Round-efficient Oblivious Database Manipulation

Sven Laur[2], Jan Willemson[1,3], and Bingsheng Zhang[1,2]

[1] Cybernetica, Ülikooli 2, Tartu, Estonia
[2] Institute of Computer Science, University of Tartu, Liivi 2, Tartu, Estonia
[3] Software Technology and Applications Competence Center, Ülikooli 2, Tartu, Estonia

Abstract. Most of the multi-party computation frameworks can be viewed as oblivious databases where data is stored and processed in a secret-shared form. However, data manipulation in such databases can be slow and cumbersome without dedicated protocols for certain database operations. In this paper, we provide efficient protocols for oblivious selection, filtering and shuffle—essential tools in privacy-preserving data analysis. As the first contribution, we present a 1-out-of-n oblivious transfer protocol with $\mathrm{O}(\log\log n)$ rounds, which achieves optimal communication and time complexity and works over any ring $\mathbb{Z}_N$. Secondly, we show that the round complexity τ_{bd} of a bit decomposition protocol can be almost matched with oblivious transfer, and that there exists an oblivious transfer protocol with $\mathrm{O}(\tau_{\mathrm{bd}} \log^* n)$ rounds. Finally, we also show how to construct round-efficient shuffle protocols with optimal asymptotic computation complexity and provide several optimizations.
Keywords. Secure multi-party computation, oblivious transfer, verifiable shuffle, oblivious filtering.

1 Introduction

Privacy issues often arise when sensitive data is gathered from individuals and organizations. Such threats are commonly addressed with organizational and physical methods; however, on some occasions cryptographic methods can provide better alternatives. In this paper, we will concentrate on the methods based on secret sharing and multi-party computations. Such an approach guarantees by design that the computing parties will have no access to the underlying data provided that the number of collaborating malicious parties is small enough. On the other hand, such a guarantee comes with a price of increased computational complexity and generic secure computation techniques [36, 7, 15] have a prohibitively large overhead.

However, for particular problems it is possible to design protocols that outperform the general-purpose ones. In this paper, we will concentrate on some specific management tasks to be applied on secret shared databases. Since the whole idea behind keeping a database in secret shared form is to prevent data leaks, corresponding database operations should not reveal anything about the underlying dataset (except for what can be concluded from the desired output). Our target is to develop protocols for oblivious database access, filtering, text categorisation and encoding. We consider security in the client-server setting, i.e., the client should obtain correct answer to the query and nothing else, whereas the database holders should not learn anything besides the type of the query. Note that in the context of secret shared databases, client privacy is only guaranteed if the share holders collude below the tolerated threshold.

In data analysis, one often needs to filter data according to a specific criterion. The corresponding oblivious filtering procedure should produce a new shared database that contains only the records which satisfy the criterion. The share holders should learn nothing beyond the number of records. In particular, they should not learn which database rows we included. As such, the oblivious filtering allows us to reduce the database size and thus can remarkably reduce the overall complexity of the remaining steps.

To understand the requirements for data encoding, consider the case of privacy-preserving questionnaire analysis. It is relatively straightforward to design cryptographically secure data aggregation mechanisms for questions with multiple choice answers, since the responses can be encoded as integers. Questions with free text answers are much more challenging. First, secure text-processing is inherently slower than secure integer arithmetic, since the former is built on top of the latter one. Second, the answers must often be interpreted by human operators in order to extract relevant information. As a trade-off between privacy and efficiency, we present a protocol for oblivious database access. The protocol assures that the human operator who reads

text entries and encodes them to standard attributes, cannot link replies to particular responders nor to non-disclosed fields.

Related work. Even though the generic methods of secure multi-party computations have been known for decades [36, 7, 15], practical implementations of the respective frameworks have emerged only recently, e.g. FairPlayMP [6], VIFF [25], SEPIA [11], SecureSCM [3], VMcrypt [29], TASTY [27] and SHAREMIND [8]. Oblivious transfer has been commonly studied in the two-party setting [10, 4, 24] where security guarantees are inherently computational. Existing results in the multi-party setting mostly hold for private information retrieval [17, 5] where the database is replicated in several sites and only the secrecy of the query index i is protected. Similarly, the secure shuffle problem has been mostly studied in computational setting. Most solutions are given in the context of e-voting with mix-nets [16, 35] and onion-routing [12, 30]. 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. As there are no public key operations, such protocols are much more efficient and naturally fit into the framework of share computing environments, such as SHAREMIND [8] and VIFF [22]. Hence, one should view our work in the long line of works [21, 8, 11, 14] that uses addition and multiplication protocols in black-box manner to build secure implementations for standard data processing operations.

Road map. The paper is organized as follows. In Section 2, we describe general frameworks for share computing. Section 3 gives generic constructions for oblivious database manipulation, namely selection, filtering and general read-write access. Differently from earlier results [21], these protocols do not assume that the underlying message space is a field—any residue ring $\mathbb{Z}_N$ is sufficient. This is an important extension, since computer arithmetic in present-day computers implements only low-level ring operations, hence we can hope for better overall performance by concentrating on rings. The described generic protocols use oblivious shuffle as an important building block and three possible instantiations with several round-communication complexity trade-offs for it are presented in Section 4. Further performance tweaks are discussed in Section 5.

2 Frameworks for Share Computing

General setup. A typical privacy-preserving data analysis application involves three types of entities: data donors, computing parties (referred to as *miners*), and clients. Data donors own sensitive data, miners gather the data and clients want to find answers for various statistical questions. In such a setting, privacy issues can be addressed with the client-server model formalized by Damgård and Ishai [23]. In this model, data donors submit their data in a secret shared form to the miner nodes, which later use share computing protocols to carry out the computations requested by the clients. Since data is secret shared, individual records are protected as long as the miners do not form non-tolerated coalitions, i.e., they as a group follow the restrictions of share-computing protocols. Clients and data donors are not trusted and can arbitrarily violate protocol specifications. Depending on the underlying primitives and protocols, the framework can tolerate either semihonest or malicious corruption of miners.

Share computing frameworks. A typical framework for multi-party computations is based on a secret sharing scheme and a set of protocols for manipulating the shares. A secret sharing scheme is specified by randomised sharing and recovery algorithms. The sharing algorithm splits a secret value $x \in \mathbb{Z}_N$ into shares $x_1, \ldots, x_m$ that must be securely transferred to the miners $\mathcal{P}_1, \ldots, \mathcal{P}_m$, respectively. To recover the shared value, miners must together execute the reconstruction algorithm that takes in all shares and outputs the corresponding secret value. For example, the additive secret sharing scheme splits a secret x into shares such that the secret can be reconstructed by adding all the shares:

$$x \equiv x_1 + x_2 + \cdots + x_m \mod N \ .$$

The vector of shares $(x_1, x_2, \ldots, x_m)$ is commonly denoted by $[\![x]\!]$. Security properties of a secret sharing scheme are defined through a list of tolerable adversarial coalitions. Let $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}$ form a tolerable coalition, then the corresponding shares should leak nothing about the secret. More formally, the distributions $(x_{i_1}, \ldots, x_{i_k})$ and $(y_{i_1}, \ldots, y_{i_k})$ must coincide for any inputs $x, y \in \mathbb{Z}_N$.

Share computing protocols enable miners to perform oblivious computations with shares. For instance, miners can obtain a valid sharing of $x + y$ by locally adding their additive shares of x and y. Similarly, multiplying shares of x locally by a constant $\alpha \in \mathbb{Z}_N$ gives us shares of αx. A secret sharing scheme satisfying these two constraints is referred to as a *linear secret sharing scheme*. There are many linear secret sharing schemes, such as Shamir secret sharing scheme [33]. As any function can be represented as a Boolean circuit, it can be also decomposed into multiplication and addition operations. Thus, a share computing framework can be built on top of a linear secret sharing scheme by specifying a protocol for multiplication. For instance, the VIFF framework [1] uses standard solution based on Shamir secret sharing [33], whereas SHAREMIND uses tailor-suited multiplication protocol for additive secret sharing [8, 9].

Although protocols for secure share addition and multiplication are sufficient to achieve Turing completeness, the corresponding generic constructions are not efficient enough for practical applications. Hence, it makes sense to design specific protocols for common operations. In most cases, the effect of the network delay is several orders of magnitudes larger than the time needed to deliver protocol messages. Thus, protocols with minimal round complexity are the most efficient. However, the round count is not an absolute measure, as the delivery time becomes dominant for large messages. Hence, overall communication complexity is also important. Table 1 depicts round complexity of various share computing operations implemented in SHAREMIND, where ℓ is the bit length of modulus N. Precise description of the protocols together with empirical benchmarking can be found in [8, 9]. It is worth noting that logarithmic complexity in the bit size of the modulus N is asymptotically sub-optimal, as theoretical construction by Damgård *et al.* [21] provides a constant round solution. However, the corresponding round count is larger than $\log \ell$ for all practical residue rings $\mathbb{Z}_N$.

Operation	Round count	Complexity	Operation	Round count	Complexity
Multiplication	τ_{mul}	$\mathrm{O}(1)$	Coin-tossing	τ_{ct}	$\mathrm{O}(1)$
Smaller than or equal	τ_{st}	$\mathrm{O}(\log \ell)$	Strictly less	τ_{sl}	$\mathrm{O}(\log \ell)$
Equality test	τ_{eq}	$\mathrm{O}(\log \ell)$	Bit-decomposition	τ_{bd}	$\mathrm{O}(\log \ell)$

Table 1. Round complexity of common share-computing operations

For binary operations, we will use a shorthand $[\![x]\!] \circledast [\![y]\!]$ to denote the outputs of a share computing protocol that securely computes $x \circledast y$ from the shares of x and y.

Adversarial model. For clarity, we consider only the static corruption model where adversary specifies parties to be corrupted before the protocol starts, although most protocols can resist more advanced corruption models. Although the list of tolerated adversarial coalitions can be arbitrary, share computing systems can achieve information theoretical security only if the condition Q2 is satisfied in the semihonest model and the condition Q3 is satisfied in the malicious model [28]. 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 sets is 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 $\frac{m}{3}$, respectively.

Universal composability. As formal security proofs are rather technical, security proofs are often reduced to the security properties of sub-protocols. More specifically, one can deduce security of a compound protocol without delving into details only if all sub-protocols are *universally composable*. Although the formal definition of universal composability is rather complex, the intuition behind it is simple. Let $\varrho\langle\cdot\rangle$ be a global

context (often named as environment) that uses the functionality of a protocol π. Let π° be an idealised implementation, where all computations are done by a trusted third party who privately gathers all inputs and distributes the resulting outputs. Then we can compare real and ideal world protocols $\varrho\langle\pi\rangle$ and $\varrho\langle\pi^\circ\rangle$. A protocol π is *universally composable* if for any real world adversary $\mathcal{A}$ there exist an adversary $\mathcal{A}^\circ$ against $\varrho\langle\pi^\circ\rangle$ 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, 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.

We refer to the standard treatments [13, 32] 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 [8, 1]. Theoretical constructions for malicious model do exist [22], but these are not widely used in practical systems, yet.

3 Oblivious Database Manipulation

In this section, we describe various database operations that are often required in data analysis and show how these can be implemented using protocols for oblivious transfer and shuffle. The objective of the random shuffle protocol is to permute the elements of the underlying database according to a uniformly chosen permutation π, which is oblivious to all miners. Depending on the protocol, the client can either learn the permutation or not. We discuss how to implement the basic protocols in Sections 3.1 and 4.

3.1 Generic Construction for Oblivious Selection

Many data-mining algorithms select and later process a particular sub-sample from the entire data. In the simplest case, the client wants to retrieve a single data record without revealing the index. This problem is commonly referred to as *oblivious transfer*. More formally, let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a database of ℓ-bit strings and let i be the desired index. Then miner nodes should learn nothing about the client's input and the client should learn nothing beyond x_i. Far more often, the client is not interested in the value itself but needs shares of x_i for further processing. At the end of *oblivious selection* protocol, miner nodes obtain shares of x_i and neither client or miners learn anything new. Note that an oblivious selection protocol becomes an oblivious transfer protocol if miners send the obtained shares securely to the client. Secondly, the client can construct a sub-sample of the original database by executing several oblivious selection operations. The latter is a rather common operation in statistical studies. For example, assume that the database entries contain private information about individuals, however, it is public which database row belongs to which individual. Then the client can form a group of relevant persons based on private or public information about these individuals. With oblivious selection, the client can select only relevant rows without leaking the selected identities.

The generic construction for oblivious selection described below works under the assumption that both the database elements x_i and indices i can be considered as elements of a ring $\mathbb{Z}_N$. By the definition of secret sharing, we achieve client privacy by sharing the index i at the client side and just transferring the shares $[\![i]\!]$ to the miners. Then the miners can compute the output shares as follows

$$[\![x_i]\!] = \sum_{j=1}^{n} ([\![i]\!] = j) \cdot [\![x_j]\!] \tag{1}$$

where $([\![i]\!] = j)$ denotes the output shares of a secure comparison protocol, see Section 5 for further details. The result is correct as comparisons yield a zero-one indicator vector. Protocol 1 depicts the corresponding oblivious transfer protocol, where the output shares are sent back to the client. The GenOt protocol is as secure as the weakest sub-protocol. In particular, as sub-protocols are universally composable, so is the GenOt protocol. If sub-protocols are secure against active corruption, so is the GenOt protocol. Since now this is what we mean by stating that assumptions of share computing are fulfilled.

Data-gathering phase

1. Data donors submit the shares of their inputs x_i to the miner nodes.

Query phase

A client submits shares of i to the miner nodes.

Processing phase

1. For $j \in \{1, \ldots, n\}$, miners evaluate in parallel: $[\![y_i]\!] \leftarrow [\![x_j]\!] \cdot ([\![i]\!] = j)$.
2. Miners compute the shares of the reply: $[\![z]\!] \leftarrow [\![y_1]\!] + \cdots + [\![y_n]\!]$.

Reconstruction phase

Miners send the shares of z to the client who reconstructs and outputs z.

Protocol 1: Generic oblivious transfer protocol GenOt for the message domain $\mathbb{Z}_N$

We also do not provide complete security proofs, since simulator constructions for evaluating (arithmetic) circuits are standard. Due to universal composability, each share computing protocol can be replaced with an ideal implementation. Hence, a share-computing algorithm can be viewed as a circuit with trusted components for operations where each signal is split into m sub-wires carrying the shares. In the semihonest model, the adversary can eavesdrop the sub-wires corresponding to corrupted parties, whereas in malicious model the adversary can also alter the signals in these wires. Now the security of secret sharing implies that all eavesdropped messages can be simulated with shares of zero. For the malicious model, the underlying secret sharing scheme must be robust and thus blocking or altering signals in ceased sub-wires has no effect on the outcome. As a result, correctness of such circuits is sufficient for universal composability. We refer to the manuscripts [13, 20] for further details.

Theorem 1. *If the assumptions of share computing protocols are fulfilled, the* GenOt *protocol is secure against malicious data donors and clients. The corresponding round complexity is* $\tau_{\mathrm{eq}} + \tau_{\mathrm{mul}} + 1$ *where* τ_{mul} *and* τ_{eq} *are round complexities of multiplication and equality test protocols.*

Proof (Sketch). For the round complexity, note that equality test can be evaluated in parallel and the addition takes zero rounds. The protocol is secure against malicious data donors, as they can submit only elements of $\mathbb{Z}_N$. Similarly, the equality tests in the first step of the processing assure that all $y_i = 0$ if the client input is invalid (i.e. $i \notin \{1, \ldots, n\}$) and the malicious client receives no information. □

Remarks. First, note that miners can use any shared value as $[\![i]\!]$ in the protocol. In particular, same shares can be used to select or fetch elements form different databases. As protocols can be run in parallel, we can obliviously select database records of arbitrary format without increase in the round complexity. Moreover, miners can also assure that the inputs are in a certain range. In particular, miners can assure that $x_i \in \{0,1\}^\ell$ by setting $[\![x_i]\!] \leftarrow [\![x_i]\!] \cdot ([\![x_i]\!] < 2^\ell)$ in the data gathering stage. Second, note that the efficiency of the protocol depends mainly on the efficiency of equality testing. Hence, we must pay special attention to this subprotocol and we will do so in Section 5. Third, note that the communication complexities of data donors and clients are optimal up to a multiplicative factor (roughly m), as in the optimal non-private protocol they send and fetch values instead of shares. The miners' workload is linear in the database size, which is again optimal, since each database element must be touched or otherwise information leaks about the queries.

3.2 Generic Construction for Oblivious Database Filtering

In data analysis tasks, one often needs to separate a certain sub-sample form the entire data. For instance, we might constrain our sample to female patients who are over 65 and have high blood pressure. When the data is secret shared due to privacy reasons, the desired sub-sample must be formed obliviously. The procedure can be split into two phases. First, we must compute shares of an indicator vector $\boldsymbol{f}$ that is set to one when

1. Miners apply oblivious shuffling to permute the shares of the database $(\boldsymbol{x}||\boldsymbol{f})$.
2. Miners reconstruct the permuted vector $\pi(\boldsymbol{f})$ of the resulting database $(\pi(\boldsymbol{x})||\pi(\boldsymbol{f}))$.
3. Miners keep the shares of $\boldsymbol{x}_{\pi(i)}$ for which $f_{\pi(i)} = 1$ as shares of $\boldsymbol{y}$.

Protocol 2: Generic oblivious filtering protocol GenOf

the inclusion criterion is met. Standard share computing frameworks can easily handle predicates consisting of arithmetic and comparison operators, such as the constraint $(\llbracket sex \rrbracket = 1) \cdot (\llbracket age \rrbracket \geq 65) \cdot (\llbracket hbp \rrbracket = 1)$. Second, we must perform the oblivious filtering step. Let $\boldsymbol{x}$ be the vector of database records and let $\boldsymbol{f}$ be a zero-one indicator vector. Then during oblivious filtering protocol miners use shares of $\boldsymbol{x}$ and $\boldsymbol{f}$ to construct a new database of shared records $\boldsymbol{y}$ such that x_i is included into $\boldsymbol{y}$ iff $f_i = 1$. Miners learn nothing except the size of $\boldsymbol{y}$ during the protocol. The latter is unavoidable, if we want to reduce the workload in later processing stages – gains in efficiency unavoidably leak information about the size of the sub-sample $\boldsymbol{y}$.

For clarity, let $(\boldsymbol{x}||\boldsymbol{f})$ denote a record-wise concatenation of databases $\boldsymbol{x}$ and $\boldsymbol{f}$ consisting of pairs (x_i, f_i). For any permutation π, let $\pi(\boldsymbol{z})$ denote a reordered database $z_{\pi(1)}, \ldots, z_{\pi(n)}$. Then Protocol 2 depicts a generic construction that reduces oblivious filtering task to oblivious shuffling. As a consequence, efficiency of many data analysis algorithms is determined by the efficiency of oblivious shuffling. See Section 4 for the description of three oblivious shuffle protocols with different trade-offs between round complexity τ_{os} and communication complexity.

Theorem 2. *If assumptions of the oblivious shuffle protocol are fulfilled, the* GenOf *protocol is secure. The round complexity is* $\tau_{\text{os}} + 1$ *where* τ_{os} *is the round complexity of oblivious shuffle.*

Proof (Sketch). The claim about round complexity is evident as reconstruction is a single round operation. The protocol is correct as the shuffle protocol indeed yields shares of $(\pi(\boldsymbol{x})||\pi(\boldsymbol{f}))$ and thus correct shares of x_i are included into the output database. The security hinges on the fact that $\pi(\boldsymbol{f})$ is a random permutation and thus the output $\pi(\boldsymbol{f})$ can be easily simulated knowing the size of $\boldsymbol{y}$. □

3.3 Oblivious Read-Write Access

Oblivious database access is a powerful tool in data processing. For instance, we can used it for post-processing secret shared free-text fields. Assume that a human operator obtains all the text entries so that they could not be linked to the responders. Then the operator can extract necessary information form free-text entries for better encoding of answers. For completing the update step, the operator must be able to obliviously update the corresponding database fields. The latter can be achieved with the GenOwr protocol, where (1) miners apply oblivious shuffling to permute the shares of the database $\boldsymbol{x}$; (2) the client reads and writes fields of the shuffled database $\pi(\boldsymbol{x})$; (3) miners apply oblivious shuffling to permute the shares of the updated database $\hat{\boldsymbol{x}}$. The protocol does not provide full privacy, as the client learns some database fields and can link field updates with accessed values. However, such a level of privacy is sufficient for many practical query processing applications, as a completely secure solution is yet practically infeasible.

Theorem 3. *The overhead of the* GenOwr *protocol is* $2\tau_{\text{os}}$ *where* τ_{os} *is the round complexity of the oblivious shuffle protocol. If the assumptions of the oblivious shuffle protocol are fulfilled and the client does not access the same field twice for reading or writing, the miners learn only the number of read and write operations and which operations were performed on the same database record. The client learns the values of accessed database fields and can link updates of database entries with the values of accessed database fields.*

Proof (Sketch). If the oblivious shuffle protocol is secure, miners cannot link shares corresponding to $\boldsymbol{x}$, $\pi(\boldsymbol{x})$ and $\hat{\boldsymbol{x}}$. Consequently, the view of corrupted miners can be simulated knowing only the database access pattern in the second step. If the assumptions hold, the pattern can be correctly reconstructed given only the information enlisted in the theorem. As the corrupted miners cannot alter the end result, the joint distribution of outputs coincide in the real and ideal world. The second security claim holds, as the protocol hides the ordering of database entries form the client. The simulation is straightforward, since the client has read and write access of the database $\pi(\boldsymbol{x})$ in the idealised implementation. □

Remarks. The default implementation of GenOwr protocol produces the database where the entries are shuffled. The original order can be restored if the second shuffle implements the inverse of π, or if we add the shared row numbers to the original database and open them after the second shuffle. As another extension, note that the GenOwr protocol can be used for implementing adaptive oblivious transfer, where the client wants to retrieve several elements from a database, given that the client learns the permutation π. After the database is shuffled, the client can use π to query the required elements, whereas the miners will not know the correspondence. Hence, a single shuffle is sufficient for adaptive oblivious transfer of many elements. If clients do not query the same element more than once, miners will learn only the number of queries and nothing else. The same construction is valid for oblivious selection. As a result, we can significantly save computational time in algorithms that require massive parallel oblivious read-write access.

4 Protocols for Oblivious Shuffle

In the following, we describe three oblivious shuffle protocols. The first protocol presented in Section 4.1 based on permutation matrices has constant round complexity. However, the communication complexity is rather high ($\Theta(n^3)$ or $\Theta(n^2)$ depending on the variation). The resharing-based solution presented in Sections 4.3 and 4.4 achieves communication complexity asymptotically optimal in database size ($\Theta(n \log n)$), but is exponential with respect to the number of miners. In many applications this is not a major issue, since the number of miners is usually very limited (e.g. three for the current implementation of Sharemind). However, for larger requirements we can propose a good sorting-based trade-off in Section 4.2, as its communication complexity is $\Theta(n \operatorname{polylog} n)$ and it scales well to a larger number of miners.

Figure 1 shows the expected performance of all three protocols from two aspects. Round complexity depicted on the vertical axis characterises the performance for small database sizes. For large databases, the running time is approximately linear to the communication complexity between individual miners. Hence, the comparison of communication complexities displayed on the graph lines is more relevant in practice.

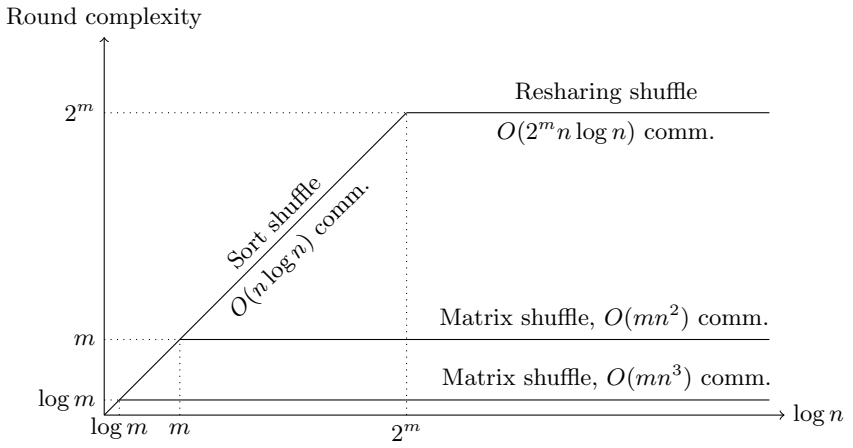

Fig. 1. Performance comparison of oblivious shuffle protocols

All three protocols are composed of consecutive shuffling phases which are designed to hide permutations from different sets of miners. In the first and the second protocol, only a single miner knows the permutation of each phase and thus O(m) phases are sufficient for security. In the third protocol, the permutation is known to a large set of miners and thus $\Theta(2^m/\sqrt{m})$ phases are needed in the worst case.

Similarly to oblivious transfer, we only consider the case where all database elements belong to $\mathbb{Z}_N$. As any collection of records can be represented as a matrix where each column is either a data field or part

of it, a vector shuffle protocol is sufficient provided that several protocol runs can share the same hidden permutation. It is easy to see that all protocols presented in this section have this property.

4.1 Oblivious Shuffle based on Permutation Matrices

Recall that for any n-element permutation π there exists a zero-one matrix M_π such that $\pi(\boldsymbol{x}) = M_\pi \boldsymbol{x}$ for all vectors $\boldsymbol{x}$ of size n. Hence, if a miner $\mathcal{P}_i$ generates and shares a permutation matrix M_{π_i}, the database can be shuffled by multiplying it with M_{π_i}. As none of the miners should know the permutation, miners must execute several such shuffle phases. More precisely, let t be the maximal size of a tolerated adversarial coalition. Then it is sufficient if $t+1$ miners permute the database, as at least one permutation remains oblivious to the adversarial coalition. Also, note that the final outcome is indeed a shuffle, as at least one permutation is chosen uniformly. As each permutation phase can be implemented with $\Theta(n^2)$ multiplications and $t = \Theta(m)$ in standard multiparty frameworks, the resulting shuffle protocol contains $\mathrm{O}(mn^2)$ multiplications, which can be performed in $\mathrm{O}(m\tau_{\mathrm{mul}})$ rounds. The number of rounds can be reduced to $\mathrm{O}(\log_2 m\tau_{\mathrm{mul}})$ with the cost of $\mathrm{O}(mn^3)$ multiplications if miners first compute the matrix product $M_{\pi_1} \cdots M_{\pi_{t+1}}$ in a balanced manner.

In the malicious model, miners have to additionally verify that each M_{π_i} is a permutation matrix, i.e., all entries are either zeroes or ones and the sums of all the rows and columns of M_{π_i} are ones. Each such test can be expressed as a share computing procedure yielding one if the input M_{π_i} is correct. Moreover, these additional zero-knowledge proofs do not change the complexity estimates. As a result, security follows from the correctness and universal composability of share computing protocols.

4.2 Oblivious Shuffle Based on Sorting

Let $y_1, \ldots, y_n \in \mathbb{Z}_N$ be a random permutation of the set $\{1, \ldots, n\}$. Then we can implement shuffle by obliviously sorting the pairs (x_i, y_i) according to the second element. In such a shuffle phase, a miner $\mathcal{P}_i$ generates and shares values $y_1, \ldots, y_n$ and then all miners use a modified sorting network consisting of compare-exchange gates

$$\textsc{CompEx}((x_i, y_i), (x_j, y_j)) = \begin{cases} (x_i, y_i), (x_j, y_j) & \text{if } y_i \leq y_j \ , \\ (x_j, y_j), (x_i, y_i) & \text{if } y_i > y_j \ , \end{cases}$$

for oblivious relocation of elements. The CompEx-gate can be securely implemented by combining secure multiplication and comparison operations, e.g. we can define the first output element as

$$([\![y_i]\!] \leq [\![y_j]\!]) \cdot [\![x_i]\!] + ([\![y_i]\!] > [\![y_j]\!]) \cdot [\![x_j]\!] \ .$$

As a result, each CompEx block can be executed by doing $\mathrm{O}(\tau_{\mathrm{st}} + \tau_{\mathrm{mul}})$ rounds. Similarly to the first protocol, several shuffle phases are needed to hide the permutation from potentially adversarial miners.

The efficiency of the resulting shuffle protocol depends on the complexity of the sorting network. For instance, randomised shell sort [26] has $\mathrm{O}(\log n)$ layers and $\mathrm{O}(n \log n)$ CompEx blocks. Thus, the shuffle protocol has $\Theta(m(\tau_{\mathrm{mul}} + \tau_{\mathrm{sl}}) \log n)$ rounds and communication complexity $\Theta(mn \operatorname{poly}(\log n))$, where the term $\operatorname{poly}(\log n)$ captures the asymptotic growth of communication in base protocols.

There are alternative methods for generating shares of $y_1, \ldots, y_n$. In particular, note that the modified sorting network provides a random shuffle also when $y_1, \ldots, y_n$ are randomly chosen form $\mathbb{Z}_N$ and there are no collisions $y_i = y_j$ for $i \neq j$. If $n < \sqrt{N}$, the probability of such collisions is less than $\frac{1}{2}$ by the birthday bound. The generation of random elements is straightforward. Each miner has to generate n random elements form $\mathbb{Z}_N$ and after that miners can sum the shares of respective elements locally.

Naïve detection of collision requires $\Theta(n^2)$ comparisons and thus does not reduce communication complexity. However, if we first sort $y_1, \ldots, y_n$ and then only compare the adjacent elements in the resulting vector $y'_1, \ldots, y'_n$, the communication complexity can be reduced to $\mathrm{O}(n \log n)$. Also, note all test results $[\![y'_i]\!] = [\![y'_{i+1}]\!]$ can be public. If all tests are negative then the combination is valid and we can proceed. If some test is positive then information is leaked about $\boldsymbol{y}$, but then we must anyway restart the procedure.

Inputs: A database of shares $[\![x_1]\!], \ldots, [\![x_n]\!]$ and a potential adversarial coalition $\mathcal{A} \subsetneq \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$.
Output: A database of shares $[\![x_{\pi(1)}]\!], \ldots, [\![x_{\pi(n)}]\!]$ for a random permutation π unknown to the coalition $\mathcal{A}$.
Clarifying remark: Steps starting with $\otimes$ should be omitted in the semihonest setting.

Mixing phase. Let $\mathcal{C} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\} \setminus \mathcal{A}$ be the complement group and let $[\![x_k]\!]_i$ denote the ith share of x_k.

1. Each miner $\mathcal{P}_i \in \mathcal{A}$ additively secret shares its share vector between the miners of $\mathcal{C}$, i.e., each miner $\mathcal{P}_j \in \mathcal{C}$ gets a share u_{kj} such that the shares $(u_{kj})_{j\in\mathcal{C}}$ sum up to $[\![x_k]\!]_i$ for $k \in \{1, 2, \ldots, n\}$.

$\otimes$ $\mathcal{P}_i$ commits $(u_{ij})_{j\in\mathcal{C}}$ to all. CTP, CMP and CSP are executed to verify correctness and consistency.

2. All miners in the group $\mathcal{C}$ locally compute additive shares for the database elements $x_1, \ldots, x_n$, i.e., each miner $\mathcal{P}_j \in \mathcal{C}$ holds a share v_{kj} for element x_k, s.t. shares $(v_{kj})_{j\in\mathcal{C}}$ sum up to x_k for $k \in \{1, 2, \ldots, n\}$.

$\otimes$ All miners use linearity to compute locally commitments to $(v_{kj})_{j\in\mathcal{C}}$.

3. All miners in the group $\mathcal{C}$ agree on a random permutation π and reorder their shares according to π.
4. Each miner $\mathcal{P}_j \in \mathcal{C}$ uses original secret sharing to share the shuffled shares between all miners, i.e., miners obtain a matrix of shares $([\![v_{\pi(k),j}]\!])_{j\in\mathcal{C},k\in\{1,2,\ldots,n\}}$ at the end of this step.

$\otimes$ Zero-knowledge proofs for shuffle correctness are run to verify that all parties $\mathcal{P}_i \in \mathcal{C}$ followed the protocol.

$\otimes$ CSP are executed to restore commitments to the shares of $(v_{\pi(k),j})_{j\in\mathcal{C},k\in\{1,2,\ldots,n\}}$.

5. All miners locally add shares $([\![v_{\pi(k),j}]\!])_{j\in\mathcal{C}}$ to obtain $[\![x_{\pi(1)}]\!], \ldots, [\![x_{\pi(n)}]\!]$.

Protocol 3: Verifiable shuffle protocol VerShf that is oblivious for a coalition $\mathcal{A}$

As $\boldsymbol{y}$ consists of random values noting is leaked about the inputs (the failures can be adequately simulated). For $n < \sqrt{N}$, testing k candidates gives us one collision-free permutation with probability at least $1 - \frac{1}{2^k}$. As all candidate sequences can be generated and tested in parallel, the number of rounds is independent of k. Finally, note that comparison results in the algorithm can be cached so that final run of the modified sorting network requires only multiplication operations.

4.3 Resharing Based Oblivious Shuffle for Semihonest Setting

The suffle phase can be viewed as a hide and seek game, where the aim of the hider set $\mathcal{C}$ is to shuffle the database in a such way that the seeker set $\mathcal{A}$ learns nothing about the permutation. Protocol 3 depicts a setting where the seekers first transfer their shares to the hiders so that the database is secret shared only between the members of $\mathcal{C}$. Next, hiders agree on a permutation π and reorder their shares accordingly. Then the secret sharing is extended to all miners. The protocol is secure and achieves its goal provided that: (a) corrupted parties in the hider set $\mathcal{C}$ cannot recover secrets; (b) shares are extended so that seekers learn nothing about shared values. By repeating this shuffle phase for every maximal tolerable adversarial coalition $\mathcal{A}$, we get a secure oblivious shuffle provided that all sub-protocols remain secure.

Lemma 1. *Let $\mathcal{A}$ be such a coalition that the complement set cannot be corrupted. If assumptions of secret sharing are fulfilled, then the protocol* VerShf *is secure in the semihonest model.*

Proof (Sketch). As the complement group cannot be entirely corrupted, the first step in the mixing phase reveals no information and can be easily simulated. The second, third and fifth step create no communication and thus are trivial to simulate. The fourth step is simulatable due to the properties of the original secret sharing. The claim follows, as the output is guaranteed to be correct in the semihonest model. □

Theorem 4. *If a secret sharing scheme satisfies the Q2 condition, there exists a secure oblivious shuffle protocol with $\mathrm{O}(2^m/\sqrt{m})$ rounds and communication complexity $\mathrm{O}(2^m m^{3/2} n \log n)$ in the semihonest model.*

Proof. Let us repeat the VerShf protocol sequentially for each maximal adversarial coalition $\mathcal{A}$. Then each protocol instance is secure as the condition Q2 guarantees that the complement of $\mathcal{A}$ cannot be corrupted. As at least one permutation remains hidden from an adversarial coalition, the protocol indeed implements oblivious shuffle. The round complexity estimate follows from the consideration that maximal adversarial coalitions form an antichain in the partially ordered set of all the subsets of the m-element set of miners.

Sperner's classical result from 1928 states that there are up to $\binom{m}{\lfloor m/2 \rfloor}$ elements in such an antichain, and on the other hand it is known that $\binom{m}{\lfloor m/2 \rfloor} = \Theta(2^m/\sqrt{m})$. The claim concerning the communication complexity follows, since each sub-protocol requires $\Theta(n)$ resharing operations and $\mathrm{O}(n \log n)$ bits to generate a random permutation and each resharing operation requires $\mathrm{O}(m^2)$ communication for fixed N. □

The protocol described above is asymptotically optimal if we consider only the asymptotic dependency on the database size. Sampling a random permutation requires $\Theta(n \log n)$ bits and thus the communication cannot be decreased further. Moreover, for small number of parties, say $m \leq 10$, the protocol is very efficient as there are no expensive multiplication and comparison operations.

Efficiency tweaks and implementation results. The first possible optimization can be obtained noting that it is not always necessary to consider the full complement of an adversary set as the set of hiding miners. For example, consider the threshold adversary setting, where the maximal adversary coalitions have t members with m strictly larger than $2t+1$. In this case it is enough to select hiding sets having $t+1$ elements which would then work against several different adversary sets, allowing us to achieve a reduced number of rounds. For example, if we have $t = 2$ then with $m = 5$ miners we would need 10 phases of computations. At the same time, increasing the number of miners to $m = 6$, six phases are sufficient, since the hider sets can be chosen to be $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_4\}$, $\{\mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_5\}$, $\{\mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_6\}$, $\{\mathcal{P}_4, \mathcal{P}_5, \mathcal{P}_1\}$, $\{\mathcal{P}_5, \mathcal{P}_6, \mathcal{P}_2\}$, $\{\mathcal{P}_6, \mathcal{P}_1, \mathcal{P}_3\}$. It is easy to verify that for no adversary set of up to two elements they have access to all of the permutations and that 6 active sets (i.e. 6 phases) is the minimum achievable for $t = 2$ and $m = 6$. Finding optimal round and communication complexity for any t, m remains an open combinatorial problem.

In practice, communication channels between miner nodes are commonly implemented using authenticated encryption and thus information-theoretical security is unachievable. Hence, the security level does not decrease if the group $\mathcal{C}$ agrees on a short random seed and later uses a secure pseudorandom generator to stretch locally into $\mathrm{O}(n \log n)$ bits needed for a random permutation. In particular, we can generate an array $f_{\mathsf{sk}}(1), \ldots, f_{\mathsf{sk}}(n)$ by applying a pseudorandom permutation f indexed by a seed sk. By sorting the array, we get a permutation that is computationally indistinguishable from a random permutation. We implemented the corresponding shuffle protocol for three miners by using 128-bit AES as f. For each mixing phase, $\mathcal{P}_a$ and $\mathcal{P}_b$ forming the group $\mathcal{C}$ exchanged 128-bit random sub-keys sk_a and sk_b and set $\mathsf{sk} \leftarrow \mathsf{sk}_a \oplus \mathsf{sk}_b$. The protocol was implemented into SHAREMIND framework using C++ programming language. The computing parties and the controller node ran on servers having two Intel Xeon X5670 2.93GHz processors and 48GB of RAM each. The servers were using Debian OS and were connected by gigabit Ethernet. Figure 2 shows the times required for oblivious shuffle of databases consisting of $10^3, \ldots, 10^7$ additively shared 32-bit integers.

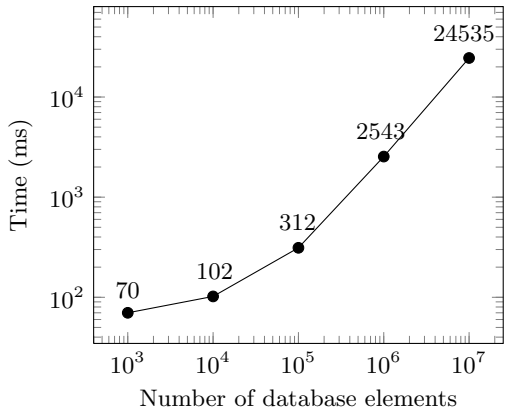

Fig. 2. Performance of oblivious shuffle in three-party setting

4.4 Resharing Based Oblivious Shuffle for Malicious Setting

The protocol described above can be used also in the malicious model provided that we can force universal consistency checks: (a) all resharing steps are correct; (b) the permutation π is indeed randomly sampled; (c) hiders permute their shares according to the permutation. To achieve such kind of protection, we follow the standard two-level secret sharing technique [19], where all original shares are secret shared. That is, any share $[\![x]\!]_i$ owned by $\mathcal{P}_i$ is always secret shared between all miners. Such a setup simplifies zero-knowledge correctness proofs, since $\mathcal{P}_i$ can more easily prove that he or she computed output shares $[\![v]\!]_i$ from the input shares $[\![u]\!]_i$. The second level secret sharing is commonly referred to as commitment, as it satisfies both perfect binding and hiding properties. As shown in [19], 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 commitment of 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 $c = ab$. In order to achieve security against active adversaries, share computing frameworks use two-level secret sharing by default. Moreover, for any robust secret sharing scheme, the second layer can be added on demand. Miners just have to commit their shares. Although a maliciously corrupted miner $\mathcal{P}_i$ can provide incorrect sharings of $[\![x]\!]_i$, the number of correctly shared shares is sufficient to correctly recover the original secret x. If needed, miners can emulate recovery procedure with commitments to detect which shares were incorrectly committed.

Correctness proofs for the resharing steps. As $\mathcal{P}_i$ commits values $(u_{kj})_{j\in\mathcal{C}}$ in the first step of Protocol 3, miners can compute $[\![x_k]\!]_i - \sum_{j\in\mathcal{C}}[\![u_{kj}]\!]$ and open it. If the result is zero, the shares were correctly formed. The result leaks no information, as the output is always zero for honest miners. By employing CTP, these commitments can be transferred to the intended recipients who are forced to correctly recommit u_{kj}. Verification of the second step is straightforward, since the reconstruction coefficients are public and all parties can locally manipulate shares of shares to get shares of the corresponding linear combinations.

Unbiased sampling of randomness. As the condition Q3 is not satisfied for the set $\mathcal{C}$, hiders cannot agree on the random permutation without outsiders. Hence, all miners must engage in a secure coin-tossing protocol to generate necessary random bits for the permutation π. For instance, a random element of $\mathbb{Z}_N$ can be generated if all parties share random elements of $\mathbb{Z}_N$ and the resulting shares are added together. Next, everybody broadcasts their shares to the set of hiding participants $\mathcal{C}$. To sample a random permutation, the same protocol can be run in parallel to generate enough random bits.

Correctness proof for the local shuffle step. The correctness proof hinges on the fact that miners have commitments to all shares of $\mathcal{P}_j$. After the second step, miners obtain commitments to additive shares $v_{1j}, \ldots, v_{nj}$ and during the fourth step miners receive commitments to permuted shares $v_{\pi(1)j}, \ldots, v_{\pi(n)j}$. Hence, if all miners $\mathcal{P}_j \in \mathcal{C}$ prove that the shares were indeed obtained by applying the permutation π, the CSP protocol assures that the resulting double-level sharing of $v_{\pi(1)j}, \ldots, v_{\pi(n)j}$ is also correct.

Protocol 4 depicts a zero-knowledge protocol for a slightly abstracted setting where $\mathcal{P}_j$ has to prove that he or she has secret shared databases $\boldsymbol{x}$ and $\boldsymbol{y}$ so that $\boldsymbol{y} = \pi(\boldsymbol{x})$ for a permutation π known to the hider set $\mathcal{C}$. At the end of the protocol, honest parties from both sets learn whether the sharing was correct or not. The soundness error of the ZkShf protocol can be efficiently amplified: any soundness error ε can be achieved by running $\lceil \log_2 1/\varepsilon \rceil$ instances of the protocol in parallel as the protocol is universally composable.

Theorem 5. *Let $\mathcal{A}$ be a maximal adversarial set. If a secret sharing scheme satisfies the Q3 condition then the protocol* ZkShf *is a perfect five round zero-knowledge proof with soundness error $\frac{1}{2}$.*

Proof (Sketch). The protocol has five rounds, since each protocol step can be implemented in a single round. For correctness note that if $\mathcal{P}_j$ is honest then $z_i = y_{\sigma(i)} = x_{\pi(\sigma(i))}$ for $i \in \{1, \ldots, n\}$. As the set $\mathcal{A}$ is maximal

Prover: A prover is a miner $\mathcal{P}_j$ in the hider set knowing $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.
Helpers: Miners in the hider set $\mathcal{C}$ know a permutation π such that $y_1 = x_{\pi(1)}, \ldots, y_n = x_{\pi(n)}$.
Common inputs: Miners have shares $[\![x_1]\!], \ldots, [\![x_n]\!], [\![y_1]\!], \ldots, [\![y_n]\!]$.

1. All miners engage secure coin-tossing protocol to fix a random permutation σ.
   The resulting bits are revealed to all miners in the hider set so that they can learn σ.
2. The prover $\mathcal{P}_j$ computes $z_1 = y_{\sigma(1)}, \ldots, z_n = y_{\sigma(n)}$ and shares them.
3. All parties engage in a secure coin-tossing protocol to get a public random bit b.
4. (a) If $b = 0$ then all miners in the hider set broadcast σ. Each miner reconstructs σ.
   All miners compute shares of $[\![z_1]\!] - [\![y_{\sigma(1)}]\!], \ldots, [\![z_n]\!] - [\![y_{\sigma(n)}]\!]$ and broadcast the results.
   (b) If $b = 1$ then all miners in the hider set broadcast $\pi \circ \sigma$. Each miner reconstructs $\pi \circ \sigma$.
   All miners compute shares of $[\![z_1]\!] - [\![x_{\pi(\sigma(1))}]\!], \ldots, [\![z_n]\!] - [\![x_{\pi(\sigma(n))}]\!]$ and broadcast the results.
5. Proof fails if any of the revealed values is non-zero, otherwise miners accept the proof.

Protocol 4: Zero-knowledge proof ZkShf for shuffle correctness

adversarial set and the condition Q3 holds, the condition Q2 must hold in the hiding set $\mathcal{C}$. Consequently, each miner can correctly reconstruct permutations σ or $\pi \circ \sigma$ in the fourth step. By the condition Q2, honest coalition is strictly larger than any adversarial set in $\mathcal{C}$. As the replies sent by honest parties coincide, each party can detect the correct answer. As the adversarial coalition cannot alter the reconstructed values by changing their shares, the revealed values will always be zeroes. For the soundness claim, assume that the prover cheats and there exists $y_j \neq x_{\pi(j)}$. Let $i = \sigma^{-1}(j)$. Then $z_i \neq y_j$ or $z_i \neq x_{\pi(j)}$ cannot simultaneously succeed. As malicious parties cannot alter the end results, the prover is bound to fail with probability at least $\frac{1}{2}$. For the simulatability, note that the adversarial coalition cannot bias the outputs of coin-tossing protocols. Hence, the permutations σ and $\pi \circ \sigma$ are random permutations in isolation. When no hider is corrupted, we can just send a random permutation to simulate the beginning of the fourth step. Otherwise, we have to extract π form the inputs of the corrupted hider and simulate permutation generation according to the protocol. The public opening of the shares is trivial to simulate, as the result is known to be zero. More precisely, note that the simulator has access to the input shares of the corrupted parties (i.e., the shares of $\boldsymbol{x}$ and $\boldsymbol{y}$) and thus can augment them to be shares of zero. By emulating the protocol with these dummy shares, the share distribution in the opening phase is guaranteed to coincide with the real protocol. □

Final security claim. As in the semihonest setting, the final shuffle protocol can be obtained by repeating the VerShf protocol for every maximal tolerable adversarial coalition $\mathcal{A}$. The correctness proofs assure that honest miners detect all deviations form the protocol and catch the corresponding culprit. After each incident, the culprit must be excluded form the computations and the corresponding shuffle phase must be restarted. As the number of malicious parties is limited and the exclusion does not invalidate Q3 condition, honest miners can always complete the protocol.

Theorem 6. *If a secret sharing scheme satisfies the condition Q3, there exists a secure oblivious shuffle protocol with* $\mathrm{O}(2^m/\sqrt{m})$ *rounds and communication complexity* $\mathrm{O}(2^m m^{3/2} n \log n)$ *in the malicious model.*

5 Efficiency Tweaks for Generic Oblivious Transfer

Since multiplication requires small constant number of rounds (e.g. 1 in case of Sharemind [9]), Theorem 1 implies that the round complexity of oblivious transfer is essentially equal to the round complexity of equality testing. In most frameworks, equality test is built on top of the generic bit-decomposition techniques by Damgård et al [21]. For elements of $\mathbb{Z}_N$, the resulting protocols have quite large overhead and non-constant round complexity $\mathrm{O}(\log \log N)$. In the following, we present several tweaks that address this issue.

5.1 Equality Testing Without Bit Decomposition

Let $\ell = \lceil \log(n+1) \rceil$ the bit length of n. Then the bit decomposition step can be circumvented if the client shares individual bits $i_\ell, \ldots, i_1$ of the index i in GenOt. Equality tests can be computed as products

$$([\![i]\!] = j) = \bigwedge_{k=1}^{\ell} ([\![i_k]\!] = j_k) = \prod_{k=1}^{\ell} ([\![i_k]\!] = j_k) \ . \tag{2}$$

Since one bit in comparison protocols is public, we can simply set

$$([\![i_k]\!] = j_k) = \begin{cases} [\![i_k]\!], & \text{if } j_k = 1 \ , \\ 1 - [\![i_k]\!], & \text{if } j_k = 0 \ . \end{cases}$$

without spending any extra rounds. The resulting protocol is very efficient, as it has only $\log n$ multiplications and the round complexity can be reduced to $\mathrm{O}(\log \log \log n)$ by balanced evaluation of the product.

However, this protocol is not secure against malicious clients unless miners verify in zero knowledge that the inputs $[\![i_\ell]\!], \ldots, [\![i_1]\!]$ are indeed bits. For rings $\mathbb{Z}_{p^t}$, the latter is straightforward, as the equation $x(x-1)$ has only two solutions $x = 0$ and $x = 1$. Hence, miners can simply publicly test if $i_k(i_k - 1) = 0$ for $k \in \{1, \ldots, \ell\}$. For arbitrary rings $\mathbb{Z}_N$, the equation $x(x-1) = 0$ can have non-trivial solutions and we must use alternative methods. For instance, the client can create additive shares over $\mathbb{Z}_2$. Then miners must convert the share domain from $\mathbb{Z}_2$ to $\mathbb{Z}_N$. For that, each miner first shares its share over $\mathbb{Z}_N$ and then resulting m shares are XOR-ed together. As each XOR operation takes one multiplication and we can parallelize evaluation, the total round complexity is $\mathrm{O}(\tau_{\mathrm{mul}} \log m)$ for m miner nodes. In general, conversion from $\mathbb{Z}_M$ to $\mathbb{Z}_N$ can be done by sharing the shares over $\mathbb{Z}_N$ and then emulating the reconstruction procedure with shares. The latter requires a secure modular reduction protocol for public M, which is usually even more complex than bit decomposition. Hence, splitting the input into bits is more efficient only if N is a prime power or reconstruction procedure for shares over $\mathbb{Z}_2$ is simple enough to do it implicitly over $\mathbb{Z}_N$.

5.2 Recursive Equality Testing

For further optimisations, note that ℓ-element conjunction can be represented by an equality test of $\lceil \log(\ell + 1) \rceil$ bit strings provided that all inputs b_k are binary:

$$\bigwedge_{k=1}^{\ell} [\![b_k]\!] = \left(\sum_{k=1}^{\ell} [\![b_k]\!] = \ell \right). \tag{3}$$

The right hand side of Equation (3) again requires comparing a secret shared value to a public one, and can hence be computed by Equation (2), leading to double recursion. Hence, equality testing can be reduced to testing equality of logarithmically shorter strings. The latter, however, cannot be done without bit-decomposition. Hence the round complexity must exceed the round complexity of bit decomposition. However, the overhead is asymptotically very small, being limited to the factor of iterated logarithm $\log^* \ell$ (see [18]).

Theorem 7. *Let τ_{bd} be the round complexity of a bit decomposition protocol for the ring $\mathbb{Z}_N$. Then there exists a protocol for equality test for ℓ-bit numbers and conjunction for ℓ terms with $\mathrm{O}(\tau_{\mathrm{bd}} \log^* \ell)$ rounds.*

Proof. Equation (3) assures that high-degree conjunction for ℓ terms can be implemented with the same round-complexity as $\lceil \log \ell \rceil$-bit equality test. By adding shares of $b_1, \ldots, b_\ell$ and then splitting the sum $b_1 + \ldots + b_\ell$ back to bits, we reduce ℓ-element conjunction to a conjunction of $\log \ell$ terms with the cost of $\mathrm{O}(\tau_{\mathrm{bd}})$ rounds, as comparison of individual bits is a constant rounds operation. By repeating the recursion as long as possible we get a protocol for conjunction with $\mathrm{O}(\tau_{\mathrm{bd}} \log^* \ell)$ rounds. The claim follows, as Equation (2) assures that ℓ-bit equality test can be reduced to conjunction of ℓ terms. □

From a theoretical viewpoint, the result is not very impressive, as Damgård *et al* [21] gave constant round protocols for bit decomposition, conjunction, disjunction, and for all comparison operations when the message space is a finite field $\mathbb{Z}_p$. However, the corresponding constants are rather high – over 50 for equality and conjunction. As the bit-decomposition can be easily done in $\log \log p$ rounds, the solution is more efficient for most practical instance sizes $\ell \leq 2^{128}$. The result is even more appealing in the case of rings $\mathbb{Z}_{p^t}$, where constant round constructions are not known.

5.3 Round Complexity versus Communication Complexity

Oblivious transfer protocols based on high-degree conjunction are very round efficient. However, the communication complexity between the miner nodes is rather high and the time needed to deliver protocol messages starts to dominate for large enough databases. For instance, oblivious transfer without bit decomposition contains $\mathrm{O}(n \log n)$ multiplication operations and thus the asymptotic running time is $\Theta(n \log n)$.

As a first step towards lower communication complexity note that the right hand side of Equation (1) can be viewed as a multivariate polynomial with arguments $i_1, \ldots, i_\ell$, where $i_1, \ldots, i_\ell$ are the bits of index i:

$$f(i_1, \ldots, i_\ell) = \sum_{j=1}^{n} \prod_{k=1}^{\ell} [(1 - j_k)(1 - i_k) + j_k i_k] \cdot x_j = \sum_{\mathcal{K} \subseteq \{1,\ldots,\ell\}} \alpha_{\mathcal{K}} \cdot \prod_{k \in \mathcal{K}} i_k \ .$$

Here the coefficients of the monomials are linear combinations

$$\alpha_{\mathcal{K}}(x_1, \ldots, x_n) = \alpha_{\mathcal{K},1} x_1 + \cdots + \alpha_{\mathcal{K},n} x_n$$

with public constants $\alpha_{\mathcal{K},1}, \ldots, \alpha_{\mathcal{K},n} \in \{-1, 0, 1\}$. For example, for the three element database $\{x_1, x_2, x_3\}$ we have

$$f(i_1, i_2) = x_1 i_1 + x_2 i_2 + (x_3 - x_2 - x_1) i_1 i_2 \ .$$

By construction, $\ell = \lceil \log(n + 1) \rceil$ and thus the number of nonzero coefficients $\alpha_{\mathcal{K}}$ is $\Theta(n)$. Moreover, as the shares of $\alpha_{\mathcal{K}}$ are locally computable, only $\Theta(n)$ monomials must be securely computed and multiplied by $\alpha_{\mathcal{K}}$ for each query. Naïvely, these can be computed by doing $\Theta(n)$ multiplication operations in κ rounds. That is, we first compute all monomials for one element sets $\mathcal{K}$, then for two element sets and so on. This solution is not round optimal, since we can compute monomials of k elements in $\lceil \log k \rceil$ rounds by balancing the expression tree. The same holds also if we compute all monomials in parallel. Hence, the online complexity of the corresponding oblivious transfer protocol is $\Theta(n)$ multiplication operations and $\Theta(\tau_{\mathrm{mul}} \log \log n)$ rounds. The round complexity can be further reduced by using high-degree conjunction protocols to evaluate the monomials, as multiplication and conjunction are equivalent for bits.

In particular, by using a single double reduction step we can reduce the round count to $\mathrm{O}(\tau_{\mathrm{mul}} \log \log \log n + \tau_{\mathrm{bd}})$ rounds. That is, we first form all possible shares of sums $i_{k_1} + \cdots + i_{k_s}$ and then use bit decomposition protocol to reduce the problem back to the conjunction. The asymptotic running time does not decrease since there are $\Theta(n)$ share additions to carry out. In practice, the running time decreases significantly as addition is much faster than multiplication.

6 Conclusions and Future Work

In this paper, we have proposed several round-efficient protocols for oblivious database manipulation over secure MPC in both semihonest and malicious model, including oblivious transfer and oblivious shuffle of secret shared databases. We also presented an oblivious filtering protocol and selection protocols as possible applications of these techniques.

For oblivious transfer, we have presented several versions with round complexity between $\mathrm{O}(\log \log n)$ and $\mathrm{O}(\log \log \log n)$. For oblivious shuffle, we have proposed three alternative implementations with different

trade-offs between the round and communication complexity. Which one of them is the fastest for a given application and deployment scenario, is a non-trivial question requiring further research and benchmarking. We have also provided several possible tweaks for optimization of the presented primitives, but several questions are left open. For example, finding out optimal hider set constructions for possible adversarial structures is an interesting combinatorial problem of its own. As a useful by-product of the tweaking, we obtained an efficient protocol for equality testing based on double recursion with bit decomposition.

In the current paper, we presented initial performance results proving that oblivious database shuffle is efficient for relatively large databases (roughly 25 seconds for 10^7 elements). Further improvements can be achieved by optimizing the implementation, however, these improvements remain the subject for future development as well.

Acknowledgments

This research was supported by Estonian Science Foundation grant #8058, the European Regional Development Fund through the Estonian Center of Excellence in Computer Science (EXCS), and Estonian ICT doctoral school.

References

1. VIFF documentation. `http://viff.dk/doc/index.html`.
2. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA*. ACM, 1988.
3. SecureSCM. Technical report D9.1: Secure Computation Models and Frameworks. `http://www.securescm.org`, July 2008.
4. William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135. Springer Berlin / Heidelberg, 2001.
5. Amos Beimel and Yoav Stahl. Robust Information-Theoretic Private Information Retrieval. *J. Cryptology*, 20(3):295–321, 2007.
6. Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266, New York, NY, USA, 2008. ACM.
7. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC* [2], pages 1–10.
8. Dan Bogdanov, Sven Laur, and Jan Willemson. 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.
9. 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`.
10. Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. All-or-Nothing Disclosure of Secrets. In Andrew M. Odlyzko, editor, *CRYPTO*, volume 263 of *Lecture Notes in Computer Science*, pages 234–238. Springer, 1986.
11. Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *USENIX Security Symposium*, pages 223–239, Washington, DC, USA, 2010.
12. Jan Camenisch and Anna Lysyanskaya. A Formal Treatment of Onion Routing. In Shoup [34], pages 169–187.
13. Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/`.
14. Octavian Catrina and Amitabh Saxena. Secure Computation with Fixed-Point Numbers. In Radu Sion, editor, *Financial Cryptography and Data Security*, volume 6052 of *LNCS*, pages 35–50. Springer, 2010.
15. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *STOC* [2], pages 11–19.
16. David L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, February 1981.

17. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private Information Retrieval. *J. ACM*, 45(6):965–981, 1998.
18. Thomas H. Cormen, Charles E. Leiserson, and Ronald R. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1992.
19. Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 316–334, Berlin, Heidelberg, 2000. Springer-Verlag.
20. Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Secure Multiparty Computation (Book Draft). Available form `http://www.daimi.au.dk/~ivan/mpc-book.pdf`, 2010.
21. Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.
22. 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.
23. Ivan Damgård and Yuval Ishai. Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. In Shoup [34], pages 378–394.
24. Juan A. Garay. Efficient and Universally Composable Committed Oblivious Transfer and Applications. In Naor [31], pages 297–316.
25. Martin Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, February 2010.
26. Michael T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1262–1277, 2010.
27. Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *CCS '10: Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462, New York, NY, USA, 2010. ACM.
28. Martin Hirt and Ueli M. Maurer. Player Simulation and General Adversary Structures in Perfect Multiparty Computation. *Journal of Cryptology*, 13(1):31–60, 2000.
29. Lior Malka and Jonathan Katz. VMCrypt – Modular Software Architecture for Scalable Secure Computation. Cryptology ePrint Archive, Report 2010/584, 2010. `http://eprint.iacr.org/`.
30. 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.
31. Moni Naor, editor. *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*. Springer, 2004.
32. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure Reactive Systems. Technical Report 3206 (#93252), IBM Research Division, Zürich, May 2000.
33. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
34. Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.
35. Douglas Wikström. A Universally Composable Mix-Net. In Naor [31], pages 317–335.
36. Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS*, pages 60–164, 1982.