymptotically optimal complexity, i.e. $\mathcal{O}(n \log n)$. AKS [AKS83] sorting network is the first famous oblivious sorting network, whose computational complexity is $\mathcal{O}(n \log n)$ compare-swap blocks. However, its constant factor is more than 6000 and it is quite complicated to implement. In practice, a lot of well-known oblivious sorting algorithms with $\mathcal{O}(n \log^2 n)$ complexity are used in secure MPC systems, such as balanced sort [DPSR83]

and Batcher’s bitonic sort [Bat68]. One can easily build an oblivious compare-swap block based on comparison circuit COM in MPC. Assume $(\llbracket C \rrbracket, \llbracket D \rrbracket) \leftarrow \text{ComSwap}(\llbracket A \rrbracket, \llbracket B \rrbracket)$, s.t. $C = \min(A, B)$ and $D = \max(A, B)$. One possible implementation could be as follows:

$$\begin{aligned}\llbracket C \rrbracket &= \llbracket A \rrbracket \cdot (1 - \text{COM}(\llbracket A \rrbracket, \llbracket B \rrbracket)) + \llbracket B \rrbracket \cdot \text{COM}(\llbracket A \rrbracket, \llbracket B \rrbracket) \\ \llbracket D \rrbracket &= \llbracket A \rrbracket \cdot \text{COM}(\llbracket A \rrbracket, \llbracket B \rrbracket) + \llbracket B \rrbracket \cdot (1 - \text{COM}(\llbracket A \rrbracket, \llbracket B \rrbracket))\end{aligned}$$

So those algorithms require at least $\mathcal{O}(\log^2 n)$ rounds. In 2009, a new simple oblivious sort called randomized Shellsort is introduced by Goodrich [Goo09]. The sorting algorithm achieves $\mathcal{O}(n \log n)$ complexity, and it can finish with $\mathcal{O}(\log n)$ rounds. However, it is not guaranteed to sort, there is a small probability that the output array is not well sorted. The authors call it dirtiness, and we call it ε -error in this paper. Therefore, it is not very convincing to use randomized Shellsort for secure $(M + 1)$ st-price sealed-bid auction, but it still can be used to generate random permutations. Very recently, Kristján Valur Jónsson *et al.* [JKU11] showed an oblivious sorting algorithm that is designed for MPC, and it uses $\mathcal{O}(n \log^2 n)$ comparisons in $\mathcal{O}(\log^2 n)$ rounds with practical constants.

3 Preliminaries

Share Computing A typical MPC is based on a threshold linear secret sharing scheme, e.g. Shamir secret sharing scheme [Sha79]. Due to the linearity, parties can obviously evaluate various linear combinations by doing local manipulations with shares. For all other operations, parties must execute secure multi-party protocols, which convert input shares to the shares of desired output without leaking any information about inputs. In particular, let $\llbracket x \rrbracket \cdot \llbracket y \rrbracket \equiv \llbracket x \cdot y \rrbracket$ denote output shares of a secure multiplication protocol. Then all other operations can be decomposed into multiplications and additions. The exact construction of a multiplication protocol uses the specifics of the underlying secret sharing scheme. For instance, the VIFF framework relies on properties of Shamir secret sharing [Sha79], whereas SHAREMIND uses tailor-suited solution for additive secret sharing [BLW08,BNTW10].

Adversarial model. For clarity and brevity, we consider only the static corruption model where adversary specifies parties to be corrupted before the protocol starts. The adaptive model could be sometimes more adequate in reality. However, the complexity of the formalism would carry us away from the core of our ideas. Although the list of tolerated adversarial coalitions can be arbitrary, share

computing systems can achieve information theoretical security is only if the condition Q2 is satisfied in the semihonest model and the condition Q3 is satisfied in the malicious model [HM00]. Recall that the condition Q2 means that any union of two tolerated adversarial coalitions is not sufficient to corrupt all parties and the condition Q3 means that any union of three tolerated adversarial set are not sufficient. In the case of threshold corruption, the conditions Q2 and Q3 imply that the number corrupted parties is strictly below $\frac{m}{2}$ and $\frac{m}{3}$.

Universal composability. As formal security proofs are rather technical, security proofs are often reduced to the security properties of sub-protocols. More specifically, one must assume that all sub-protocols are *universally composable* to automatically deduce security of a compound protocol. Although the formal definition of universal composability is rather complex, the intuition behind it is simple. Let $\varrho(\cdot)$ be a global context that uses the functionality of a protocol π . Let π° be an idealized implementation, where all computations are done by trusted third party who privately gathers all inputs and distributes the resulting outputs. Then we can compare real and ideal world protocols $\varrho(\pi)$ and $\varrho(\pi^\circ)$. A protocol π is *universally composable* if for any real world adversary \mathcal{A} there exist an adversary \mathcal{A}° against $\varrho(\pi^\circ)$ with comparable complexity and success rate. That is, the joint distribution of all outputs in the real and ideal world must coincide for all input distributions. As a result, if a compound protocol consisting of several instances of π° preserves security if we replace π° by π . The latter means that we combine universally composable sub-protocols without any usage restrictions, e.g., execute them in parallel. See the standard treatments [Can00,PSW00] for further details.

Achieving universal composability in the semihonest model is rather straightforward, most share computing protocols satisfy this including the protocols used in SHAREMIND and VIFF [BLW08,vif]. Theoretical constructions for malicious model do exist [DGKN09], but these are not widely used in practical systems, yet.

Security against active adversaries. In this work, we explicitly assume that share-computing uses two-level secret sharing [CM00], where the shares $\llbracket x \rrbracket_i$ of shared values are shared again. Essentially, if \mathcal{P}_i is honest then the malicious adversary is guaranteed to learn nothing during these proofs. The second level secret sharing scheme is commonly referred to as linear distributed commitment and it satisfies both perfect binding and hiding properties. As shown in [CM00], security against an active adversary can be achieved with three auxiliary protocols: commitment transfer protocol (CTP), commitment sharing protocol (CSP) and commitment multiplication protocol (CMP). CTP allows to transfer a com-

mitment for a secret from one party to another, CSP allows to share a committed secret in a verifiable way such that the parties will be committed to their shares, and CMP allows to prove that three committed secrets a , b and c satisfy the relation $c = ab$.

Counting Sort and Bead Sort Counting sort [Knu98] was created by Harold H. Seward in 1954. It is used to sort an array of numbers with discrete values, especially integers. Counting sort works with complexity $\mathcal{O}(n + R)$. During the counting sort, it first creates an array of counters. It then scans the input array once, increasing the corresponding counters one by one. At the end, it reconstructs the elements into output array, according to the counters.

Bead sort is a kind of natural sorting algorithm specifically for sorting natural numbers [ACD02]. The counting frame looks like an abacus that has rods and beads. Each numbers are represented as beads, e.g. 5 beads for number 5. For instance, the left matrix illustrates input numbers 4, 3, 5, 1, 3, where 1 stands for bead and 0 stands for empty space. From top-down, there are 4 beads, 3 beads, 5 beads, 1 beads and 3 beads in each layer, respectively. Due to gravity, the beads fall down automatically, as the right matrix shows. Then if we read from top-down, the sorted numbers are 1, 3, 3, 4, 5.

$$\begin{array}{|c|} \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline \end{array} \implies \begin{array}{|c|} \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}$$

The authors constructed some special hardware to simulate such natural falling, while if we implement bead sort in software, its complexity is still around $\mathcal{O}(Rn)$. To sort numbers a_0, \dots, a_{n-1} , the best complexity with software implementation is $\mathcal{O}(S)$, where $S = \sum_{i=0}^{n-1} a_i$.

4 Unbounded Fan-in AND Gate

It is possible to evaluate unbounded fan-in AND gate with constant rounds. As shown by Damgård *et al.* [DFNT05,DFK⁺06], there is a technique to evaluate unbounded fan-in multiplication in constant rounds. Assume we want to compute $\llbracket s \rrbracket = \prod_{i=1}^k \llbracket a_i \rrbracket$, where $a_i \in \mathbb{F}^*$. We can first generate random invertible pairs as follows: randomly pick $(\llbracket b \rrbracket, \llbracket c \rrbracket) \leftarrow \mathbb{F}^* \times \mathbb{F}^*$; Compute and open $d = \llbracket b \rrbracket \cdot \llbracket c \rrbracket$; Compute d^{-1} , then $\llbracket b \rrbracket$ and $\llbracket b^{-1} \rrbracket = d^{-1} \cdot \llbracket c \rrbracket$ give us a random invertible pair. To compute unbounded fan-in multiplication, we first generate random invertible pairs $(\llbracket b_0 \rrbracket, \llbracket b_0^{-1} \rrbracket), \dots, (\llbracket b_k \rrbracket, \llbracket b_k^{-1} \rrbracket)$. Compute and

open $c_i = \llbracket b_{i-1} \rrbracket \cdot \llbracket a_i \rrbracket \cdot \llbracket b_i^{-1} \rrbracket$ for $i \in \{1, \dots, k\}$. Compute $d = \prod_{i=1}^k c_i = b_0(\prod_{i=1}^k a_i)b_k^{-1}$, and then return $\llbracket s \rrbracket = \llbracket b_0^{-1} \rrbracket \cdot d \cdot \llbracket b_k \rrbracket$.

Armed with unbounded fan-in multiplication, one can evaluate unbounded fan-in AND gate, denoting as $\bigwedge_{i=1}^m \llbracket e_i \rrbracket$, where $e_i \in \{0, 1\}$. Because $e_i \in \{0, 1\}$, then $\sum_{i=1}^m \llbracket e_i \rrbracket \in [0, m]$. Thus, $\llbracket Y \rrbracket = (\sum_{i=1}^m \llbracket e_i \rrbracket + 1) \in [1, m+1] \subset \mathbb{F}^*$. It is possible to use Lagrange interpolation to construct a polynomial $P(X) = \sum_{i=0}^{m+1} \alpha_i X^i$, s.t. $P(j) = 0$ for $j \in \{1, 2, \dots, m\}$ and $P(m+1) = 1$. Note that α_i are public, and we can use constant-round unbounded fan-in multiplication to compute Y^2, \dots, Y^{m+1} . Therefore, we can evaluate polynomial $P(\llbracket Y \rrbracket)$ in constant rounds as well.

This technique can not be efficiently generalized to MPC over ring \mathbb{R} . However, for unbounded fan-in AND gate and OR gate, we can simply reduce them to equality-check.

Theorem 1. *There exists constants $c_1, c_2 > 0$ s.t. $\tau_{\text{and}} = c_1 \cdot \tau_{\text{eq}}$ and $\tau_{\text{or}} = c_2 \cdot \tau_{\text{eq}}$ for any MPC over linear secret sharing scheme.*

Proof. Let τ_{ad} be the round complexity for compute $\llbracket x \rrbracket + \llbracket y \rrbracket$. Due to linearity of the shares, we have $\tau_{\text{ad}} = 0$. τ_{and} is the round complexity of unbounded fan-in AND gate, i.e. $\alpha = \bigwedge_{i=1}^m \llbracket e_i \rrbracket$, where $e_i \in \{0, 1\}$. It is obvious that we have $\alpha = \text{EQ}(\sum_{i=1}^m \llbracket e_i \rrbracket, m)$. Similarly, τ_{or} is the round complexity of unbounded fan-in OR gate, i.e. $\beta = \bigvee_{i=1}^m \llbracket e_i \rrbracket$, where $e_i \in \{0, 1\}$. So we have $\beta = 1 - \text{EQ}(\sum_{i=1}^m \llbracket e_i \rrbracket, 0)$. Note that we assume $[0, m] \subseteq \mathbb{K}$, the same as Damgård *et al.* did in [DFNT05,DFK⁺06]. \square

5 Constant-Round Oblivious Counting Sort

In a typical secure MPC, the underlying secret sharing scheme is over some finite field \mathbb{F} or some finite ring \mathbb{R} . Thus, by its nature, the values are discrete and in limited range. It is a good scenario to use counting sort, but we have to make it oblivious. Alternatively, we can simulate bead sort in software and make it oblivious. It turns out that both methods converge to the same algorithm during our research. Assume that there is an input array A to be sorted, where $A[i] \in [0, R]$ for $i \in \{0, 1, \dots, n-1\}$. Alg. 1 depicts a constant-round oblivious counting sort that requires $(n+R)$ constant-round bit-decomposition sub-protocol(, e.g. [DFNT05,]) calls and $2Rn$ constant-round unbounded fan-in AND gates. How to evaluate such unbounded fan-in AND gate in constant rounds is already shown in Sect. 4.

Analysis and Examples In the first step, it compares each $A[i]$ with all the possible values in $[0, R]$. It first de-composites $A[i]$, and then it uses unbounded

Algorithm 1: Constant-Round Oblivious Counting Sort

Data: Array $\llbracket A[0] \rrbracket, \dots, \llbracket A[n-1] \rrbracket$
Result: Sorted array $\llbracket A[0] \rrbracket, \dots, \llbracket A[n-1] \rrbracket$

1. **for** $i \in \{0, \dots, n-1\}$ **do**
 - Call bit-decomposition protocol $\{\llbracket A_0[i] \rrbracket, \dots, \llbracket A_t[i] \rrbracket\} \leftarrow \text{Split}(\llbracket A[i] \rrbracket)$, where $t = \lceil \log R \rceil$.
 - for** $j \in \{0, \dots, R\}$ **do**
 - Let $b_0 b_1 \dots b_t$ to be the binary expression of j .
 - for** $k \in \{0, \dots, t\}$ **do**
 - if** $b_k = 0$ **then**
 - | $\llbracket e_k \rrbracket = 1 - \llbracket A_k[i] \rrbracket$;
 - end**
 - if** $b_k = 1$ **then**
 - | $\llbracket e_k \rrbracket = \llbracket A_k[i] \rrbracket$;
 - end**
 - end**
 - $\llbracket C_{i,j} \rrbracket = \bigwedge_{k=0}^t \llbracket e_k \rrbracket$;
 - end**
2. **for** $j \in \{0, \dots, R\}$ **do**
 - | $\llbracket S[j] \rrbracket = \sum_{w=j}^R \sum_{i=0}^{n-1} \llbracket C_{i,w} \rrbracket$;
- end**
3. **for** $j \in \{0, \dots, R\}$ **do**
 - Call bit-decomposition protocol $\{\llbracket S_0[j] \rrbracket, \dots, \llbracket S_m[j] \rrbracket\} \leftarrow \text{Split}(\llbracket S[j] \rrbracket - 1)$, where $m = \lceil \log n \rceil$.
 - for** $i \in \{0, \dots, n-1\}$ **do**
 - Let $b_0 b_1 \dots b_m$ to be the binary expression of i .
 - for** $k \in \{0, \dots, m\}$ **do**
 - if** $b_k = 0$ **then**
 - | $\llbracket e_k \rrbracket = 1 - \llbracket S_k[j] \rrbracket$;
 - end**
 - if** $b_k = 1$ **then**
 - | $\llbracket e_k \rrbracket = \llbracket S_k[j] \rrbracket$;
 - end**
 - end**
 - $\llbracket E_{i,j} \rrbracket = \bigwedge_{k=0}^m \llbracket e_k \rrbracket$;
 - end**
 - for** $i \in \{0, \dots, n-1\}$ **do**
 - | $\llbracket D_{i,j} \rrbracket = \sum_{w=i}^{n-1} \llbracket E_{i,w} \rrbracket$;
 - end**
 - end**
 - 4. **for** $i \in \{0, \dots, n-1\}$ **do**
 - | $\llbracket A[n-i-1] \rrbracket = \sum_{j=0}^R \llbracket D_{i,j} \rrbracket - 1$;
 - end**
 - 5. **return** $\llbracket A[0] \rrbracket, \dots, \llbracket A[n-1] \rrbracket$;

fan-in AND gate to evaluate equality-check gate $\text{EQ}(\llbracket x \rrbracket, y)$. We did not write equality-check gates explicitly in Alg. 1 in order to avoid unnecessary duplication of bit-decomposition sub-protocol calls. A stand alone equality-check gate $\text{EQ}(\llbracket x \rrbracket, y)$ can be evaluated by Alg. 2.

Algorithm 2: Equality-Check

Data: $\llbracket x \rrbracket$ and public y ($x, y \in [0, R]$)

Result: $\text{EQ}(\llbracket x \rrbracket, y)$

1. Call bit-decomposition protocol $\{\llbracket x_0 \rrbracket, \dots, \llbracket x_t \rrbracket\} \leftarrow \text{Split}(\llbracket x \rrbracket)$, where $t = \lfloor \log R \rfloor$.
2. Let $b_0 b_1 \dots b_t$ to be the binary expression of y .

for $k \in \{0, \dots, t\}$ **do**

if $b_k = 0$ **then**
 | $\llbracket e_k \rrbracket = 1 - \llbracket x_k \rrbracket$;

end

if $b_k = 1$ **then**
 | $\llbracket e_k \rrbracket = \llbracket x_k \rrbracket$;

end

end

3. **return** $\llbracket C \rrbracket = \bigwedge_{k=0}^t \llbracket e_k \rrbracket$;
-

Theorem 2. $\tau_{\text{eq}} \leq \tau_{\text{bd}} + \tau_{\text{and}}$ for any MPC over linear secret sharing scheme.

Proof. There exists a construction to evaluate equality-check by using bit-decomposition and unbounded fan-in AND, such as Alg.2. Therefore, it is clear that τ_{eq} is upper-bounded by $\alpha = \tau_{\text{bd}} + \tau_{\text{and}}$. \square

Take input array $A[6] = \{3, 4, 1, 0, 2, 1\}$ as an example. After step 1, we have the following left matrix C , where each row has exactly one position to be 1. Each column stands for values $\{0, 1, \dots, R\}$. After the inner sum of step 2, $S^* = \{1, 2, 1, 1, 1\}$. Here, $S^*[j]$ are the counters in counting sort. In order to make counting sort oblivious, we compute $\llbracket S[j] \rrbracket = \sum_{w=j}^R S^*[w]$. After that, $S = \{6, 5, 3, 2, 1\}$. However, implicitly, we can have matrix C' during step 2, and it is equivalent to bead representation of the numbers.

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad C' = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

During steps 3 and 4 in Alg. 1, it obviously reconstructs the numbers back to array A , according to the modified counters. It first compare the modified counters $S[j] - 1$ with all the possible values $[0, n - 1]$. Consequently, we have the follow matrices E and D . In fact, we “visualize” the counters as bitmap D in step 3. After step 4, we got sorted array $A = \{0, 1, 1, 2, 3, 4\}$.

$$E = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Efficiency and Security We have to make sure that $[0, \max(n, R)] \subseteq \mathbb{K}$, where \mathbb{K} is the message space of underlying secret sharing scheme. Although the sorting algorithm needs $\mathcal{O}(Rn)$ memory, it minimize the communication cost, for unbounded fan-in AND consumes much less communication than general comparison.

Theorem 3. *Sorting algorithm Alg. 1 achieves UC security, and it terminates after $2(\tau_{\text{bd}} + \tau_{\text{and}})$ rounds if the underlying MPC primitives are UC secure.*

Proof. For round complexity, evaluating share addition takes 0 rounds without communication cost in a typical secure MPC over linear secret sharing schemes. The sequentially dependent steps in Alg. 1 are only twice bit-decompositions and twice unbounded fan-in AND gates. Thus, it is clear that the sorting algorithm takes $2(\tau_{\text{bd}} + \tau_{\text{and}})$ rounds. Note that τ_{bd} and τ_{and} are constants that independent to input size n [DFNT05].

In terms of security, it follows standard hybrid model under UC theorem [Can01,Can00]. First, it is clear that this sorting algorithm is data-oblivious, because its behavior is always scanning the whole arrays and matrices. In our setting, if a tolerable subset of parties are corrupted, then each share computing protocol remains secure. Because of UC, we can replace them with ideal functionalities, where a trusted third party \mathcal{T} carries over the computations. We only consider the malicious model here, and the security proof is analogous for the semihonest model. We replace each share computing protocol with an ideal functionality (i.e. \mathcal{F}_{Add} , $\mathcal{F}_{\text{Split}}$ and \mathcal{F}_{AND}), \mathcal{T} privately collects input shares, reconstructs the inputs, computes the output, and sends the shares of the output back to the parties. In the resulting hybrid model, parties just use outputs as inputs of other protocols and corrupted parties can only alter the shares of intermediate results. However, the security of share computing protocols implies that the secret sharing scheme must be robust, i.e. the shared value can be still

reliably reconstructed even if corrupted parties alter their shares. Consequently, the actions of a malicious coalition do not change the outcomes of sub-protocols in the hybrid model.

Hence, the resulting simulator construction \mathcal{S} is straightforward. As \mathcal{S} must simulate the remain parties to corrupted parties, it can reconstruct the corresponding input array from the shares sent from the parties and forward them to \mathcal{T} . To simulate the outcomes of sub-protocols for the parties, \mathcal{S} can simply share 0 as many times as needed. This does not change the output distribution, as a tolerated malicious coalition cannot distinguish between shares of different values. For the simulation of the final result, \mathcal{S} just forwards the shares of sorted output array generated by \mathcal{T} to the corrupted parties. The simulation is perfect, because we have the same distribution in the ideal and hybrid world. As the simulator is non-rewinding, the oblivious sorting protocol is also universally composable, see the manuscript [Can00, p. 48–50] for further details. \square

6 Oblivious Arrayless Bead Sort

Since, sometimes, $\mathcal{O}(Rn)$ memory requirement is huge in Alg. 1. We are going to show how to get rid of the bitmap and simplify the sorting algorithm in this section. As depicted in Alg. 3, the pseudo-code looks very efficient at high level, and its minimum memory requirement is only $\mathcal{O}(R)$. Recall that $\text{COM}(\llbracket x \rrbracket, y) = \llbracket 1 \rrbracket$ if $x \geq y$; $\text{COM}(\llbracket x \rrbracket, y) = \llbracket 0 \rrbracket$, otherwise.

Algorithm 3: Constant-Round Oblivious Arrayless Bead Sort

Data: Array $\llbracket A[0] \rrbracket, \dots, \llbracket A[n-1] \rrbracket$ ($A[i] \in [0, R]$)
Result: Sorted array $\llbracket A[0] \rrbracket, \dots, \llbracket A[n-1] \rrbracket$

1. **for** $j \in \{1, \dots, R\}$ **do**
 | $\llbracket S_j \rrbracket = \sum_{i=0}^{n-1} \text{COM}(\llbracket A[i] \rrbracket, j)$;
 end
2. **for** $i \in \{1, \dots, n\}$ **do**
 | $\llbracket A[n-i] \rrbracket = \sum_{w=1}^R \text{COM}(\llbracket S_w \rrbracket, i)$;
 end
3. **return** $\llbracket A[0] \rrbracket, \dots, \llbracket A[n-1] \rrbracket$;

Analysis and Examples Take the same input array $A[6] = \{3, 4, 1, 0, 2, 1\}$ as an example. $n = 6$ and $R = 4$. After step 1, we have the array of counters $\{S_j\}_{j=1}^R = \{5, 3, 2, 1\}$. S_j is used to count how many numbers in input array A is larger than or equal to j . Step 2 reconstructs the numbers obliviously according to the counters. After step 2, we get the sorted array $A = \{0, 1, 1, 2, 3, 4\}$.

Efficiency and Security Again, we have to make sure that $[0, \max(n, R)] \subseteq \mathbb{K}$, where \mathbb{K} is the message space of underlying secret sharing scheme. The oblivious arrayless bead sort, Alg. 3, only needs R counters as minimum memory requirement. However, in order to evaluate the algorithm in parallel, the actual temporary memory usage is higher. The high level code looks very tidy, but the real communication is higher than Alg. 1 by a constant factor. On the other hand, since the sorting algorithm is very close to bead sort, we also can deal with negative numbers with anti-gravity bead sort, i.e. all the 1-s raise up to the ceiling for those columns corresponding to negative numbers.

Theorem 4. *Sorting algorithm Alg. 3 achieves UC security, and it terminates after $2\tau_{\text{com}}$ rounds if the underlying MPC primitives are UC secure.*

Proof (Sketch). For round complexity, evaluating share addition takes 0 rounds without communication cost. The sequentially dependent steps in Alg. 3 is only twice comparisons. Thus, it is clear that the sorting algorithm takes $2\tau_{\text{com}}$ rounds. Here, τ_{com} is constant that independent to input size n [DFK⁺06]. In terms of security, it is clear that this sorting algorithm is data-oblivious, for it only scans the whole array and counters. The security proof is very similar as the proof of Theorem 3, and it follows standard hybrid model under universally composable [Can01,Can00]. Since all the share computing sub-protocols are UC secure, we can replace them with ideal functionalities. We skip straightforward simulator construction \mathcal{S} for space limitation. □

7 Sorting Key Indexed Data Structure

In this section, we are going to show how to construct a constant-round oblivious sort that is based on comparison. In some cases, we do not only want to sort an array of numbers, but also want to sort key indexed data structures. Denote \hat{D}_i as an arbitrary shared data structure, e.g. a set of shared data. We want to sort \hat{D}_i according to their associated keywords $[[A_i]]$, for $i \in \{0, 1, \dots, n-1\}$. The intuition of the oblivious sort is based on Observation 1.

Observation 1 *Given array A with size n to be sorted such that $\forall i, j$, if $i \neq j$ then $A[i] \neq A[j]$. The number of elements in A that are smaller than the element $A[i]$ indicates the exactly position that $A[i]$ should be after the sort.*

Therefore, we need a transformation scheme that guarantees no duplicated transformed keywords.

Theorem 5. *There exists a transformation map: $\zeta : \{A_i\}_{i=0}^{n-1} \rightarrow \{A'_i\}_{i=0}^{n-1}$ such that the following properties holds:*

1. $\forall i, j \quad i \neq j : A'_i \neq A'_j$
2. $A'_i < A'_j$ if $A_i < A_j$
3. $A'_i < A'_j$ if $A_i = A_j$ and $i < j$
4. $A'_i > A'_j$ otherwise

Proof. Let $\beta \geq n$ be a constant. One possible transformation map ζ can be $A'_i \leftarrow A_i \cdot \beta + i$. It is obvious that this transformation scheme achieves these properties. \square

Algorithm 4: ExComp

Data: $(\llbracket A_i \rrbracket, i)$ and $(\llbracket A_j \rrbracket, j)$
Result: $\llbracket 1 \rrbracket$ if $A[i] \cdot n + i > A[j] \cdot n + j$; $\llbracket 0 \rrbracket$, otherwise
if $i > j$ **then**
 | **return** $\text{COM}(\llbracket A_i \rrbracket, \llbracket A_j \rrbracket)$;
end
if $j > i$ **then**
 | **return** $(1 - \text{COM}(\llbracket A_j \rrbracket, \llbracket A_i \rrbracket))$;
end

We are going to “use” the transformation ζ is our oblivious keyword sort. (c.f. Alg. 5, below) Note that the third property, i.e. $A'_i < A'_j$ if $A_i = A_j$ and $i < j$, is very important to keep our sort stable. However, in terms of efficiency, it is not necessary to require $[0, Rn] \subseteq \mathbb{K}$. Conceptually, we consider $A'[i] = A[i]||i$ and extend the comparison as Alg. 4.

Algorithm 5: Constant-Round Oblivious Keyword Sort

Data: Array of pairs: $(\hat{D}_0, \llbracket A_0 \rrbracket), \dots, (\hat{D}_{n-1}, \llbracket A_{n-1} \rrbracket)$
Result: Sorted array \hat{D} : $\hat{D}_0, \dots, \hat{D}_{n-1}$
1. **for** $i \in \{0, \dots, n-1\}$ **do**
 | $\llbracket C_i \rrbracket = \sum_{j=0, j \neq i}^{n-1} \text{ExComp}(\llbracket A_i \rrbracket, i), (\llbracket A_j \rrbracket, j))$;
end
2. **for** $i \in \{0, \dots, n-1\}$ **do**
 | $\hat{D}_i = \sum_{w=0}^{n-1} (\text{EQ}(\llbracket C_w \rrbracket, i) \cdot \hat{D}_w)$;
end
3. **return** $\hat{D}_0, \dots, \hat{D}_{n-1}$;

Efficiency and Correctness Conceptually, the original keywords A_i is transformed to $A'_i \leftarrow A_i \cdot n + i$ to ensure that there is no duplicated value. After step 1, C_i stores the position that \hat{D}_i is supposed to be in the sorted array. Step 2 puts \hat{D}_i to their corresponding positions obliviously. This step is also called oblivious selection, which is very close to oblivious transfer. The complexity of this sorting algorithm is $\mathcal{O}(n^2)$, but it finishes in constant rounds. Note that we write EQ for simplicity. As mentioned in previous sections, n^2 EQ gates can share n bit-decomposition calls, and they can be evaluated with n^2 unbounded fan-in AND gates.

Theorem 6. *Sorting algorithm Alg. 5 achieves UC security, and it terminates after $\tau_{\text{com}} + \tau_{\text{eq}} + \tau_{\text{mul}}$ rounds if the underlying MPC primitives are UC secure.*

Proof (Sketch). Transformation $\zeta : \llbracket A_i \rrbracket \cdot n + i$ is virtual, and ExComp only costs τ_{com} rounds. Thus, it is clear that the sorting algorithm takes $\tau_{\text{com}} + \tau_{\text{eq}} + \tau_{\text{mul}}$ rounds. In terms of security, it is clear that this sorting algorithm is data-oblivious, for it only scans the whole array and counters. The security proof is very similar as the proof of Theorem 3, and it follows standard hybrid model under universal composability [Can01, Can00]. Since all the share computing sub-protocols are UC secure, we can replace them with ideal functionalities. We skip the straightforward simulator construction \mathcal{S} for space limitation. \square

8 Dealing With Huge R

In some special cases, we want to sort huge numbers, where range $R \notin \mathbb{K}$. The numbers can be divided into several chunks and stored in several shares. Since the sorting algorithm Alg. 5 is stable, we can combine Alg. 5 with radix sort. We call it oblivious hybrid radix sort, as shown in Alg. 6, below. Assume $[0, 2^\alpha] \subseteq \mathbb{K}$ and $L = 2^\alpha$, where $L < R$ and $\alpha \geq 1$. The sorting algorithm takes $\mathcal{O}(\beta \cdot (\tau_{\text{com}} + \tau_{\text{eq}} + \tau_{\text{mul}}))$ rounds, where $\beta = \lceil \lceil \log_2 R \rceil / \alpha \rceil$. We denote the number $A[i]$ as $A_{\beta-1}[i] \cdots A_0[i]$, where $A[i] = \sum_{j=0}^{\beta-1} A_j[i] \cdot 2^{j\alpha}$ and $A_j[i] \in [0, 2^\alpha - 1]$.

Theorem 7. *Sorting algorithm Alg. 6 achieves UC security, and it terminates after $\mathcal{O}(\log R \cdot (\tau_{\text{com}} + \tau_{\text{eq}} + \tau_{\text{mul}}))$ rounds if the underlying MPC primitives are UC secure..*

Proof (Sketch). Since $\alpha = \lceil \log_2 L \rceil \geq 1$, the maximum number of sequential sorting algorithm Alg. 5 calls is $\beta = \Theta(\log R)$. As Alg. 5 terminates in $\tau_{\text{com}} + \tau_{\text{eq}} + \tau_{\text{mul}}$ rounds. The whole sort terminates in $\mathcal{O}(\log R \cdot (\tau_{\text{com}} + \tau_{\text{eq}} + \tau_{\text{mul}}))$

Algorithm 6: Oblivious Hybrid Radix Sort

Data: Array of splited numbers ($\llbracket A_{\beta-1}[i] \rrbracket, \dots, \llbracket A_0[i] \rrbracket$), for $i \in [0, n-1]$
Result: Sorted array ($\llbracket A_{\beta-1}[i] \rrbracket, \dots, \llbracket A_0[i] \rrbracket$), for $i \in [0, n-1]$

1. **for** $j \in \{0, \dots, \beta-1\}$ **do**
 - Set $\hat{D}_i \leftarrow (\llbracket A_{\beta-1}[i] \rrbracket, \dots, \llbracket A_0[i] \rrbracket)$;
 - Call Alg. 5 to sort array \hat{D} according to keywords $\llbracket A_j[i] \rrbracket$;
- end**
2. **return** $\hat{D}_i = (\llbracket A_{\beta-1}[i] \rrbracket, \dots, \llbracket A_0[i] \rrbracket)$, for $i \in [0, n-1]$;

rounds. Since it is just a batch of Alg. 5, the security proof is similar as the proof for Alg. 5. □

9 Applications

9.1 $(M+1)$ st-price Auction

The first direct application is secure sealed-bid auction protocol. During a $(M+1)$ st-price auction, each bidder i submits pairs $(\llbracket \text{name}_i \rrbracket, \llbracket \text{bid}_i \rrbracket)$. Set $\hat{D}_i \leftarrow (\llbracket \text{name}_i \rrbracket, \llbracket \text{bid}_i \rrbracket)$ and sort data structure \hat{D}_i according to keywords $\llbracket \text{bid}_i \rrbracket$ by calling Alg. 5. After sort, open the winner's name $\llbracket \text{name}_0 \rrbracket$ and open the win price $\llbracket \text{bid}_M \rrbracket$, (the $(M+1)$ -th price).

9.2 Constant-round Obfuscated Shuffle

Naively, one can achieve obfuscated shuffle in by multiplying a $n \times n$ permutation matrix \mathcal{A}_π , because any permutation π can be represented as a permutation matrix \mathcal{A}_π . However, it is not trivial to prepare a $n \times n$ permutation matrix \mathcal{A}_π in constant rounds. Alternatively, if each party \mathcal{P}_i shares a permutation matrix \mathcal{A}_{π_i} . All parties check whether the sums of every column and row of matrix \mathcal{A}_{π_i} are 1 by opening them for $i \in \{1, 2, \dots, m\}$. Note there is no information leakage, for they are always 1 if the party is honest. Let \mathcal{M} be the message vector, the obfuscated shuffle is computed by $\mathcal{M}' = \prod_{i=1}^m \mathcal{A}_{\pi_i} \cdot \mathcal{M}$. The resulting shuffle protocol contains $\mathcal{O}(mn^2)$ multiplications, which can be performed in $\mathcal{O}(m \cdot \tau_{\text{mul}})$ rounds. By computing $M_{\pi_1} \cdots M_{\pi_m}$ first, we can reduce the number of rounds to $\mathcal{O}(\log m \cdot \tau_{\text{mul}})$ with the cost of $\mathcal{O}(mn^3)$ multiplications.

On the other hand, obfuscated shuffle can be implemented by sorting random numbers. Assume we want to shuffle array $A[0], \dots, A[n-1]$, we first generate a fresh random number with $k \log n$ -bit long for each shared message, denoting as $R[i]$ for $i \in \{0, 1, \dots, n-1\}$. Similarly, we can use out oblivious

sorting algorithms to sort the shared messages according to associated random numbers. k is security parameter. If same random numbers are generated for two (or more) different documents, the relative order of them will be preserved after sort. The probability that we select n numbers out of $2^{(k \log n)} = n^k$ without collision is

$$P(n) = 1 \cdot \left(1 - \frac{1}{n^k}\right) \left(1 - \frac{2}{n^k}\right) \dots \left(1 - \frac{n-1}{n^k}\right) \\ \approx e^{-\frac{1}{n^k}} \cdot e^{-\frac{2}{n^k}} \cdot \dots \cdot e^{-\frac{n-1}{n^k}} \approx e^{-n^{2-k}}.$$

In practice, $k > 3$ is enough for large n . If $2^{k \log n} \notin \mathbb{K}$, we can divide the random numbers to several shares. (cf. Sect. 8) When m is big, it is clear that this approach is more round-efficient than the permutation matrix one.

10 Conclusions and Future Work

We proposed several constant-round oblivious sorting algorithms for secure MPC, which significantly improves current poly-logarithmic round complexity oblivious sorting network based solutions. We also showed some applications for our sorting algorithm is $(M + 1)$ st-price auction protocol and obfuscated shuffle. The future work will be improving the efficiency and investigating constant-round oblivious sorting algorithms with poly-logarithmic complexity overhead, i.e. $\mathcal{O}(n \log^c n)$ computation & communication complexity.

Acknowledgments. The author is supported by Estonian Science Foundation, grant #8058, the European Regional Development Fund through the Estonian Center of Excellence in Computer Science (EXCS) and ICT doctoral school. Most of this work was done while the author was working at Cybernetica AS.

References

- ACD02. Joshua J. Arulanandham, Cristian Calude, and Michael J. Dinneen. Bead-sort: A natural sorting algorithm. *Bulletin of the EATCS*, 76:153–161, 2002. 3
- AKS83. M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, January 1983. 1, 1, 2
- Bat68. K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM. 2
- BLW08. Dan Bogdanov, Sven Laur, and Jan Willemsen. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of LNCS, pages 192–206. Springer, 2008. 1, 3, 3

- BNTW10. Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. Improved protocols for the sharemind virtual machine. Research report T-4-10, Cybernetica, 2010. Available at <http://research.cyber.ee>. 3
- Can00. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. <http://eprint.iacr.org/>. 3, 5, 6, 7
- Can01. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:136, 2001. 5, 6, 7
- CFIK03. Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multi-party computation over rings. In Eli Biham, editor, *Advances in Cryptology EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 642–642. Springer Berlin / Heidelberg, 2003. 1, 2
- Cha81. David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981. 1
- CKMH07. Koji Chida, Hiroaki Kikuchi, Gembu Morohashi, and Keiichi Hirota. Efficient multiparty computation for comparator networks. In *ARES*, pages 1183–1189, 2007. 1
- CL05. Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 169–187. Springer, 2005. 1
- CM00. Ronald Cramer and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. pages 316–334. Springer-Verlag, 2000. 3
- Cob66. Alan Cobham. The recognition problem for the set of perfect squares. In *Proceedings of the 7th Annual Symposium on Switching and Automata Theory (swat 1966)*, pages 78–87, Washington, DC, USA, 1966. IEEE Computer Society. 2
- DFK⁺06. Ivan Damgrd, Matthias Fitzi, Eike Kiltz, Jesper Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer Berlin / Heidelberg, 2006. 1, 4, 4, 6
- DFNT05. Ivan Damgård, Matthias Fitzi, Jesper Buus Nielsen, and Tomas Toft. How to split a shared secret into shared bits in constant-round, 2005. <http://eprint.iacr.org/2005/140>. 1, 4, 4, 5, 5
- DGKN09. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Irvine: Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography*, pages 160–179, Berlin, Heidelberg, 2009. Springer-Verlag. 1, 3
- DPSR83. M. Dowd, Y. Perl, M. Saks, and L. Rudolph. The balanced sorting network. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, PODC '83, pages 161–172, New York, NY, USA, 1983. ACM. 1, 2
- Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, New York, NY, USA, 2009. ACM. 1
- Goo09. Michael T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. *CoRR*, abs/0909.1037, 2009. 1, 2
- HM00. Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000. 3
- IK02. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002. 1, 2

- IP07. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil Vadhan, editor, *Theory of Cryptography*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer Berlin / Heidelberg, 2007. 1
- JKU11. Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. Secure multi-party sorting and applications. Cryptology ePrint Archive, Report 2011/122, 2011. <http://eprint.iacr.org/>. 1, 2
- Knu98. Donald E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, 2 edition, May 1998. 3
- MTHK09. Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 590–599. ACM, 2009. 1
- PSW00. Birgit Pfizmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. RZ 3206 (#93252), IBM Research Division, Zrich, May 2000. 3
- RBO89. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 73–85, New York, NY, USA, 1989. ACM. 1
- Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22:612–613, November 1979. 3
- vif. Viff documentation. <http://viff.dk/doc/index.html>. 3
- Weg00. Ingo Wegener. *Branching programs and binary decision diagrams: theory and applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 2
- Wik04. Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004. 1
- WWW98. Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electronic commerce: theory and implementation. *Decision Support Systems*, 24(1):17–27, 1998. 1
- Yao82. Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society. 1