Fast and Secure Oblivious Stable Matching over Arithmetic Circuits

Arup Mondal

Department of Computer Science Ashoka University Haryana, India arup.mondal_phd19@ashoka.edu.in Priyam Panda Department of Computer Science Ashoka University Haryana, India priyam.panda@alumni.ashoka.edu.in Shivam Agarwal Department of Computer Science Georgia Institute of Technology Georgia, USA s.agarwal@gatech.edu

Abdelrahaman Aly Cryptography Research Centre Technology Innovation Institute UAE abdelrahaman.aly@tii.ae

Abstract—The classic stable matching algorithm of Gale and Shapley (American Mathematical Monthly '69) and subsequent variants such as those by Roth (Mathematics of Operations Research '82) and Abdulkadiroglu et al. (American Economic Review '05) have been used successfully in a number of real-world scenarios, including the assignment of medical-school graduates to residency programs, New York City teenagers to high schools, and Norwegian and Singaporean students to schools and universities. However, all of these suffer from one shortcoming: in order to avoid strategic manipulation, they require all participants to submit their preferences to a trusted third party who performs the computation. In some sensitive application scenarios, there is no appropriate (or cost-effective) trusted party. This makes stable matching a natural candidate for secure computation. Several approaches have been proposed to overcome this, based on secure multiparty computation (MPC), fully homomorphic encryption, etc.; many of these protocols are slow and impractical for real-world use.

We propose a novel primitive for privacy-preserving stable matching using MPC (i.e., arithmetic circuits, for any number of parties). Specifically, we discuss two variants of oblivious stable matching and describe an improved oblivious stable matching on the random memory access model based on lookup tables. To explore and showcase the practicality of our proposed primitive, we present detailed benchmarks (at various problem sizes) of our constructions using two popular frameworks: SCALE-MAMBA and MP-SPDZ.

1. Introduction

The Stable Matching Problem (SMP) was first formalized in the 1960s by Gale and Shapley [22]. SMP has seen extensive real-world use in scenarios that require the clearing of two-sided markets, including NRMP – the assignment of graduating medical students to residency programs¹, the assignment of New York City teenagers to high schools [1], and the assignment of Norwegian and Singaporean students to schools and universities [50]. The SMP involves two sets of agents, such as schools and students, or employees and factories, or, the most common version, women and men². It is assumed that each participant has a predefined, ordered, "preference" list (possibly empty) of participants belonging to the other set: so, every woman will have a list of men in her order of preference (vice versa for the men).

A valid solution must match men and women to produce a list of pairs (man, woman) such that no man and woman belonging to different pairs would both be willing to break their current pairing in favour of each other. More formally, we have a set M of male suitors, a set W of female reviewers (with |M| = |W| and $M \cap W = \emptyset$), and a strict preference ordering (or "ranking") of W (resp. M) for each $m \in M$ (resp. $w \in W$); we say that "m prefers w_1 to w_2 " if m would rather marry w_1 than w_2 . A solution comprises a one-to-one correspondence between M and W which is *stable*, i.e., it contains no pairs (m_1, w_1) and (m_2, w_2) such that such that m_1 prefers w_2 to w_1 and w_2 prefers m_1 to m_2 .

In the general SMP (defined in Sec. 3.1), the two sets of participants (*suitors* who propose matches and *reviewers* who may or may not accept the proposals) need not be of the same size, and each participant may rank only a proper subset of its potential "matches". Gusfield and Irving [28] provide a comprehensive overview of stable matching algorithms.

Clearly, agents on both sides of these markets have legitimate concerns about the privacy of their rankings and the integrity of the computation, but straightforward implementations of stable-matching algorithms may leak participants' preferences or the history of matches made and broken on the path to a stable solution. Therefore, stable matching is a natural candidate for secure multiparty computation (MPC), *i.e.*, computation by a protocol that leaks no information about the participants' inputs (except that which is logically implied by the output), even in the presence of malicious (or "byzantine") participants, who may deviate from the protocol specification in order to gain information about other participants' inputs or to

Debayan Gupta Department of Computer Science Ashoka University Haryana, India debayan.gupta@ashoka.edu.in

^{1.} National Residency Matching Program, http://www.nrmp.org.

^{2.} For simplicity, we will use this example throughout this paper.

manipulate the output. Previous works on MPC for stable matching include [18], [20], [21], [26], [35], [42], [54].

Stable matching procedures exhibit additional useful properties. Roth [43] proved that stable matchings always exist and that there will always be a stable arrangement weakly preferred by the men, and that there will always be a stable arrangement weakly preferred by the women. The algorithm proposed by Gale and Shapley will always output a stable matching weakly preferred by the men; that is, each man will be at least as well off under the Gale-Shapley assignment as under any other. We say that such an assignment is *male-optimal*. Abdulkadiroglu et al. [1] and Roth [43] leveraged this property in designing the New York City Match and revising the NRMP to favour students and residents.

Prior work has shown that, in the traditional setting in which all inputs are known by all players, stable matching procedures are susceptible to manipulation by strategic players [43], [50]. Although members of the suitor group have no incentive to misrepresent their preferences [44], and members of the reviewing group never benefit from misrepresenting their first choices, a strategic player in the reviewing group may misrepresent his or her preferences to change the outcome and can sometimes switch the stable arrangement from suitor-optimal to revieweroptimal [43], [50]. For example, if partial lists are allowed, a reviewer may simply list as "unacceptable" each suitor that he or she prefers less than his or her revieweroptimal suitor. Reviewers may also cause "faux stability" by misrepresenting their preferences; the algorithm might output an assignment that is stable with respect to the inputs it receives but unstable with respect to the true preferences of the players. By enforcing input privacy, we leave reviewers in a position in which they may still be able to change the outcome by misrepresenting their true preferences but will not be able to change it in a predictable way; because lying about their preferences may wind up making the outcome worse for them instead of better, they are motivated simply, to tell the truth.

A naive solution to the privacy constraints presented in these cases would involve the participation of a trusted third party. Such an entity would be in charge of the execution of the algorithm, using the private inputs of the participants. In this setting, parties would submit their sealed preferences and await for the answer from such an entity. Furthermore, the parties would have to trust correctness and non-collusion from his behalf, but more importantly, finding such a party is not obvious.

The idea of using MPC to obviate a trusted third party in the context of stable matching has been previously studied, leading to the development of new privacy-preserving schemes to solve associated subproblems e.g. [18], [35], [42]. The results have been mixed, with vastly different set sizes and other settings.

1.1. Contributions

Our work centers around the design of black-box primitives for **Oblivious Stable Matching** and all the necessary supporting building blocks, in the context of multiparty protocols. This includes **a novel secret index extraction protocol** with constant cost inspired by [17], [33]. Our focus is practical/realistic implementations for generic MPC. Our constructions are mainly based on arithmetic circuits ³. In this context, we divide our contributions as:

• **Theoretical Aspects:** We provide a method to obliviously solve instances of the Stable Matching Problem via variations of the Gale-Shapley algorithm [22], and offer a security analysis of the problem. We achieve this via the use of a generic Arithmetic Black Box \mathcal{F}_{ABB} (refer to Sec. 3.5). This allows us to abstract functionality and decouple security proofs realizations, of the underlying protocols. Our protocols are information theoretic – in reality, it depends of the protocols used for their implementation.

• Technical Aspects. We benchmark our constructions using the most commonly used frameworks for the evaluation of arithmetic circuits over MPC(namely SCALE-MAMBA [4] and, MP-SPDZ [31]). Out setup considers different adversarial settings, number of parties, and latencies. We go a step further and offer technical recommendations, related to the implementation on both tools in our discussion section. Finally, we make our repositories publicly available⁴.

Besides these main points, we also include brief discussions on the state of the art and similar constructions, as well as applications.

Organization. The paper is organized as follows: Sec. 2 discusses related work. Sec. 3 provides preliminaries and technical background. Sec. 4 describes our proposed oblivious stable matching algorithm and its variants. In Sec. 5 we describe a novel protocol for private array access (and using this protocol we describe an oblivious stable matching algorithm on the random access model). Sec. 6 provides a security sketch of our proposed oblivious stable matching algorithm and its variants. Sec. 7 discusses experimental performance evaluations of our proposed algorithm, and finally, Sec. 8 concludes.

2. Related Work

Brito and Meseguer [10] propose a distributed approach to the stable marriage problem with ties and incomplete lists with the aim of keeping preference lists private for privacy reasons. They extend some specialized centralized algorithms (such as the Extended Gale-Shapley algorithm [30]) to the distributed case, using a generic distributed constraint programming model. However, their proposed method is not cost efficient.

Golle [26] first explored and proposed a secure version of the classic Gale-Shapley [22] stable matching algorithm. Golle's variant introduces semi-honest "matching authorities" and adds "dummy" men; it uses threshold homomorphic encryption and re-encryption mixnets to compute a stable matching in a privacy-preserving fashion. Privacy and correctness are guaranteed when a majority of the matching authorities are honest. Golle's protocol requires $O(n^5)$ public key operations. Soon after, Franklin et al. [20] proposed two new protocols – the first protocol

^{3.} Note that, for efficiency, our implementation rely on widely used mixed-circuit protocols like [29], [34] via [6].

^{4.} For ease of access, all of our code and experiments are available in a repository: https://github.com/StableMatch/OblivSM.

was based on an XOR secret sharing scheme and used private information retrieval to process bids, and the second protocol used garbled circuits in combination with Naor-Nissim's protocol for secure function evaluation [39]. The first protocol required $O(n^4)$ public key operations with $O(n^2)$ communication rounds and the second protocol required $O(n^4)$ computation complexity with $O(n^2)$ communication rounds. In [21] Franklin et al.'s design an efficient multiparty Look-Up Table (mLUT) protocol and discussed a design of improved secure stable matching protocol of the algorithm in [20]. However, both Golle's protocol and Franklin et al.'s protocols does not appear to be practical and has never been implemented.

Teruya and Sakuma [51] proposed a protocol building on Golle's secure stable matching protocol using additive homomorphic encryption to simplify the bidding process which offers lower communication complexity rounds, i.e., $O(n^2)$ than the Golle's protocol. They shown an implemented study of their protocol as a client-server system, using hand-held devices running on a LAN.

Keller and Scholl [35] first proposed a secure version of Gale-Shapley stable matching protocol using ORAM, and implemented their protocol using Path ORAM [49] and the SPDZ MPC protocol [16]. They reported that in the worst case of their proposed protocol with 8k pairs can be done in 1.5×10^{12} seconds but they did not provide the implementation details. Further results in this area were reported by Keller more recently [31] (see Table 7 for the results and comparison). Zahur et al. [54] present an implement study of the classical Gale-Shapley stable matching algorithm using Square-Root ORAM [55], and reported a runtime of more than 33 hours for 512 pairs of participants. The required computation and communication complexity of theirs protocol is $O(n^3 \log^2 n)$ which limits the scalability of their approach significantly.

Riazi et al. [42] proposed a provably secure stable matching protocol using Yao's Garbled Circuits [53] and introduce a sub-linear size circuit making the protocol computationally efficient, and discussed three different variations of their – a protocol without using ORAM which required $O(n^4 \log n)$ symmetric key operations, using ORAM which required $O(n^2 \log^3 n)$ symmetric key operations, and using ORAM and early termination technique (ETT) which required $O(n^2 \log^3 n)$ symmetric key operations. They shows an implementation study of their protocols for several problem sizes.

Doerner et al. [18] study the development of strategies for RAM-based secure computation by modification to the ORAM access protocol that enables efficient function application within an ORAM access. Doerner et al. designed an oblivious linked list structure that can be used when the order in which data is accessed must be hidden. Using the aforementioned techniques, they developed and evaluated the two secure stable matching algorithms. They implemented a secure version of the modified Gale-Shapley and Roth-Peranson stable matching algorithm and achieved lower asymptotic complexity than the previous design. Their secure version of Gale-Shapley algorithm performs $\Theta(n^2)$ operations upon a $\Theta(n)$ -length memory. This results in a total complexity in $\Theta(n^{2.5} \log^{1.5} n)$ when using a Secure-Root ORAM and $\Theta(n^2 \log^3 n)$ by using Circuit ORAM. However, both papers [18], [42] implementation codes are not publicly available, so we cannot provide a quantitative comparison.

We achieve an efficient method for solving secure stable matching by proposing a novel secret index extraction protocol. See Table 4 for the asymptotic complexity comparison between the existing works and our proposed methods.

3. Preliminaries and Background

In this section, we present a detailed background of the stable matching problem and cryptographic primitives upon which our proposed primitive relies to achieve secure oblivious stable matching.

3.1. The Basic Stable Matching Algorithm

In [22], Gale and Shapley showed that the stable matching problem has a solution for all cases where the number of suitors is equal to the number of reviewers.

3.1.1. Informal Description. The *suitors* propose to the *reviewers*, in a series of rounds. In each round, an unmatched suitor proposes to his most preferred reviewer to whom he has not already proposed. If the chosen reviewer is unmatched, she tells her suitor "maybe" and asks him to wait. If she already has a partner, she says "no" to the less preferred of her currently waiting suitor and her current proposer, and she answers "maybe" to the other. When no unmatched suitors exist, all reviewers accept their current partner.

Thus, at the end of each round, a reviewer has said "maybe" to her most preferred suitor and "no" to everyone else. The suitors in the round, of course, have just proposed to their most preferred reviewer.

The algorithm provides two main guarantees. First, if the set of suitors and the set of reviewers are the same size, it ensures that everybody gets matched, or married. Once a reviewer has said maybe, at no point can she be unmatched, or single, since she never says no to anyone unless she is already matched via a previous maybe. At the end of the protocol, there can be no suitor and reviewer that are not matched, since an unmatched suitor eventually proposes to everyone. (In this scenario, the reviewer, as she isn't already matched, must say maybe).

Second, it ensures that all the matches are stable. Let there exist two pairings (Alice, X) and (Y, Bob) are where *Alice* prefers *Bob* to X and *Bob* prefers *Alice* to Y. Since *Bob* prefers *Alice* to Y, he must have proposed to *Alice* before he proposed to Y. At this point, *Alice* can say either maybe or no to *Bob*:

- *Alice* only says no to *Bob* if she prefers her current match.
- If *Alice* says maybe to *Bob*, the only way in which she can be separated from *Bob* is if she finds someone she prefers more.

Thus, at the end of the protocol, if *Alice* is not matched to *Bob*, she is with someone she prefers over *Bob*. So, the two pairings (*Alice*, X) and (Y, *Bob*) cannot exist if *Alice* prefers *Bob* to X and *Bob* prefers *Alice* to Y.

A stable assignment is considered optimal for a group of suitors or reviewers if every member of the group is at least as well off under that assignment as under any other assignment. Roth [43] showed that stable assignments can always be constructed for only one of the disjoint sets involved in the matching, but not both.

3.1.2. Formal Description. We define $S = \{1, 2, ..., n\}$ to be the set of suitors such that the *i*-th suitor is *i*, and $R = \{1, 2, ..., n\}$ to be the set of reviewers such that *j*-th reviewer is *j*, in the stable matching. $M_{S,R}$ describes a stable matching as a binary relation in $S \times R$ such that $(i, j) \in M$ if $(i, j)|i \in S, j \in R$ is a pair in the assigned matching. $S' \subseteq S|S'$ only suitors that are unmatched.

$$(i, \cdot) = \{j \mid (i, j) \in M\}$$

 $(\cdot, j) = \{i \mid (i, j) \in M\}$

 p_s and p_r describes the set of preferences for each suitor and reviewer for the members of the other party, respectively. $(p_s)_{ij}$ denotes the preference of *i*-th suitor for *j*-th reviewer, and similarly $(p_r)_{ji}$ denotes the preference of *j*-th reviewer for the *i*-th suitor.

- (p_s)_i describes i's preferences [k, l, ...] such that i's preference for 1st reviewer is k, 2nd reviewer is l, and so on. If k > l then suitor i prefers 1st over 2nd.
- (p_r)_j describes j's preferences [k, l, ...] such that j's preference for 1st suitor is k, 2nd suitor is l, and so on. If k > l then reviewer j prefers 1st over 2nd.

The algorithm described in the previous paragraph is formally presented in Algorithm 1. The protocol Π_{GS} in Algorithm 1 always produces a stable matching (i.e., computes \mathcal{F}_{GS} , the ideal functionality for Gale-Shapley stable matching).

Algorithm 1 Gale-Shapley Stable Matching $\Pi_{\rm GS}(S,R,p_s,p_r)$ 1: $S' \leftarrow S$ 2: while $\exists i \in S'$ do $i \leftarrow S'$ \triangleright Pick a suitor i from the set S'3: 4: $j \leftarrow (p_s)_i | (p_s)_{ij} \ge (p_s)_{ij'} \ \forall j' \in R$ such that suitor i has not already chosen the reviewer j, j'5: if (\cdot, j) then $M' \leftarrow M' \cup \{(i,j)\}$ 6: $S \leftarrow S' \backslash \{i\}$ 7: 8. else 9: $i' \leftarrow (\cdot, j)$ if $(p_r)_{ji} > (p_r)_{ji'} \triangleright$ i.e., reviewer j prefers suitor i10: to i' then $M' \gets M' \backslash \{(i',j)\}$ 11: $S' \leftarrow S' \cup \{i'\}$ 12: $M' \leftarrow M' \stackrel{\scriptstyle{\scriptstyle{\leftarrow}}}{\cup} \{(i,j)\}$ 13: $\overset{\frown}{S} \leftarrow S' \backslash \{i\}$ 14: 15: end if end if 16: 17: end while 18: return M

Theorem 3.1 (based on [22]). The stable matching protocol Π_{GS} always correctly computes and produces a correct stable matching.

Proof (Sketch). The standard Gale-Shapley stable matching protocol Π_{GS} described in Algorithm 1 is an iterative procedure (each iteration has to two stages: proposal and

matching) that correctly computes and produces a stable set of matching. The proof of Theorem 1 in [22] proves that the end of the iterative procedure (at most $n^2 - 2n + 2$ stages) will produce a correct stable set. To avoid redundancy we refer the reader to Section 3 (Theorem 1) in [22] for detailed proofs of Theorem 3.1.

3.2. Other Matching Problems

Matching problems are not restricted to the version just described. Indeed, depending on the objective function, matching problems can be used to interpret and solve a great variety of applications, especially in graph theory [19], [37], such as bipartite matching [38] optimizing for maximum cardinality or weight, as well as oblivious versions [9] of the same.

3.3. Cryptographic Building Blocks

3.3.1. Secret Sharing. In cryptography, secret sharing [45] refers to the process of splitting a secret among n parties such that each party does not learn anything about the whole secret from the share it holds. The secret can be reconstructed only if a certain minimum number of parties, greater than or equal to a threshold, t, combine their shares. The scheme is known as the (t, n) threshold scheme or t-out-n secret sharing (or Shamir secret sharing). In this work, we both Shamir secret sharing over arithmetic fields (integer length is set to the default 64).

3.3.2. Secure Multiparty Computation. Secure Multiparty Computation (MPC) [24], [53] in its most general form allows a set of parties to interact and compute a joint function of their private inputs while revealing nothing but the output. A MPC set-up assumes that there are n parties, $\{P_1, P_2, \ldots, P_n\}$, each with private input x_i , to jointly compute a function $(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_n)$, where each party P_i receives y_i as output and doesn't learn anything about inputs or outputs that is not logically implied by x_i and y_i . MPC can be achieved using different techniques such as garbled circuit with oblivious transfer, secret sharing, fully or partially homomorphic encryption, and functional encryption.

In this work, we use general model of MPC is the socalled generic Arithmetic Black Box \mathcal{F}_{ABB} , which is an ideal functionality that allows parties to input and output values to be secret-shared, and performs basic arithmetic operations on these secret values over a finite field \mathbb{F}_p . Concretely, the MPC protocol we use to implement \mathcal{F}_{ABB} is the SPDZ protocol. The main arithmetic operations in \mathcal{F}_{ABB} have roughly the following complexity when implemented in SPDZ.

We used two popular MPC frameworks SCALE-MAMBA [4] and MP-SPDZ [31] for benchmarking of our various oblivious stable matching constructions.

3.4. Technical Notation

3.4.1. Basic Notation. We denote |S| size of the suitors and reviewers. We use the notation $\langle x \rangle$ to denote a secret shared value. We used the notation defined in Table 1 throughout our paper.

Notation	Description		
$\langle s_p \rangle_i$	Suitor <i>i</i> 's preference For current		
	match.		
$\langle s_m \rangle_i$	Suitor <i>i</i> 's current match.		
$\langle r_p \rangle_i$	Reviewer <i>i</i> 's preference For current		
	match.		
$\langle r_m \rangle_i$	Reviewer <i>i</i> 's current match.		
$\langle u_s \rangle_i$	Matched status For suitor <i>i</i> (0 for		
	matched and 1 for unmatched).		
$\langle p_s \rangle_{ij}$	Suitor <i>i</i> 's preference For Reviewer		
	<i>j</i> .		
$\langle p_r \rangle_{ij}$	Reviewer <i>i</i> 's preference For Suitor		
	j.		
$\langle t^p \rangle, \langle t^m \rangle$	To store current reviewer prefer-		
	ence and current match.		
$\langle p_{us} \rangle_i$	To store preference For the <i>i</i> th re-		
	viewer.		

TABLE 1: We use this notation throughout all algorithms.

3.4.2. Conditional Operator. Given the intrinsic limitations on flow branching (e.g., control sentence if), the mechanism shown in Algorithm 2 allows us to simulate conditional assignments to variables, based on a secret shared binary value $\langle c \rangle$. Such approach has been previously used in other applications such as [3], [7], [13], [41].

Algorithm 2 Conditional Operator $\langle z \rangle \leftarrow_{\langle c \rangle} \langle x \rangle : \langle y \rangle$	
1: $\langle z \rangle \leftarrow \langle c \rangle . (\langle x \rangle - \langle y \rangle) + \langle y \rangle$ 2: return $\langle z \rangle$	

Input Data. We assume all input data to be elements of a finite field \mathbb{Z}_q bounded by some sufficiently large prime or RSA modulus q, such that no overflow occurs. This, let us treat all input values as plain integers, as long as their size is bounded by q such that $x \ll q$ for any input x.

Complexity. We measure complexity in communication rounds i.e., the number of operations that require information exchange in between the parties performing the computations.

3.5. Arithmetic Black Box \mathcal{F}_{ABB}

An Arithmetic Black Box (\mathcal{F}_{ABB}), is an abstraction technique used to simplify theoretical analysis in regards to security under composition (UC model [11]). First introduced by Damgåd and Nielsen [15], it encapsulates an ideal functionality and can be extended, as long as secure realizations are provided. We name ours \mathcal{F}_{ABB} , and use it to decouple the analysis of our constructions from the underlying MPC protocols.

Table 2, defines our arithmetic black box \mathcal{F}_{ABB} , and provides the round complexity and the corresponding UC secure realizations of any given functionality.

• **extract** ($\langle i \rangle, \langle V \rangle$). In Algorithm 5, we show how to secretly extract the $\langle i \rangle$ -th element in constant round, by shifting the elements of the shared vector $\langle V \rangle$ by a uniformly random secret mask $\langle r \rangle$, without leaking any information about the secret index $\langle i \rangle$.

3.6. Oblivious Random Access Memory

Oblivious RAM (ORAM) was first introduced by Goldreich and Ostrovsky [23], [25] in the context of protecting software from piracy, and efficient simulation of programs on oblivious RAMs which allow a client to conceal its access pattern to the remote storage by continuously shuffling and re-encrypting data as they are accessed. An adversary can observe the physical storage locations accessed, but the ORAM algorithm ensures that the adversary has negligible probability of learning anything about the true (logical) access pattern. Goldreich and Ostrovsky [25] showed that the lower bound for the cost of accessing a single entry in the storage is sub-linear with respect to the size of the storage. Linear ORAM is a naïve approach of ORAM which linearly searches the entire storage for each access such that the client can choose the desired entry using multiplexer. Gordon et al. [27] proposed to use ORAM mechanism inside twoparty secure computation (e.g., GC) in order to reduce the amortized cost of accessing a memory entry from linear to sub-linear. There are several improvements on the original idea of ORAM, including [40], [46], [48] which reduced the amortized per-access complexity to $O(\log^3 n)$.

An ORAM scheme uses an oblivious data structure in order to hide the access pattern, and it must implement two protocols: initialization protocol and access protocol. Initialization protocol is used to create and initialize the oblivious data structure from the given array of data. Access protocol is used to implement the actual access to the data structure. It translates the logical address that is created in the MPC protocol to the sequence of physical addresses. In RAM based secure computation, the memory accesses are handled by ORAM. Once the secret logical address is generated inside the MPC protocol, it is translated into multiple physical addresses by the access protocol (client side) that are revealed to both parties. Both parties then provide the requested memory entities back to the MPC protocol. At the end, the MPC protocol changes all the memory entities' data to hide which element was accessed and how it was changed and then it sends them to two parties to store them. There are different data structures for ORAM. Goldreich and Ostrovsky [23], [25] introduced two hierarchical layered structure ORAMs: Square-Root ORAM and Hierarchical ORAM. Shi et al. [46] initiated a tree-based ORAM scheme.

ORAM For MPC. Launchbury et al. [36] describe a protocol for obliviously searching an array of size N without revealing indices, with linear overhead (i.e., the cost of the procedure is O(N)). It involves expanding the secret-shared index i < N to a vector of size N that contains a one in the *i*-th position and zeroes elsewhere. The inner product of this index vector and the array of field elements gives the desired field element. The index vector can likewise be used to replace this field element with a new one while leaving the other field elements intact.

More relevant for us, Keller and Scholl [35] proposed a protocol with polylogarithmic overhead (i.e., the cost of the procedure is $O(poly(\log N))$), in the context of MPC, for oblivious implementations of an oblivious array using both the Binary Tree ORAM [46] and a variation of Path

Functionality	Description	Rounds	Protocol
$x \leftarrow \operatorname{open}(\langle x \rangle)$	Opening secret field elements	1	e.g. [8], [16], [47]
$\langle x \rangle \leftarrow x$	Strong public input in a secret field elements	1	e.g. [8], [16], [47]
$\langle z \rangle \leftarrow \langle x \rangle + \langle y \rangle$	Addition of secret field elements	0	e.g. [8], [16], [47]
$\langle z \rangle \leftarrow \langle x \rangle + y$	Addition of secret field element and public element	0	e.g. [8], [16], [47]
$\langle z \rangle \leftarrow \langle x \rangle \cdot \langle y \rangle$	Multiplication of secret field elements	1	e.g. [8], [16], [47]
$\langle z \rangle \leftarrow \langle x \rangle \cdot y$	Multiplication of secret field element and public element	0	e.g. [8], [16], [47]
$\langle b \rangle \leftarrow \langle x \rangle < \langle y \rangle$	Comparison of secret field elements	4 or $(1 + \log_2(\ell))$	e.g. [5], [12]
$\langle b \rangle \leftarrow \langle x \rangle < y$	Comparison of secret field element and public element	4 or $(1 + \log_2(\ell))$	e.g. [5], [12]
$\langle b \rangle \leftarrow \langle x \rangle \stackrel{?}{=} 0$	Equality test of secret field element and zero	4 or $(1 + \log_2(\ell))$	e.g. [5], [12]
$\langle b \rangle \leftarrow \langle x \rangle \stackrel{?}{=} \langle y \rangle$	Equality test between two secret field elements	4 or $(1 + \log_2(\ell))$	e.g. [5], [12]
$\langle b angle \leftarrow \langle x angle \stackrel{?}{=} y$	Equality test of a secret field element and a public element	4 or $(1 + \log_2(\ell))$	e.g. [5], [12]
$\langle x \rangle \xleftarrow{\$} random(size)$	Sample a some randomness in $[0, 2^{size})$	1	[12], [14]
$\langle V \rangle' \leftarrow \text{permute}(\langle V \rangle)$	Secretly permutes $\langle V \rangle$ s.t. elements are uniformly distributed in $ \langle V \rangle $	8 or $(1 + \log_2(l))$	[17], [33]
$\langle V \rangle_{\langle i \rangle} \leftarrow \operatorname{oram_array}(\langle i \rangle)$	Secretly extract the $\langle i \rangle$ -th element of a secret vector $\langle V \rangle$ using ORAM	$\log^3(V)$	[35]
$\langle x \rangle \leftarrow \texttt{oram_priority_queue}(\langle y \rangle)$	Secretly add and remove an element from a secret shared priority queue	$\log^5(V)$	[35]

TABLE 2: Secure Arithmetic operations provided by the Arithmetic Black Box \mathcal{F}_{ABB} .

ORAM [49] schemes, as a secret shared array that can be accessed using a secret index, without revealing this index. In other words, Keller and Scholl [35] improved the protocol in [36] by improving the position map of the elements using tree ORAM, i.e, storing several field elements in the same memory cell and packing several positions per field element. We refer the reader to Section 4 in [35] for the more detailed construction of the protocol.

ORAM based Stable Matching. Riazi et al. [42] construct a secure stable matching protocol using the combination of Yao's GC [53] and ORAM (Circuit ORAM [52] and Square-root ORAM [54]). Doerner et al. [18] study the development of strategies for RAM based secure computation by modification to the ORAM access protocol that enables efficient function application within an ORAM access. In this work, we mainly study and design of the primitives for Oblivous Stable Matching, in the context of MPC.

4. Oblivious Stable Matching

In this section, we describe a variant of **oblivious** Gale-Shapley [22] stable matching protocol, without revealing user private inputs, and briefly discuss correctness and complexity.

4.1. Threat Model

Semi-honest Participants: A *semi-honest* (also known as *honest-but-curious*) adversary follows the protocol specifications honestly, but may try to learn information about the private input data by inspecting the shared inputs sent by the participants in the process. Our approach ensures that the adversary (colluding with any subset of participants) can't learn any information about the private inputs of the honest participants when any subset of participants are semi-honest corruption. We note that these protocols achieve the same security level than the underlying protocol on any given setting. This includes security with abort, where parties would be susceptible to the same kind of reconstruction attacks already present in MPC protocols.

4.2. Näive Oblivious Stable Matching

We have adapted the standard Gale-Shapley [22] stable matching protocol Π_{GS} in Algorithm 1 so that its flow does

Functionality 1: Ideal Functionality for Näive and Optimized Oblivious Gale-Shapley Stable Matching $\mathcal{F}_{OblivGS}$

• **Parameters:** This functionality has all of the features of \mathcal{F}_{ABB} , and receives suitors' and reviewers' secret shared preference list $\langle p_s \rangle$ and $\langle p_r \rangle$ respectively.

• Stable Match: On receiving the aforementioned preference shares $(\langle p_s \rangle, \langle p_r \rangle))$ from all parties, it computes a stable matching $(\langle s_m \rangle, \langle r_m \rangle)$ which stores matched reviewer index and suitor index respectively.

not depend on the input data (obliviousness) as reflected in the oblivious stable matching protocol Π_{OblivGS} in Algorithm 3. Hence, our focus in protocol Π_{OblivGS} realizes the functionality $\mathcal{F}_{\text{OblivGS}}$ shown in Functionality 1.

Suitors propose to reviewers in a series of rounds. In each round, a suitor proposes to every reviewer. Since whether a suitor is unmatched or matched is oblivious and reviewers can't be accessed in the list of preference, every suitor HAS to propose to every reviewer. A proposal checks whether matching the current suitor and reviewer would result in a more stable match than their existing matches. In other words, we check whether the suitor and reviewer prefer each other over their existing matches. This check is also done in an oblivious manner so the result is stored in a shared value. Using conditional assignment operator (see Algorithm 2), the matches of the suitor and the reviewer are either updated or kept the same.

Algorithm 3 is almost identical to the original Gale-Shapley protocol Π_{GS} (see Algorithm 1). In the original protocol, each suitor (unless already matched) proposes to the unmatched reviewer whom it prefers the most in every iteration. In the oblivious version of the Gale-Shapley stable matching protocol $\Pi_{OblivGS}$, the suitor has to be checked against each reviewer for a new stable match. The last reviewer that the suitor's match is updated against becomes the final match of that suitor from that iteration of the algorithm. It can be proven (refer to Section 3 Theorem 1 in [22]; Theorem 3.1) that the final match formed will always be the match of highest preference that could be made for the suitor in that iteration. Thus, our proposed oblivious stable matching protocol Π_{OblivGS} in Algorithm 3 achieves the same output as a regular iteration of the Gale-Shapley protocol would have. After checking the suitor against every reviewer, we update the match of the last reviewer⁵ that was matched to the suitor and update the suitor that this reviewer was matched to, if it was matched to someone, as unmatched. The protocol keeps track (obliviously) of single suitors by means of vector $\langle u_s \rangle_i$ (the same that is used to identify non-matched suitors). The final two for loops obliviously update the suitors' and reviewers' current matches & preference for current matches.

The algorithm can be constructed by replacing all basic operations with their secure equivalents (provided by the underlying multiparty functionality); replacing conditional blocks by using oblivious assignments $\leftarrow_{\langle c \rangle}$; and limiting the number of iterations to the protocol iteration upper bound. We note that our time complexity is $\mathcal{O}(n^3)$ on the number of comparisons and multiplications as reflected in Algorithm 3.

The intuition of protocol Π_{OblivGS} in Algorithm 3 is as follows: at the core of the protocol there is a decisional procedure that allows us to retain or disregard a matched tuple. This is achieved, in part by the two inner loops, which identify a new preference pairing, each iteration, allowing us to contrast them with the current best matchings stored by the $\langle s_m \rangle$ and the $\langle r_m \rangle$ vectors. We do this at least as many times as elements on the suitors and reviewers. At the end of the last iteration, vectors $\langle s_m \rangle$ and $\langle r_m \rangle$ will contain the stable match.

4.2.1. Correctness. The protocol Π_{OblivGS} in Algorithm 3 is an oblivious adaptation of the Galey-Shapley protocol Π_{GS} . In their seminal result [2], [22], it is proven that at least one stable match is fixed per protocol iteration, giving an upper bound of at most the size of the matching vectors on the number of iterations. Using this principle, we can run the internal inner workings of the protocol, further adapted to work in an oblivious fashion guaranteeing correctness. As proved in Theorem 3.1 the standard stable matching protocol Π_{GS} always produces a correct stable set of matching, then our proposed oblivious stable matching protocol IIOblivGS also always produces a correct stable set of matching.

4.2.2. Complexity. Time complexity of $\Pi_{OblivGS}$ is $\mathcal{O}(n^3)$, on the number of secure comparisons and multiplications as reflected in Algorithm 3.

4.3. Optimized Oblivious Stable Matching

The proposed oblivious stable matching protocol Π_{OblivGS} in Algorithm 3 can be further optimized to achieve a reduction of the asymptotic complexity on the number of comparisons. Such changes are reflected on optimized oblivious stable matching protocol $\Pi_{OptOblivGS}$ in Algorithm 4⁶. Our focus in protocol $\Pi_{OptOblivGS}$ is to realize the functionality $\mathcal{F}_{OblivGS}$ shown in Functionality 1.

This optimized version of the protocol is an extension of our initial construction, but with a more sophisticated mechanism for stopping the protocol. Here, the protocol checks whether there exists a suitor with no match. In case Algorithm 3 Näive Oblivious Gale-Shapley Stable Matching $\Pi_{\text{OblivGS}}(\langle p_s \rangle, \langle p_r \rangle)$

- $\begin{array}{l} 1: \ \{(\langle s_p \rangle_i, \langle s_m \rangle_i), \cdots, (\langle s_p \rangle_{|S|}, \langle s_m \rangle_{|S|})\} \leftarrow \langle \bot \rangle \\ 2: \ \{(\langle r_p \rangle_i, \langle r_m \rangle_i), \cdots, (\langle r_p \rangle_{|S|}, \langle r_m \rangle_{|S|})\} \leftarrow \langle \bot \rangle \\ 3: \ \{\langle u_s \rangle_i, \cdots, \langle u_s \rangle_{|S|}\} \leftarrow \langle 1 \rangle \qquad \triangleright \text{ initially, all } \end{array}$ ▷ initially, all suitors are unmatched.
- 4: $\{\langle t^p \rangle, \langle t^m \rangle\} \leftarrow \langle \perp \rangle$
- 5: while i < |S| do
- for $j \in S$ do $u^s \leftarrow \langle 0 \rangle$ 6:
- 7:
- $\begin{array}{c} x^{r} \leftarrow \langle 0 \rangle \\ u^{m} \leftarrow \langle u \end{array}$ 8: 9

9:
$$u \leftarrow \langle u_s \rangle_j$$

10: for $k \in R$ do

$$\begin{array}{c} | 11: \\ \langle u^m \rangle \end{array} \stackrel{?}{\leftarrow} (\langle s_p \rangle_j \stackrel{?}{<} \langle p_s \rangle_{jk}) \times (\langle r_p \rangle_k \stackrel{?}{<} \langle p_r \rangle_{kj}) \times \\ \end{array}$$

 \triangleright match status of suitor *j*.

12: ▷ current match of reviewer k.

13: \triangleright update suitor *j*'s current match.

 $\langle s_p \rangle_j \leftarrow_{\langle c \rangle} \langle p_s \rangle_{jk} : \langle s_p \rangle_j$ \triangleright update suitor *j*'s 14: preference.

15: $\langle u_s \rangle_j \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle u_s \rangle_j$ \triangleright update suitor j's match status

16:
$$\langle r^r \rangle \leftarrow_{\langle c \rangle} k : \langle r^r \rangle \qquad \triangleright$$
 store current reviewer.
17: $\langle t^m \rangle \leftarrow_{\langle c \rangle} k : \langle t^m \rangle \qquad \triangleright$ reviewer's new match

17:
$$\langle t^m \rangle \leftarrow_{\langle c \rangle} j : \langle t^m \rangle \mapsto$$
 reviewer's new match

18: $\langle t^{\nu} \rangle \leftarrow_{\langle c \rangle} \langle p_r \rangle_{kj} : \langle t^p \rangle$ ▷ reviewer's new preference.

```
19:
           end for
```

for $z \in |S|$ do 20:

21:
$$\langle c \rangle \leftarrow z \stackrel{?}{=} \langle u^s \rangle \Rightarrow$$
 previously matched suitor

 $\langle u_s \rangle_z \leftarrow_{\langle c \rangle} \langle 1 \rangle : \langle u_s \rangle_z$ ▷ update matched to 22: unmatched.

 $\langle s_m \rangle_z \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle s_m \rangle_z \triangleright$ update current match. $\langle s_p \rangle_z \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle s_p \rangle_z \qquad \triangleright$ update current 23: 24: preference.

25: end for

- for $z \in |R|$ do 26:
- $\langle c \rangle \leftarrow z \stackrel{?}{=} \langle r^r \rangle \triangleright$ currently matched reviewer. $\langle r_m \rangle_z \leftarrow_{\langle c \rangle} \langle t^m \rangle : \langle r_m \rangle_z \triangleright$ update current 27: 28: match.

 $\langle r_p \rangle_z \leftarrow_{\langle c \rangle} \langle t^p \rangle : \langle r_p \rangle_z$ 29: ▷ update current preference.

30: end for

31: end for

i + +32:

33: end while

34: return $\langle s_m \rangle, \langle r_m \rangle$

no more single suitors are to be found the protocol has to be stopped, hence the break. This means that parties learn the result of $\langle m \rangle == 0$, to determine whether or not the protocol should be stopped. Again, in each round, a suitor proposes to every other reviewer. The protocol follows this by performing the same comparison introduced by our initial approach from Algorithm 3.

The advantage is that, although the worst case scenario complexity matches our previous algorithm (in the number of multiplications), the actual number of iterations depends on the inputs. The evident drawback is that some leakage is produced, in this case the number of iterations that were needed to obtain the result of the computation.

4.3.1. Correctness. The protocol $\Pi_{OptOblivGS}$ in Algorithm 4 is the optimized version of our proposed oblivious stable matching protocol Π_{OblivGS} in Algorithm 3, which is an oblivious adaptation of the standard Galey-Shapley pro-

^{5.} Removing the dependence on the last match would allow us to run all the internal loops in parallel, considerably speeding up execution time. We leave this for future work.

^{6.} The differences between Algorithm 3 and Algorithm 4 are only in lines 6-14 of Algorithm 4.

Algorithm 4 Optimized Oblivious Gale-Shapley Stable Matching $\Pi_{\text{OptOblivGS}}(\langle p_s \rangle, \langle p_r \rangle)$ $\begin{array}{l} 1: \ \{(\langle s_p \rangle_i, \langle s_m \rangle_i), \cdots, (\langle s_p \rangle_{|S|}, \langle s_m \rangle_{|S|})\} \leftarrow \langle \bot \rangle \\ 2: \ \{(\langle r_p \rangle_i, \langle r_m \rangle_i), \cdots, (\langle r_p \rangle_{|S|}, \langle r_m \rangle_{|S|})\} \leftarrow \langle \bot \rangle \\ 3: \ \{\langle u_s \rangle_i, \cdots, \langle u_s \rangle_{|S|}\} \leftarrow \langle 1 \rangle \\ 4: \ \{\langle t^p \rangle, \langle t^m \rangle\} \leftarrow \langle \bot \rangle \end{array}$ 5: while true do $\langle v \rangle \leftarrow \langle 0 \rangle$ 6: $\langle m \rangle \leftarrow \langle 0 \rangle$ 7: 8: for $j \in S$ do ▷ check if there exists an unmatched suitor. 9: $\langle v \rangle \leftarrow ((\langle 1 \rangle - \langle v \rangle) \cdot \langle u_s \rangle_j)$ $\langle m \rangle \leftarrow_{\langle v \rangle} j : \langle m \rangle$ 10: end for 11: if $\operatorname{open}(\langle m \rangle \stackrel{?}{=} 0)$ then \triangleright if no unmatched suitor exists 12: then break. 13: break 14: end if for $j \in S$ do 15: 16: $u^s \leftarrow \langle 0 \rangle$ $r^r \leftarrow \langle 0 \rangle$ 17: $u^m \leftarrow \langle u_s \rangle_j$ 18: 19: for $k \in R$ do $\langle c \rangle \leftarrow (\langle s_p \rangle_j \stackrel{?}{<} \langle p_s \rangle_{jk}) \times (\langle r_p \rangle_k \stackrel{?}{<} \langle p_r \rangle_{kj}) \times$ 20: $\langle u^m \rangle$ $\langle u^m \rangle \leftarrow_{\langle c \rangle} \langle r_m \rangle_k : \langle u^m \rangle$ 21: $\begin{array}{l} \langle a_{m} \rangle \langle \langle c_{c} \rangle \langle m/k \rangle \langle a_{m} \rangle_{j} \\ \langle s_{m} \rangle_{j} \leftarrow_{\langle c \rangle} k : \langle s_{m} \rangle_{j} \\ \langle s_{p} \rangle_{j} \leftarrow_{\langle c \rangle} \langle p_{s} \rangle_{jk} : \langle s_{p} \rangle_{j} \\ \langle u_{s} \rangle_{j} \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle u_{s} \rangle_{j} \\ \langle r^{r} \rangle \leftarrow_{\langle c \rangle} k : \langle r^{r} \rangle \\ \langle t^{m} \rangle \leftarrow_{\langle c \rangle} j : \langle t^{m} \rangle \\ \langle t^{p} \rangle \leftarrow_{\langle c \rangle} \langle m \rangle \rangle \dots \langle t^{p} \rangle$ 22: 23: 24: 25. 26: $\langle t^p \rangle \leftarrow_{\langle c \rangle} \langle p_r \rangle_{kj} : \langle t^p \rangle$ 27: 28: end for for $z \in |S|$ do 29: $\langle c \rangle \leftarrow z \stackrel{?}{=} \langle u^s \rangle$ 30: 31: $\langle u_s \rangle_z \leftarrow_{\langle c \rangle} \langle 1 \rangle : \langle u_s \rangle_z$ $\langle s_m \rangle_z \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle s_m \rangle_z$ 32: 33: $\langle s_p \rangle_z \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle s_p \rangle_z$ end for 34: for $z \in |R|$ do 35: $\langle c \rangle \leftarrow z \stackrel{?}{=} \langle r^r \rangle$ 36: $\langle \vec{r}_m \rangle_z \leftarrow_{\langle c \rangle} \langle \vec{t}^m \rangle : \langle r_m \rangle_z$ 37: $\langle r_p \rangle_z \leftarrow_{\langle c \rangle} \langle t^p \rangle : \langle r_p \rangle_z$ 38: 39: end for end for 40: 41: end while 42: return $\langle s_m \rangle, \langle r_m \rangle$

tocol Π_{GS} . We discussed in §4.2.1 the correctness of the protocol $\Pi_{OblivGS}$ and, using the same principle, we argue that the optimized version of oblivious stable matching protocol $\Pi_{OptOblivGS}$ also produces correct output.

4.3.2. Complexity. Time complexity of our optimized oblivious stable matching protocol $\Pi_{\text{OptOblivGS}}$ is $\mathcal{O}(n^3)$.

5. Oblivious Stable Matching based on Random Access Memory

Let us consider the oblivious nature of the protocols in Algorithm 3 and Algorithm 4. To achieve oblivious vector access on a secret index, we are required to explore all elements contained in our matrices and vectors. In this section, we explore an alternative way, where we make use of techniques to operate over secret shared memory. In other words, private array access at a private index location.

To achieve this, we propose two alternative paths. First, the use of ORAM (in $\S5.1$) and second, a *novel* constant round lookup table-based method for random memory access (in $\S5.2$); these are both realizations of the extraction functionality described in 2. We then describe a *novel* oblivious Gale-Shapley stable matching algorithm based on the random memory access model (in $\S5.3$), that can be implemented using any of these realizations.

The variants of extract functionality defined in Table 3 can be trivially derived from the extract functionality in the First row from the same table, which is our focus henceforward.

Functionality 2: Ideal Functionality for private array access $\mathcal{F}_{extract}$

• **Parameters:** Receives a secret index positions $\langle i \rangle$ and $\langle j \rangle$, and secret shared vector $\langle V \rangle$ and $\langle M \rangle$.

• Memory Access: On receiving $(\langle i \rangle, \langle j \rangle)$ from all parties, where either $(\langle V \rangle, \langle M \rangle)$ are stored in memory, retrieves either $\langle V \rangle_{\langle i \rangle}$ or $\langle M \rangle_{\langle i \rangle, \langle j \rangle}$ according to Table 3.

5.1. ORAM Based Random Access

As shown in our Related Work, the more typical approach in the literature is to instantiate our Ideal Functionality is via ORAM. Keller and Scholl [35] for instance, provided several UC secure data structures based on ORAM for MPC. The extract functionality, introduced in the first row in Table 3 can be trivially implemented using their Oblivious Array (oram_array) construction. This comes at the additional cost⁷ of some logarithmic overhead on the vector size.

The extraction of a secret element from a vector $\langle V \rangle$ is trivially obtained via invoking:

 $\langle V \rangle_{\langle i \rangle} = \texttt{extract}(\langle i \rangle, \langle V \rangle) = \texttt{oram_array}(\langle i \rangle, \langle V \rangle)$.

Whereas, the extraction of an element from a secret shared matrix $\langle M \rangle$ using the same ORAM construction can be then defined as follows:

```
\begin{array}{l} \langle M \rangle_{\langle i \rangle \langle j \rangle} = \texttt{extract}(\langle i \rangle, \langle j \rangle, \langle M \rangle) = \\ \texttt{oram\_array}(\langle i \rangle \times |M_0| + \langle j \rangle, \langle M \rangle). \end{array}
```

 $|M_0|$ is the size of the first row of the matrix M.

Finally, note that as Keller and Scholl pointed out, there are scenarios on which their protocols underperform accessing every element to extract an specific index.

5.2. Lookup Table Random Access

Given the loss of performance in the above, and inspired by [17], [33], we propose a constant round construction, based on *lookup tables* in Algorithm 5. Our formulation is black box: its main advantage is that it

^{7.} On the number of communication rounds.

Functionality	returns The:
$\langle V \rangle_{\langle i \rangle} \leftarrow \text{extract}(\langle i \rangle, \langle V \rangle)$	$\langle i \rangle$ -th element of a secret vector $\langle V \rangle$
$\langle M \rangle_{\langle i \rangle \langle j \rangle} \leftarrow \text{extract}(\langle i \rangle, \langle j \rangle, \langle M \rangle)$	$\langle i \rangle$ -th row and $\langle j \rangle$ -th column element of a secret matrix $\langle M \rangle$
$\langle M \rangle_{i\langle j \rangle} \leftarrow \text{extract}(i, \langle j \rangle, \langle M \rangle)$	<i>i</i> -th row and $\langle j \rangle$ -th column element of a secret matrix $\langle M \rangle$
$\langle M \rangle_{\langle i \rangle j} \leftarrow \texttt{extract}(\langle i \rangle, j, \langle M \rangle)$	$\langle i \rangle$ -th row and j-th column element of a secret matrix $\langle M \rangle$
$\langle M \rangle_{\langle i \rangle} \leftarrow \text{extract}(\text{row}, \langle i \rangle, \langle M \rangle)$	$\langle i \rangle$ -th row of a secret matrix $\langle M \rangle$
$\langle M \rangle_{\langle j \rangle} \leftarrow \texttt{extract}(\texttt{column}, \langle j \rangle, \langle M \rangle)$	$\langle j \rangle$ -th column of a secret matrix $\langle M \rangle$

TABLE 3: The operations provided by $\mathcal{F}_{extract}$.

does not require function dependant pre-processing – nor it is tied to a specific party setup or configuration.

We describe the extract functionality, to secretly extract the $\langle i \rangle$ -th element, by shifting the elements of the shared vector $\langle V \rangle$ by a uniformly random secret mask $\langle r \rangle$, without leaking any information about the secret index $\langle i \rangle$. From this point on in this section, we assume w.l.g. the size of the vector *shareV* is a power of 2. A brief sketch of the Algorithm 5 is as follows:

First, we sample some random secret $\langle r \rangle$ and use it to mask $\langle i \rangle$, such that: $\langle i \rangle + \langle r \rangle \pmod{|V|}$. We note the result is uniform in |V|, hence it can be securely disclosed (*on lines 1. and 2.*). To be able to use our masked index, we require to shift the entries of V by r. We achieve this by constructing a 1 hot vector of size |v| for $\langle r \rangle$ and multiplying it by publicly available shift permutation matrices $\{I^0, \dots, I^{|V|}\}$. They are designed to shift the elements by any number between 0 and (|V| - 1). Given that the aim of this work is practicality and implementation, we propose to achieve this by means of an equality test (which is widely available on commonly used MPCframeworks). However, this 1 hot vector could be part of a pre-computed tuple $(\langle r \rangle, \{b_0^r, \dots, b_{|V|}^r)$ or created via some DPF. Given the *non-input data* related nature of this process, the selection the rotating permutation could be also pre-computed.

The shifting of the shared vector $\langle V \rangle$ is achieved by multiplying it with the rotation/permutation matrix selected via the steps described above – these matrices are public and are stored in a lookup table. The lookup table is accessed via the secret value $\langle r \rangle$ and the matrix I' is obtained. Finally, we obtain the shifted shared vector $\langle V' \rangle$ by multiplying $\langle V \rangle$ with I', we then can simply return the secret shared element at index i'.

Algorithm 5 extract $(\langle i \rangle, \langle V \rangle)$					
1: Sample $\langle r \rangle \stackrel{R}{\leftarrow} \mathbb{Z}_{ V }$ 2: $i' \leftarrow \text{open}(\langle i \rangle + \langle r \rangle$ 3: $I \leftarrow \{I^0, \cdots, I^{ V }\}$ 4: for $i = 0$ to $ V $ do	$ \begin{tabular}{lllllllllllllllllllllllllllllllllll$				
4. If $J = 0$ is $ V $ is 5. $C_j \leftarrow \langle (\langle r \rangle \stackrel{?}{=} j) \rangle$ 6. end for 7. $I' \leftarrow \sum_{j=0}^{ V } C_j \times I^j$ 8. $\langle V' \rangle \leftarrow \langle V \rangle \times I'$ 9. return $\langle V' \rangle_{i'}$					

5.2.1. Complexity. Now, we describe the round complexity of our proposed extract protocol (step-wise) as follows:

The operation $\operatorname{open}(\langle i \rangle + \langle r \rangle \mod |V|)$ (Step 2) is required 1 round of complexity. Then, all the comparison inequalities $\langle (\langle r \rangle \stackrel{?}{=} j) \rangle$ (Step 5) can be performed in parallel, and at the same time than the modulo operation.

Furthermore, obtaining the module requires 4 rounds of complexity using the same approach as in [5]. If we were to use Catrina and De Hoogh's modulo operation from [12], complexity would be sublinear on the input size, but constant on the vector size. Note that we assume the complexity of the modulo function dominates over the equality test. Finally, the protocol requires 1 round complexity for point-to-point multiplications $C_j \times I^j$ (Step 7) and one secret multiplication $\langle V \rangle \times I'$ (Step 8). So, in total is 1 round for opening + 4 rounds for the equality tests + 1 round for the matrix multiplication.

• Round 1:
$$i' \leftarrow \operatorname{open}(\langle i \rangle + \langle r \rangle \mod |V|)$$
.
• Round 2 - 5: $C_0 \leftarrow \langle (\langle r \rangle \stackrel{?}{=} 0) \rangle, \dots, C_{|V|} \leftarrow \langle (\langle r \rangle \stackrel{?}{=} |V|) \rangle$.
• Round 6: $\langle V' \rangle \leftarrow \langle V \rangle \times I'$.

5.3. Stable Matching on the RAM Model

The proposed optimized oblivious stable matching protocol $\Pi_{OptOblivGS}$ in Algorithm 4 can be further optimized using the constructions above. Indeed, we can achieve a reduction of the asymptotic round complexity and on the number of comparisons. In Algorithm 6, we present an improved protocol for Gale-Shapely Stable Matching on the RAM Model.

This is achieved by employing any of the previous realization of the $\mathcal{F}_{\text{extract}}$, for the selection of suitors. This allows us to avoid iterating over each suitor in every round instead obliviously selecting a single unmatched suitor in each round. We do this after the initial check for unmatched suitors itself. More precisely, the suitor's currently matched preference, its preference list and reviewers preferences for suitor are all obliviously accessed. Thereafter, the for loops are a direct extension of the optimized oblivious stable matching protocol Algorithm 4. The time complexity of Algorithm 6 is $O(n^2)$.

Functionality 3: Ideal Functionality for Gale-Shapley Stable Matching on the RAM Model $\mathcal{F}_{LutOblivGS}$

• **Parameters:** This functionality has all of the features of \mathcal{F}_{ABB} , all the features of the functionality $\mathcal{F}_{\text{extract}}$, and receives suitors' and reviewers' secret shared preference list $\langle p_s \rangle$ and $\langle p_r \rangle$ respectively.

• Stable Match: On receiving the aforementioned preference shares $(\langle p_s \rangle, \langle p_r \rangle))$ from all parties, it computes a stable matching $(\langle s_m \rangle, \langle r_m \rangle)$ which stores matched reviewer index and suitor index respectively.

Algorithm 6 (Non-Amortized) Gale-Shapely Stable Matching on the RAM Model $\Pi_{\text{LutOblivGS}}(\langle p_s \rangle, \langle p_r \rangle)$

 $1: \{ (\langle s_p \rangle_i, \langle s_m \rangle_i), \cdots, (\langle s_p \rangle_{|S|}, \langle s_m \rangle_{|S|}) \} \leftarrow \langle \bot \rangle \\ 2: \{ (\langle r_p \rangle_i, \langle r_m \rangle_i), \cdots, (\langle r_p \rangle_{|S|}, \langle r_m \rangle_{|S|}) \} \leftarrow \langle \bot \rangle \\ 3: \{ \langle u_s \rangle_i, \cdots, \langle u_s \rangle_{|S|} \} \leftarrow \langle 1 \rangle \\ 4: \{ \langle t^p \rangle, \langle t^m \rangle \} \leftarrow \langle \bot \rangle \end{cases}$ while true do 5: $\langle v \rangle \leftarrow \langle 0 \rangle$ 6: $\langle m \rangle \leftarrow \langle 0 \rangle$ 7: 8: for $j \in S$ do $\langle v \rangle \leftarrow ((\langle 1 \rangle - \langle v \rangle) \cdot \langle u_s \rangle_j)$ 9: $\langle m \rangle \leftarrow_{\langle v \rangle} j : \langle m \rangle$ 10: end for 11: if $\operatorname{open}(\langle m \rangle \stackrel{?}{=} 0)$ then 12: 13: break end if 14: 15: $\langle u^s \rangle \leftarrow \langle 0 \rangle$ $\langle r^r \rangle \leftarrow \langle 0 \rangle$ 16: $\langle s^p \rangle \leftarrow \langle 0 \rangle$ 17: 18: $\langle s_p \rangle_{\langle m \rangle} = \text{extract}(\langle m \rangle, \langle s_p \rangle)$ $\langle p_s \rangle_{\langle m \rangle} = \text{extract}(\text{row}, \langle m \rangle, \langle p_s \rangle)$ 19: $\langle p_r \rangle_{\langle m \rangle} = \text{extract}(\text{column}, \langle m \rangle, \langle p_r \rangle)$ 20: 21: for $k \in R$ do $\begin{array}{l} \langle c \rangle \leftarrow (\langle s_p \rangle_{\langle m \rangle} \stackrel{?}{<} \langle p_s \rangle_{\langle m \rangle k}) \times (\langle r_p \rangle_k \stackrel{?}{<} \langle p_r \rangle_{\langle m \rangle k}) \\ \langle u^s \rangle \leftarrow_{\langle c \rangle} \langle r_m \rangle_k : \langle u^s \rangle \\ \langle r^r \rangle \leftarrow_{\langle c \rangle} k : \langle r^r \rangle \\ \langle s^p \rangle \leftarrow_{\langle c \rangle} \langle p_s \rangle_{\langle m \rangle k} : \langle s^p \rangle \\ \langle t^m \rangle \leftarrow_{\langle c \rangle} \langle m \rangle : \langle t^m \rangle \\ \langle t^p \rangle \leftarrow_{\langle c \rangle} \langle m \rangle : \langle t^p \rangle \\ \end{array}$ 22: 23: 24: 25: 26: $\langle t^p \rangle \leftarrow \langle c \rangle \langle p_r \rangle \langle m \rangle_k : \langle t^p \rangle$ 27: end for 28: for $z \in S$ do 29: $\langle c \rangle \leftarrow z \stackrel{?}{=} \langle u^s \rangle$ 30: 31: $\langle u_s \rangle_z \leftarrow_{\langle c \rangle} \langle 1 \rangle : \langle u_s \rangle_z$ $\begin{array}{c} \langle s_m \rangle_z \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle s_m \rangle_z \\ \langle s_p \rangle_z \leftarrow_{\langle c \rangle} \langle 0 \rangle : \langle s_p \rangle_z \end{array}$ 32: 33: $\langle c' \rangle \leftarrow z \stackrel{\scriptscriptstyle !}{=} \langle m \rangle$ 34: $\begin{array}{l} \langle u_s \rangle_z \leftarrow_{\langle c' \rangle} \langle 0 \rangle : \langle u_s \rangle_z \\ \langle s_m \rangle_z \leftarrow_{\langle c' \rangle} \langle r^r \rangle : \langle s_m \rangle_z \\ \langle s_p \rangle_z \leftarrow_{\langle c' \rangle} \langle s^p \rangle : \langle s_p \rangle_z \end{array}$ 35: 36: 37: 38: end for 39: for $z \in R$ do $\begin{array}{l} \langle c \rangle \leftarrow z \stackrel{?}{=} \langle r^r \rangle \\ \langle r_m \rangle_z \leftarrow_{\langle c \rangle} \langle t^m \rangle : \langle r_m \rangle_z \\ \langle r_p \rangle_z \leftarrow_{\langle c \rangle} \langle t^p \rangle : \langle r_p \rangle_z \end{array}$ 40: 41: 42: end for 43. 44: end while 45: return $\langle s_m \rangle, \langle r_m \rangle$

6. Security Proof Sketch

In this section, we model and prove the security (we provide only a proof sketch for brevity) of our constructions using the real/ideal paradigm. For our proofs, security is modelled by defining two interactions: a *real* interaction where the parties execute a protocol in the presence of an adversary \mathcal{A} and the environment \mathcal{E} , and an *ideal* interaction where parties send their inputs to a trusted functionality \mathcal{F} that conducts the desired computation truthfully. Security requires that for every adversary \mathcal{A} in the real interaction, there is a simulator Sim in the ideal interaction, such that no environment \mathcal{E} can distinguish between real and ideal interactions. A protocol II is said to *securely realize* a functionality \mathcal{F} if for every adversary \mathcal{A} in the real interaction, such that no environment \mathcal{E} is a simulator Sim in the ideal interaction, such that no environment \mathcal{F} if for every adversary \mathcal{A} in the real interaction, there is a simulator Sim in the ideal interaction, such that no environment \mathcal{E} is a simulator Sim in the ideal interaction, such that no environment \mathcal{E} is a simulator Sim in the ideal interaction, such that no environment \mathcal{E} .

on any input, can tell apart the real interaction from the ideal interaction, except with negligible probability (in the security parameter λ).

We show that our protocols are secure under a malicious adversary. If a protocol invokes another sub-protocol for a functionality \mathcal{F} , we prove the security by replacing the sub-protocol invocation with the corresponding functionality call. This refers to \mathcal{F} -hybrid model.

Theorem 6.1. The protocol $\Pi_{\text{OblivGS}}(\langle p_s \rangle, \langle p_r \rangle)$ in Algorithm 3, securely realizes the functionality $\mathcal{F}_{\text{OblivGS}}$ (in Functionality 1) in the \mathcal{F}_{ABB} -hybrid model.

Proof Sketch of Theorem 6.1. The computing party of the Algorithm 3 learn the following information:

- For each suitor $j \in S$ secretly choose a perfect match reviewer $k \in R$ by checking the condition $\langle c \rangle$, and then update the current match, current preference and match status for the suitor j by checking the same condition $\langle c \rangle$. A fresh share of the condition $\langle c \rangle$ is computed as $\langle c \rangle \leftarrow (\langle s_p \rangle_j \stackrel{?}{<} \langle n \rangle_{Y_j}) \times (\langle r_j \rangle_Y) \stackrel{?}{<} \langle n \rangle_{Y_j} \rangle \times \langle u^m \rangle \times \langle u^m \rangle$ (Step 11)
- ⟨p_s⟩_{jk}) × (⟨r_p⟩_k [?]< ⟨p_r⟩_{kj}) × ⟨u^m⟩ (Step 11).
 Update the current match, current preference and match status for previously matched suitor of reviewer k by computing and checking a fresh share of the condition ⟨c⟩ ← z [?] = ⟨u^s⟩ (Step 21). Then, update the current match and current preference for the reviewer k by computing and checking a fresh share of the condition ⟨c⟩ ← z [?] = ⟨r^r⟩ (Step 27).
- Finally, once all the suitor finds the perfect match, it outputs a fresh share of stable matching (⟨s_m⟩, ⟨r_m⟩) which stores matched reviewer index and suitor index respectively.

However, these are all fresh shares of these values and hence can be perfectly simulated by sending random fresh shares of 0.

Theorem 6.2. The protocol $\Pi_{\text{OptOblivGS}}(\langle p_s \rangle, \langle p_r \rangle)$ in Algorithm 4, securely realizes the functionality $\mathcal{F}_{\text{OptOblivGS}}$ (see section 4.3) in the \mathcal{F}_{ABB} -hybrid model.

Proof Sketch of Theorem 6.2. The computing party of the Algorithm 4 learn the following information:

 For each suitor j ∈ S choose a perfect match reviewer k ∈ R by checking the condition ⟨c⟩, and then update the current match, current preference and match status for the suitor j by checking the same condition ⟨c⟩. A fresh share of the condition

$$\langle c \rangle$$
 is computed as $\langle c \rangle \leftarrow (\langle s_p \rangle_j < \langle p_s \rangle_{jk}) \times (\langle r_p \rangle_k < \langle p_r \rangle_{kj}) \times \langle u^m \rangle$ (Step 22).

Then, update the current match, current preference and match status for the previously matched suitor of reviewer k by computing and checking a fresh share of the condition ⟨c⟩ ← z = ⟨u^s⟩ (Step 30). And then, update the current match and current preference for the reviewer k by computing and checking a fresh share of the condition ⟨c⟩ ← z = ⟨r^r⟩ (Step 40).

 Finally, once all the suitor finds the perfect match, it outputs a fresh share of stable matching ((s_m), (r_m)) which stores matched reviewer index and suitor index respectively.

However, these are all fresh shares of these values and hence can be perfectly simulated by sending random fresh shares of 0. The only information that the computing party learns is from <code>open($\langle m \rangle \stackrel{?}{=} 0$) (Step 12); this reveals only whether or not there are any unmatched suitors. It does not reveal any information about said unmatched (or matched) suitor to the adversary. Hence, the revealed information by $open(\langle m \rangle \stackrel{?}{=} 0)$ is uniformly random from the computing party's view, so the information learned by the computing party can be perfectly simulated.</code>

Theorem 6.3. The extract protocol in Algorithm 5, securely computes the functionality $\mathcal{F}_{extract}$ (in Functionality 2) in the \mathcal{F}_{ABB} -hybrid model.

Proof Sketch of Theorem 6.3. Now, we prove the security of our extract protocol. The computing party of Algorithm 5 learn a fresh share of following information: $C_j \leftarrow \langle (\langle r \rangle \stackrel{?}{=} j) \rangle \quad \forall j = 0 \text{ to } |V| \text{ (Step 4 - 6)}, I' \leftarrow \sum_{j=0}^{|V|} C_j \times I^j \text{ (Step 7)}, \langle V' \rangle \leftarrow \langle V \rangle \times I' \text{ (Step 8)}$ and $\langle V' \rangle_{i'}$ (Step 9). However, these are all fresh shares of these values and hence can be perfectly simulated by sending random fresh share of 0.

The only information that the computing party learns is $i' \leftarrow \text{open}(\langle i \rangle + \langle r \rangle \pmod{|V|})$ (Step 2), where $\langle i \rangle$ is a secret index and $\langle r \rangle$ is secret random value unknown to the computing party. It does not reveal any information to the adversary, beyond indexes of a permuted vector (or matrix). Hence, the distribution of i' is uniformly random from the computing party's point of view, so the information learned by the computing party can be perfectly simulated. \Box

Theorem 6.4. The protocol $\Pi_{\text{LutOblivGS}}(\langle p_s \rangle, \langle p_r \rangle)$ in Algorithm 6, securely realizes the functionality $\mathcal{F}_{\text{LutOblivGS}}$ (in Functionality 3) in the ($\mathcal{F}_{\text{extract}}, \mathcal{F}_{ABB}$)-hybrid model.

Proof Sketch of Theorem 6.4. The computing party of Algorithm 6 learns the following information:

- Throughout all suitors j ∈ S, first it calculates a fresh share of condition ⟨v⟩ ← ((⟨1⟩ ⟨v⟩) ⋅ ⟨u_s⟩_j) (Step 9), and find an unmatched suitor ⟨m⟩ ←_{⟨v⟩} j : ⟨m⟩ (Step 10) by checking ⟨v⟩.
- Then, extracting a fresh share of suitor $\langle m \rangle$'s preference for the current match (Step 18), suitor $\langle m \rangle$'s preference for each reviewer (Step 19), and each reviewer's preference for suitor $\langle m \rangle$ (Step 20) using extract (as $\mathcal{F}_{\text{extract}}$ functionality provides the perfect security; see Theorem 6.3).
- For the suitor $\langle m \rangle$ choose a perfect match reviewer $k \in R$ by checking the condition $\langle c \rangle$, and a fresh share of the condition is computed as $\langle c \rangle \leftarrow (\langle s_p \rangle_{\langle m \rangle} \stackrel{?}{<} \langle p_s \rangle_{\langle m \rangle k}) \times (\langle r_p \rangle_k \stackrel{?}{<} \langle p_r \rangle_{\langle m \rangle k})$ (Step 22).
- Then update the current match, current preference and match status for previously matched suitor of reviewer k by computing and checking a fresh

share of the condition $\langle c \rangle \leftarrow z \stackrel{?}{=} \langle u^s \rangle$ (Step 30), update the current match, current preference and match status for the suitor j by computing and checking a fresh share of the same condition $\langle c \rangle' \leftarrow z \stackrel{?}{=} \langle m \rangle$ (Step 34), and update the current match and current preference for the reviewer k by computing and checking a fresh share of the condition $\langle c \rangle \leftarrow z \stackrel{?}{=} \langle r^r \rangle$ (Step 40).

• Finally, once all the suitor finds the perfect match by checking $\operatorname{open}(\langle m \rangle \stackrel{?}{=} 0)$ (Step 12). This reveals only whether or not there are any unmatched suitors. It does not reveal any information about the unmatched (or matched) suitor to the adversary. The revealed information from $\operatorname{open}(\langle m \rangle \stackrel{?}{=} 0)$ is uniformly random from the computing party's point of view and can be perfectly simulated. In the end, it outputs a fresh share of stable matching ($\langle s_m \rangle, \langle r_m \rangle$) which stores matched reviewer index and suitor index respectively.

However, these are all fresh shares of these values and hence can be perfectly simulated.

7. Empirical Evaluation

In this section, we experimentally evaluate and compare our proposed oblivious variants and random memory access based Gale-Shapley [22] stable matching.

7.1. Asymptotic Complexity Comparison

In, Table 4 we show an asymptotic complexity comparison between our proposed protocols and the existing state-of-the-art secure stable matching approaches. The main drawback of existing work is the high computation and round complexity, which limits scalability.

TABLE 4: Round complexity for our proposed protocols vs existing work, where n is the set size.

Paper	Round Complexity
Golle [26]	$\mathcal{O}(n^3 poly(\log n)))$
Franklin et al. [20]	$\mathcal{O}(n^2 poly(\log n)))$
Franklin et al. [21]	$\mathcal{O}(n^2 poly(\log n))$
Zahur et al. [54]	$\mathcal{O}(n^2 poly(\log n))$
Keller et al. [35]	$\mathcal{O}(n^2 poly(\log n))$
Riazi et al. [42]	$\mathcal{O}(n^2 poly(\log n))$
Doerner et al. [18]	$\mathcal{O}(n^2 poly(\log n))$
Algorithm 3	$\mathcal{O}(n^3)$
Algorithm 4	$\Omega(n^2)$
Algorithm 6	$O(n^2)$

7.2. Setup and Implementation Details

We implemented and conducted benchmarking of our secure stable matching algorithms on top of the two most commonly used MPC frameworks SCALE-MAMBA [4] version 1.14^8 and MP-SPDZ [31] version $0.3.2^9$. We

^{8.} https://github.com/KULeuven-COSIC/SCALE-MAMBA

^{9.} https://github.com/data61/MP-SPDZ

have used the SCALE-MAMBA and MP-SPDZ Shamir [45] secret sharing based protocol in malicious adversarial settings in variations of number of parties. We have also used the existing ORAM module in MP-SPDZ framework.

The tests have been run on three Amazon AWS c4.2xlarge nodes and all the nodes were located in the same region. These nodes are provided with 15GiB DDR4 memory and an Intel Xeon E5-2666 v3 processor containing 4 cores and running at 2.9GHz. The latency between the nodes was measured to be roughly $150\mu s$, and the bandwidth between the nodes was measured to be 2.58Gbps. For ease of access, we have made our code and experiments available in a repository: https://github.com/StableMatch/OblivSM.

MP-SPDZ [31]. Our experiments in MP-SPDZ utilise Shamir-Secret Sharing over Arithmetic Fields (integer length is set to the default 64).

SCALE-MAMBA [4]. Our SCALE-MAMBA implementation uses the default secret-sharing and network settings.

7.3. Experimental Results and Analysis

In Table 5, we show the total execution time (in seconds) of our proposed oblivious stable matching and its variants for various set sizes using MP-SPDZ framework in 3-Party settings. For all set sizes, Algorithm 6 provides the best results (Algorithm 6 is nearly two orders of magnitude faster than the Näive oblivious Gale-Shapley stable matching in Algorithm 3).

TABLE 5: 7	Fiming (sec) results: ME	P-SPDZ
NULL OF L	Optimized Obligious	Lookup Tabla

	Naive Oblivious	Optimized Oblivious	LOOKUP TADIC
Set	Gale-Shapley	Gale-Shapley	Gale-Shapley
Size	(Algorithm 3)	(Algorithm 4)	(Algorithm 6)
2^{1}	0.06	0.05	0.04
2^{2}	0.49	0.28	0.19
2^{3}	4.23	1.44	0.70
2^{4}	25.31	5.35	3.07
2^{5}	216.39	18.68	9.92
2^{6}	1589.01	82.36	36.48
2^{7}	8658.98	218.56	132.09

In Table 6, we show the total execution time (in seconds) of our proposed oblivious stable matching and its variants for various set sizes using SCALE-MAMBA framework in 3-Party settings. For all set sizes, Algorithm 6 provides the best results (Algorithm 6 is nearly two orders of magnitude faster than the Näive oblivious Gale-Shapley stable matching in Algorithm 3).

TABLE 6: Timing (sec) re	esults: SCALE-MAMBA
--------------------------	---------------------

	Näive Oblivious	Optimized Oblivious	Lookup Table
Set	Gale-Shapley	Gale-Shapley	Gale-Shapley
Size	(Algorithm 3)	(Algorithm 4)	(Algorithm 6)
2^{1}	0.07	0.07	0.05
2^{2}	0.51	0.42	0.29
2^{3}	5.24	1.92	1.21
2^{4}	30.10	7.44	4.35
2^{5}	280.22	24.11	16.74
2^{6}	2031.44	191.69	76.61
2^{7}	12852.22	576.69	298.01

7.3.1. Comparison. In Table 7, we show a running time comparison between gale-shapley_tutorial [32] and our lookup table based Gale-Shapley stable matching (Algorithm 6) using MP-SPDZ. For all set sizes beyond 2, Algorithm 6 is better (in the case of set size 2^7 , we are nearly two orders of magnitude faster).

TABLE 7: Timing (sec) comparison between gale-shapley_tutorial [32] and Algorithm 6.

Set Size	gale-shapley_tutorial [32]	Algorithm 6
2^{1}	0.03	0.04
2^{2}	0.24	0.19
2^{3}	2.01	0.70
2^{4}	20.37	3.07
2^{5}	264.55	9.92
2^{6}	1679.87	36.48
27	10863.76	132.09

8. Conclusion

We present a set of novel secure stable matching algorithms. We implement and benchmark our algorithms using two popular MPC frameworks, SCALE-MAMBA and MP-SPDZ, for various set sizes. Our experiments show that our proposed "lookup table based oblivious stable matching" protocol (see Algorithm 6) is the most efficient.

References

- Atila Abdulkadiroğlu, Parag A Pathak, and Alvin E Roth. The new york city high school match. *American Economic Review*, 95(2):364–367, 2005.
- [2] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.
- [3] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In Ahmad-Reza Sadeghi, editor, Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers, volume 7859 of Lecture Notes in Computer Science, pages 239–257. Springer, 2013.
- [4] Abdelrahaman Aly, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Nigel P. Smart, and Tim Wood. SCALE and MAMBA documentation, 2018. https://homes.esat.kuleuven. be/~nsmart/SCALE/.
- [5] Abdelrahaman Aly, Kashif Nawaz, Eugenio Salazar, and Victor Sucasas. Through the looking-glass: Benchmarking secure multiparty computation comparisons for relu's. *IACR Cryptol. ePrint Arch.*, page 202, 2022.
- [6] Abdelrahaman Aly, Emmanuela Orsini, Dragos Rotaru, Nigel P. Smart, and Tim Wood. Zaphod: Efficiently combining lsss and garbled circuits in scale. In *Proceedings of the 7th ACM Workshop* on Encrypted Computing & Applied Homomorphic Cryptography, WAHC'19, page 33–44, New York, NY, USA, 2019. Association for Computing Machinery.
- [7] Abdelrahaman Aly and Mathieu Van Vyve. Securely solving classical network flow problems. In Jooyoung Lee and Jongsung Kim, editors, *Information Security and Cryptology - ICISC 2014* - 17th International Conference, Seoul, Korea, December 3-5, 2014, Revised Selected Papers, volume 8949 of Lecture Notes in Computer Science, pages 205–221. Springer, 2014.
- [8] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 274–302. Springer, Heidelberg, August 2019.



(a) Algorithm 3 vs. Algorithm 4.

(b) Algorithm 4 vs. Algorithm 6.

Figure 1: Execution Time (sec) using MP-SPDZ framework in 3-Party settings. x-axis denotes set size and y-axis denotes times.



Figure 2: Execution Time (sec) using SCALE-MAMBA (3-Party). x-axis denotes set size and y-axis denotes time.

- [9] Marina Blanton and Siddharth Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. In *European Symposium on Research in Computer Security*, pages 384–406. Springer, 2015.
- [10] Ismel Brito and Pedro Meseguer. Distributed stable marriage problem. In 6th Workshop on Distributed Constraint Reasoning at IJCAI, volume 5, pages 135–147. Citeseer, 2005.
- [11] Ran Canetti. Security and composition of multiparty cryptographic protocols. J. Cryptol., 13(1):143–202, 2000.
- [12] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings, volume 6280 of Lecture Notes in Computer Science, pages 182–199. Springer, 2010.
- [13] Octavian Catrina and Sebastiaan de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings, volume 6345 of Lecture Notes in Computer Science, pages 134–150. Springer, 2010.
- [14] 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 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Heidelberg, March 2006.

- [15] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, Advances in Cryptology -CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings, volume 2729 of Lecture Notes in Computer Science, pages 247– 264. Springer, 2003.
- [16] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, volume 7417 of Lecture Notes in Computer Science, pages 643– 662. Springer, 2012.
- [17] Siemen Dhooghe. Applying multiparty computation to car access provision. URL: https://www. esat. kuleuven. be/cosic/publications/thesis-296. pdf, last checked on, pages 08– 4, 2018.
- [18] Jack Doerner, David Evans, and Abhi Shelat. Secure stable matching at scale. In *Proceedings of the 2016 ACM SIGSAC Conference* on Computer and Communications Security, pages 1602–1613, 2016.
- [19] Jack Edmonds. Paths, trees, and flowers. Canadian Journal of Mathematics, 17:449–467, 1965.
- [20] Matthew Franklin, Mark Gondree, and Payman Mohassel. Improved efficiency for private stable matching. In *Cryptographers' Track at the RSA Conference*, pages 163–177. Springer, 2007.

- [21] Matthew Franklin, Mark Gondree, and Payman Mohassel. Multiparty indirect indexing and applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 283–297. Springer, 2007.
- [22] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9– 15, 1962.
- [23] Oded Goldreich. Towards a theory of software protection. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 426–439. Springer, Heidelberg, August 1987.
- [24] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.
- [25] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. J. ACM, 43(3):431–473, 1996.
- [26] Philippe Golle. A private stable matching algorithm. In *International Conference on Financial Cryptography and Data Security*, pages 65–80. Springer, 2006.
- [27] S Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 513–524, 2012.
- [28] Dan Gusfield and Robert W Irving. *The stable marriage problem:* structure and algorithms. MIT press, 1989.
- [29] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, ASIACRYPT 2017, Part I, volume 10624 of LNCS, pages 598–628. Springer, Heidelberg, December 2017.
- [30] Robert W Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48(3):261–272, 1994.
- [31] Marcel Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. Association for Computing Machinery, New York, NY, USA, 2020.
- [32] Marcel Keller. gale-shapley_tutorial.mpc, Accessed: 30-11-2022. https://github.com/data61/MP-SPDZ/blob/master/Programs/ Source/gale-shapley_tutorial.mpc.
- [33] Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multiparty computation of AES and DES using lookup tables. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings, volume 10355 of Lecture Notes in Computer Science, pages 229–249. Springer, 2017.
- [34] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.
- [35] Marcel Keller and Peter Scholl. Efficient, oblivious data structures for mpc. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 506–525. Springer, 2014.
- [36] John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In Peter Thiemann and Robby Bruce Findler, editors, ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012, pages 189–200. ACM, 2012.
- [37] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pages 253–262. IEEE, 2013.
- [38] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

- [39] Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 590–599. ACM, 2001.
- [40] Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In Annual cryptology conference, pages 502–519. Springer, 2010.
- [41] Prashanthi Ramachandran, Shivam Agarwal, Arup Mondal, Aastha Shah, and Debayan Gupta. S++: A fast and deployable securecomputation framework for privacy-preserving neural network training. *CoRR*, abs/2101.12078, 2021.
- [42] M Sadegh Riazi, Ebrahim M Songhori, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Toward practical secure stable matching. *Proc. Priv. Enhancing Technol.*, 2017(1):62– 78, 2017.
- [43] Alvin E Roth. The economics of matching: Stability and incentives. Mathematics of operations research, 7(4):617–628, 1982.
- [44] Alvin E Roth. Misrepresentation and stability in the marriage problem. *Journal of Economic theory*, 34(2):383–387, 1984.
- [45] Adi Shamir. How to share a secret. Commun. ACM, 22(11):612– 613, 1979.
- [46] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious ram with o ((logn) 3) worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*, pages 197–214. Springer, 2011.
- [47] Nigel P. Smart and Tim Wood. Error detection in monotone span programs with application to communication-efficient multi-party computation. In Mitsuru Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 210–229. Springer, Heidelberg, March 2019.
- [48] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious ram. arXiv preprint arXiv:1106.3652, 2011.
- [49] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. *IACR Cryptol. ePrint Arch.*, page 280, 2013.
- [50] Chung-Piaw Teo, Jay Sethuraman, and Wee-Peng Tan. Galeshapley stable marriage problem revisited: strategic issues and applications. In *International Conference on Integer Programming* and Combinatorial Optimization, pages 429–438. Springer, 1999.
- [51] Tadanori Teruya and Jun Sakuma. Round-efficient private stable matching from additive homomorphic encryption. In Yvo Desmedt, editor, *Information Security, 16th International Conference, ISC* 2013, Dallas, Texas, USA, November 13-15, 2013, Proceedings, volume 7807 of Lecture Notes in Computer Science, pages 69–86. Springer, 2013.
- [52] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the* 22nd ACM SIGSAC Conference on Computer and Communications Security, pages 850–861, 2015.
- [53] Andrew C Yao. Protocols for secure computations. In 23rd annual symposium on foundations of computer science (sfcs 1982), pages 160–164. IEEE, 1982.
- [54] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting square-root oram: efficient random access in multi-party computation. In 2016 IEEE Symposium on Security and Privacy (SP), pages 218–234. IEEE, 2016.
- [55] Samee Zahur, Xiao Wang, Mariana Raykova, Adrià Gascón, Jack Doerner, David Evans, and Jonathan Katz. Revisiting square-root ORAM: efficient random access in multi-party computation. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 218–234. IEEE Computer Society, 2016.