MPCircuits: Optimized Circuit Generation for Secure Multi-Party Computation

M. Sadegh Riazi, Mojan Javaheripi, Siam U. Hussain, Farinaz Koushanfar
University of California San Diego
La Jolla, CA, USA
{mriazi, mojan, siamumar, farinaz}@ucsd.edu

***Abstract*—Secure Multi-party Computation (MPC) is one of the most influential achievements of modern cryptography: it allows evaluation of an arbitrary function on private inputs from multiple parties without revealing the inputs. A crucial step of utilizing contemporary MPC protocols is to describe the function as a Boolean circuit. While efficient solutions have been proposed for special case of *two-party* secure computation, the general case of more than two-party is not addressed. This paper proposes `MPCircuits`, the *first automated* solution to devise the optimized Boolean circuit representation for any MPC function using hardware synthesis tools with new customized libraries that are scalable to multiple parties. `MPCircuits` creates a new end-to-end tool-chain to facilitate practical scalable MPC realization. To illustrate the practicality of `MPCircuits`, we design and implement a set of five circuits that represent real-world MPC problems. Our benchmarks inherently have different computational and communication complexities and are good candidates to evaluate MPC protocols. We also formalize the metrics by which a given protocol can be analyzed. We provide extensive experimental evaluations for these benchmarks; two of which are the *first* reported solutions in multi-party settings. As our experimental results indicate, `MPCircuits` reduces the computation time of MPC protocols by up to 4.2×.**

***Index Terms*—Multi-party computation, secure function evaluation, logic synthesis, secure auction, secure voting, private-set intersection, stable matching, nearest-neighbor search**

I. Introduction

Secure multi-party computation (MPC a.k.a., SMC) is one of the most influential achievements of modern cryptography. It provides a provably-secure method for multiple parties to jointly evaluate a function on their private inputs without disclosing the input values to each other. MPC protocols can be categorized into two main groups: protocols based on (i) the GMW (Goldreich-Micali-Wigderson) paradigm [1] and (ii) the Garbled-Circuit (GC) paradigm [2]. The original idea of two-party GC is later generalized for multi-party setting in the Beaver-Micali-Rogaway (BMR) protocol [3]. Both paradigms require the underlying function to be represented as a *Boolean circuit*. The tools and methods for Boolean computations of two-party protocols are available, but they are not readily scalable or available for multiple parties. Present ad-hoc realization of secure multi-party tasks do not provide a holistic tool usable for a variety of other MPC applications.

Two standing challenges that users face while utilizing MPC protocols are: (i) generating optimized Boolean circuits for the pertinent task, and (ii) the necessity of knowing the details of the protocol. The first complication often results in a high inefficiency in the protocol execution. The latter is even more critical, triggering possible security breaches if the exact protocol is not followed. Protocols that interpret a multi-party computation as multiple invocations of two-party secure computation are not just impractical, but also specifically susceptible to such breaches. Therefore, there is a need for an end-to-end solution that bridges the gap between usability and secure realization of MPC protocols.

During the past two decades, a number of practical realizations for the special case of *two-party* secure computation have been presented, reducing the execution time by several orders of magnitude [4], [5], [6], [7], [8], [9], [10], [11]. However, less focus has been on the general problem of secure *multi-party* (more than two) computation, even though many practical problems (e.g., auction and voting) are inherently multi-party and cannot be reduced to two-party settings.

To the best of our knowledge, there have only been three MPC realizations that support more than two parties [12], [13], [14]. Among them, only [14] supports circuit generation: the crucial first step of MPC. However, this framework precedes one of the most crucial MPC optimizations: free-XOR [15], which allows `XOR`, `XNOR`, and `NOT` gates to be evaluated without any communication or encryption. The two more recent frameworks [12], [13] support free-XOR but only focus on the specific (ad-hoc) protocol execution with no systematic solution for the circuit generation step.

In this paper, we present the *first automated* methodology to generate Boolean circuits, customized for MPC protocols with state-of-the-art optimizations. Inspired by TinyGarble [8], the most efficient Boolean circuit generator for the two-party setting, we leverage standard logic synthesis tools for this purpose. Note that two-party libraries such as TinyGarble cannot be used for the MPC problem since the synthesis technology libraries are not compatible with the MPC protocols. In addition, the order of logic computation (determined by the API and the Boolean netlist sorter) is radically different for two-party protocols. `MPCircuits` relies on designing new technology libraries for the logic synthesis tools customized for MPC protocols. Our solution can be integrated with any cryptographic back-end engine for the MPC protocol, e.g., the realizations in [12], [13], to allow users to perform a holistic secure multi-party computation.

This work also aims to facilitate future research on MPC. One of the key enablers in analyzing different protocols is to have a comprehensive set of benchmarks representing different applications/tasks. To do so, we compile a set of benchmarks that represent critical real-world MPC problems. Our benchmark set includes practical problems of auction, voting, and set intersection that have been evaluated in prior MPC literature. However, the methodologies presented in this paper for auction and voting do not need the presence of any trusted third parties as in the existing work [16], [17]. We also design circuits for two new benchmarks, namely, stable matching and nearest-neighbor search. Our circuits for stable matching and nearest-neighbor search are the *first* solutions in the multi-party setting. Each proposed benchmark captures a different set of requirements and domains, which ensures the applicability of `MPCircuits` to diverse scenarios.

We use a set of metrics to quantify the performance of `MPCircuits` on each of the benchmarks. These metrics encompass different characteristics of the MPC protocol, analyzing the performance of the solution for settings with varying computation power and communication bandwidth. We prototype our solution and perform extensive evaluations on our benchmarks for a range of parameter sizes. In brief, our three main contributions are as follows:

- Introducing the first holistic solution to automatically create optimized Boolean circuits for MPC. We provide a new technology library for hardware synthesis tools to generate circuits compatible with MPC. Our approach is modular and can produce efficient circuits for various functionalities.
- Designing and implementing circuits for five compelling secure multi-party computation tasks. These tasks represent five key MPC problems that cover most of the applications suggested in literature, namely, auction, voting, set intersection, stable matching, and nearest neighbor search. We further introduce metrics by which each benchmark could be analyzed for efficiency and scalability.
- Creating automated tools for end-to-end MPC realization, i.e., a simple-to-use API and interpreter programs to convert the generated netlist to formats usable by MPC back-end engines. Extensive experimental results are provided for the different parameter configurations confirming the efficiency and scalability of `MPCircuits`.

Code Availability: https://github.com/sadeghriazi/MPCircuits

II. Preliminaries

The methodologies presented in this paper are compatible with MPC frameworks based on both BMR [12] and GMW [13] protocols. For brevity, we only report proof-of-concept evaluations based on BMR [3]. This protocol has two main phases: garbling and evaluation. In the first phase, all parties jointly create the *garbled* version of the circuit. In the second phase, each party receives partial information from other parties and begins to evaluate the circuit locally. The garbling phase is usually the most costly stage in the protocol execution. However, since it is independent of the actual inputs from the participating parties, it can be pre-computed in advance.

Garbling. In this phase, all parties assign two random labels for every wire in the circuit, one for semantic value zero and one for semantic value one. We use notations consistent with [12]: $k^i_{w,a} \in \{0,1\}^\kappa$ denotes random label of wire w for the semantic value $a \in \{0,1\}$ held by party P_i $i = 1...n$ where n is the total number of parties. κ is the security parameter and is usually set to 128. For each *gate*, parties encrypt output labels using F^2, a double-key pseudorandom function and use two input labels as keys. Consider gate g, with two inputs u and v, and output wire w. For example, in the case of an `AND` gate, output label for semantic value 1 ($k_{w,1}$) is encrypted using the two input labels of semantic value 1 ($k_{u,1}$ and $k_{v,1}$). Since there are four possible input combinations for any two-input Boolean gate, parties create four different encryptions of the correct output label and their corresponding input keys. The collection of all four encrypted values is called a *garbled table*. More precisely, for every $a, b \in \{0,1\}$, the output label for $c = g(a,b) \in \{0,1\}$ is encrypted as

$$\left\{ \left(\bigoplus_{i=1}^{n} F^2_{k^i_{u,a},\ k^i_{v,b}}(n_g \circ j) \oplus k^j_{w,g(a,b)} \right) \right\}_{j=1}^{n} \tag{1}$$

where n_g is the unique ID number for a gate and $\circ$ denotes concatenation operation. In order to mask the relationship between labels and actual semantic values, each party also assigns a *permutation bit* λ^i_w and sets $\lambda_w = \oplus^n_{i=1} \lambda^i_w$. All four encrypted values are permuted according to permutation bits.

Evaluation. In order to transfer the labels associated with the true value of an input wire, the *Oblivious Transfer* (OT) protocol is used. In OT, one party holds two (or multiple) messages m_i and another party holds the selection bit(s) b. At the end of the protocol, the receiver gets m_b and learns nothing about other message(s) while the sender learns nothing about the selection bit(s). Given the collection of n keys for each input wire, all parties can decrypt one row of each garbled table (those connected to input gates) and generate the output keys of those gates. The evaluation process continues until output gates are reached. Therefore, the evaluation process can be computed locally once each party has the correct combination of all n keys for all input gates. Note that none of the intermediate values are revealed to any party. In fact, the semantic value of each wire is XOR-shared among all parties. All labels are unintelligible by themselves. At the end of the protocol, each party only sends her share of the output wires' labels such that everyone can locally compute the plaintext output result. Please see [12] for more detailed explanation.

Free-XOR Optimization. Kolesnikov et al. [15] proposed a method that eliminates the need for creating garbled tables for `XOR` gates, rendering them almost free of cost. To utilize this technique, each party P_i needs to create a one-time random number $R_i \in \{0,1\}^\kappa$. Same as before, $k^i_{w,0}$ is generated randomly but $k^i_{w,1}$ is set to $R_i \oplus k^i_{w,0}$ for every wire. Due to this correlation of labels, the output label of each `XOR` gate can be computed by XORing the two input labels without any communication between parties.

III. Automated Circuit Generation

The first, and one of the most critical, step in practical secure realization of a function through MPC protocols is generation of the Boolean circuit that describes the pertinent functionality. We now elaborate on our methodology for automatically creating the Boolean circuit such that it is optimized for MPC as well as making it compatible with any given realization of the protocol. The corresponding steps of the circuit generation in MPCircuits are illustrated in Figure 1.

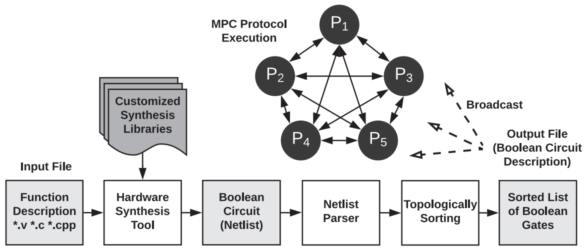
MPC Protocol Execution
P1
P2
P3
P4
P5
Broadcast
Output File (Boolean Circuit Description)
Customized Synthesis Libraries
Input File
Function Description *.v *.c *.cpp
Hardware Synthesis Tool
Boolean Circuit (Netlist)
Netlist Parser
Topologically Sorting
Sorted List of Boolean Gates

Fig. 1: Global flow of MPCircuits circuit generation.

In the first step, the user writes the function description in a Hardware Description Language (HDL), e.g., Verilog. With recent progress in High-Level Synthesis (HLS) tools, it is also possible to develop the function in C and convert it to HDL using these tools. Inspired by TinyGarble [8], we employ logic synthesis tools to compile the HDL source code using our customized libraries. Thus, we generalize the idea of TinyGarble for multi-party (more than two-party) secure execution. Recall that in contrast to non-XOR gates, XOR gates do not require communication or encryption during garbling/evaluation due to the free-XOR technique [15]. As illustrated in [12], the number of non-XOR gates determines the computation and communication cost. Therefore, MPCircuits objective function is to minimize the *total number of non-XOR gates* ($n_{non-\texttt{XOR}}$) in the circuit description.

We re-define the problem of minimizing $n_{non-\texttt{XOR}}$ as a special case of logic optimization to be performed by the synthesis tools. These tools have been subject to more than two decades of improvement leading to sophisticated algorithms to minimize a given constraint, e.g., power consumption, circuit area, or critical path. We develop new synthesis and cell libraries to be utilized by such logic synthesis tools. Our synthesis libraries incorporate the realization of basic arithmetic and logic operations (add, subtract, multiplication, division, if-else, etc.) using minimum number of non-XOR gates. The cell library in MPCircuits, which is used for ASIC mapping, contains Boolean logic gates AND, shifted_AND, XOR, and XNOR. The area of the XOR and XNOR gates are set to zero and that of the other gates to one and the constraint is set to *minimizing the total area of the circuit*. As a result, the synthesis process minimizes the total number of non-XOR gates in the final netlist. This process *automatically* generates an optimized Boolean circuit for the BMR protocol. Note that the output of the TinyGarble framework cannot be used for the MPC protocols since the produced netlist contains logic gates that are not supported by MPC realizations [12].

The logic synthesis tool outputs a standard Verilog netlist containing cells that are included in the cell library. In order to use the netlist in a MPC protocol, one has to perform certain post-synthesis steps. This translation involves three steps: (i) a parser reads the Verilog netlist and converts it to an array of structures defining the logic operation of a particular gate and its input/output connections, (ii) a scheduler reads the connection information of the gates and topologically sorts them based on the dependencies, (iii) the sorted array of gates is written to a file in a specific format. This file is public and is sent to all the parties participating in the secure protocol. Note that, only the last step is specific to one particular MPC realization (BMR protocol in our implementation). Therefore, our methodologies can be applied to a large variety of secure protocols that are based on a Boolean netlist (e.g., the framework in [13]) with a simple modification to the last step.

Our proof-of-concept implementation is based on the BMR realization presented in [12]. To the best of our knowledge, this is one of the only two MPC frameworks that supports the free-XOR [15] optimization. Moreover, authors have optimized the BMR protocol to have a *constant number of rounds* which is strongly preferred in Internet settings where the communication delay is significant. In [12], most of the computation can be shifted to the offline phase and precomputed in advance. We utilize the SCAPI [18] library (source code of [12]) in our implementation. This library supports only five types of Boolean logic. Therefore, our cell library contains only those five gates. However, this constraint does not increase the number of non-XOR gates and only increases the number of XOR gates. Incorporating the other gates in the cell library is straightforward, in case they are supported by another realization of an MPC protocol.

Security Model. The BMR protocol in [12] is provably secure in the Honest-but-Curious (HbC) security model. In this model, all participating parties follow the protocol but they can attempt to extract more about the other parties' input from the information they send and receive. There are two variants of HbC adversary model: honest-majority and dishonest-majority. In the former model, the protocol is secure as long as the majority of the parties are not corrupted. In the latter model, *any* number of corrupted parties cannot infer information other than what can be inferred from the outcome of the computation. For example, the FairplayMP [14] framework is secure for an honest majority. In addition, protocols in HbC model are generally orders of magnitude faster than the ones in *malicious* model [12] in which parties can deviate from the protocol anytime. HbC model is the building block for stronger security models such as security against malicious adversaries. MPCircuits can automatically generate optimized Boolean circuits for both security models. Note that the methodologies presented in this work do not change the security or correctness of the MPC protocol. MPCircuits provides an automated solution to generate Boolean circuit representations that have minimal number of non-XOR gates, thus, improving the *efficiency* of the MPC protocol.

IV. Circuit Designs

We design and implement a set of Boolean circuits for five compelling MPC tasks and report the experimental results for all of them using our circuit synthesis approach. Three of these applications, i.e., Auction, Voting, and PSI have been studied in the prior literature and we compare `MPCircuits` with state-of-the-art solutions. In `MPCircuits`, we have provided the *first secure solution* for K-nearest neighbor search (K-NNS) and Stable Matching. Not only do these two benchmarks address real-world needs, but each of them captures an important criteria: (i) in K-NNS (unlike other benchmarks) only one party receives the result of the protocol and therefore makes the process of Oblivious Transfer (OT) asymmetric since only one party evaluates the garbled circuit and only $(n-1)$ OTs are performed (compare to $(n-1)^2$); this is the first reported instance of an asymmetric OT. (ii) In Stable Matching, the size of the circuit grows with $\mathcal{O}(n^4)$. In all other benchmarks the point-to-point communication rapidly becomes the bottleneck. Therefore, they cannot be used to validate other properties such as the scalability of the circuit generation. Evaluating an MPC framework on the Stable Matching benchmark illustrates the scalability of the framework in terms of circuit generation since the number of gates rapidly increases with the increase in the number of parties and soon becomes the bottleneck.

V. Auction

We design a Boolean circuit for auction in which the highest bidder is selected and pays the bid value. In traditional mechanisms, the auction is processed by a third party and all of the bid values are revealed to the third party. This raises many privacy concerns such as the possibility of information leakage or collusion between one of the bidders and the executing party. Here, we implement and analyze both types of auctions and illustrate the practicality and scalability of our solution.

Input: Each party P_i holds a bid_i, $i = 1...n$.
Output: The index (ID) of the highest bidder i_{max} and the highest bid value $x_{pay} = max(x_1, ..., x_n)$.
Parameter(s): Number of bits b to represent a bid.

Circuit Design: Figure 2 shows the internal architecture of the Boolean circuit for auction. The inputs to the circuit are the bids (upper side) and the output is the maximum bid value and the index (ID) of the winner. The darker blocks correspond to inputs and outputs of the circuit. Bids are compared in pairs using the comparison (`CMP`) blocks. The maximum value is then passed on to the next *layer* and so forth. A simple solution to compute the winner ID (WID) is to pass along the index of the higher bid at each stage. However, this solution requires $\mathcal{O}(n\ lg(n))$ `AND` gates where $lg(.)$ is the base-2 logarithm.

We propose a more optimized approach that requires $\mathcal{O}(n)$ number of `AND` gates. While the complexity does not change significantly, in practice, the number of `AND` gates is reduced by a factor of $lg(n)$. Our method is based on re-using the output of the `CMP` blocks. Consider the last `CMP` block at the bottom right corner. Depending on the output of this block,

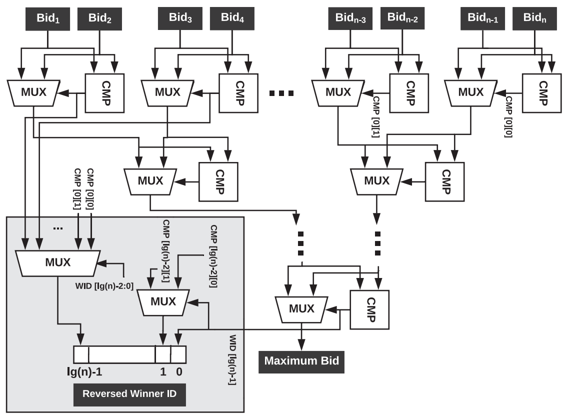

Fig. 2: Boolean circuit for auction.

one can identify the most significant bit (MSB) of the WID. For example, if the output is 1, it shows that the winner is from the second half of the bidders and hence, the WID starts with one. We can *recursively* continue this approach and depending on the already computed WID digits, the next digit is selected. More precisely, if we denote the output of the j^{th} `CMP` block at layer l by CMP[l][j], we have

$$\text{WID}[\alpha] = \text{CMP}[\alpha - 1][\text{WID}[\alpha - 1 : 0]]$$

At the end, WID holds the ID of the winner in reversed order (from least significant bit to most). The complexity of the total number of `AND` gates in the circuit is $\mathcal{O}(n\ b)$.

VI. Voting

A secure voting mechanism can preserve the privacy of all voters in an election. This, in turn, can replace old solutions based on anonymization and law-enforcement. The aforementioned solutions are all centralized and have a single point of failure. A modern secure voting mechanism can ensure the correctness of the election and privacy of voters even in the presence of any set of corrupted parties.

Input: Each party P_i holds the index of the candidate to whom she wants to vote ($vote_i$).
Output: The index of the candidate, n_h, with the highest vote.
Parameter(s): Number of candidates n_c.

Circuit Design: The internal design of the Boolean circuit for voting is illustrated in Figure 3. The inputs to the circuit are n votes, each representing the index of a candidate ($lg(n_c)$-bit). Each vote is an input to a "$lg(n_c)$ to n_c" Decoder (`DEC`) module. Therefore, based on the vote value, only one of the output lines of the DEC is set to one. These output wires are then connected to a `COUNT` module to count the number of votes. The `COUNT` module is implemented as a binary tree of `ADD` blocks. At each level of the binary tree, the operands' bit-length of the `ADD` blocks are set to the minimal value that can accumulate the result. Hence, the `COUNT` module results in an efficient realization that only requires $x-1$ non-XOR gates for counting the number of ones in a binary array of size x. The final step is to find the maximum number of votes. This task can be implemented using the *auction* module (Section V), where the bids are vote-counts. Finally, the output of the circuit is the ID of the winner candidate (WID).

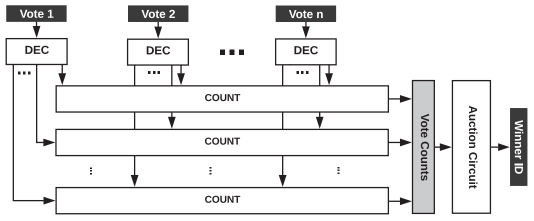

Fig. 3: Boolean circuit for voting.

In case of a tie in the vote-counts, our circuit outputs the first candidate as the winner. If another tie-breaking mechanism is needed, our solution can easily be modified due to its modular structure. The complexity of the total number of AND gates in each of the circuit components are $\mathcal{O}(n\ lg(n_c))$ (decoders), $\mathcal{O}(n)$ (COUNT modules), and $\mathcal{O}(n_c\ lg(n))$ (auction circuit). The overall circuit complexity is $\mathcal{O}(n\,lg(n_c)+n_c\,lg(n))$ which scales linearithmicly with n and n_c.

VII. Set Intersection

Private Set Intersection (PSI) allows two or multiple parties to obtain the elements at the intersection of their sets without revealing the other elements that are not in common. For example, multiple people can identify their mutual contact profiles/friends by inputting their contact list to the PSI protocol without revealing the rest of their contact lists. At the end of the protocol, only the mutual list of all parties is revealed.

In E-commerce, an online advertisement agency and a company can participate in the PSI protocol where the advertisement agency inputs its list of all the people who have been shown the ads of the company. The second set of inputs to the protocol is the list of the people who have bought the products provided by the company. At the end of the PSI protocol, both entities know how many people have bought the product as a result of seeing the advertisement. This provides a way to understand the effectiveness of the advertisement for the company. Note that the same process could not be realized in plaintext due to various privacy/security reasons. Revealing such information is privacy invasive and can damage the reputation of both the companies. In addition, disclosing customer's data might be against the law in some situations.

Input: Party P_i holds a set $S_i \subset \Omega$ where Ω is the universal set.
Output: The intersection set $S = \cap_{i=1}^{n} S_i$.
Parameter(s): The size of the universal set Ω or equivalently the number of bits required to describe an element in the universal set $b = \lg|\Omega|$. Maximum number of elements in each party's set m.

Circuit Design: Two different implementations are provided for PSI: a Bitwise-AND based circuit and a Sort-Merge-Compare-Shuffle (SMCS) based circuit. The first one is more efficient for scenarios in which Ω is small whereas the second approach is more suitable when m is small and Ω can be very large. Note that sets are represented differently in the two implementations as we explain in each section.

Bitwise-AND. In this implementation, each set is equivalent to a binary vector. The binary value at index j denotes the presence of the j-th element in a given set. Therefore, each set is represented as a $|\Omega|$-bit binary vector. The intersection set S is computed as *bit-wise* AND between all of the sets provided by all parties. As a result, the complexity of the circuit is $\mathcal{O}(n\ |\Omega|)$, linear in both the number of parties and the size of the universal set; but *independent* from the number of elements in each parties' set m.

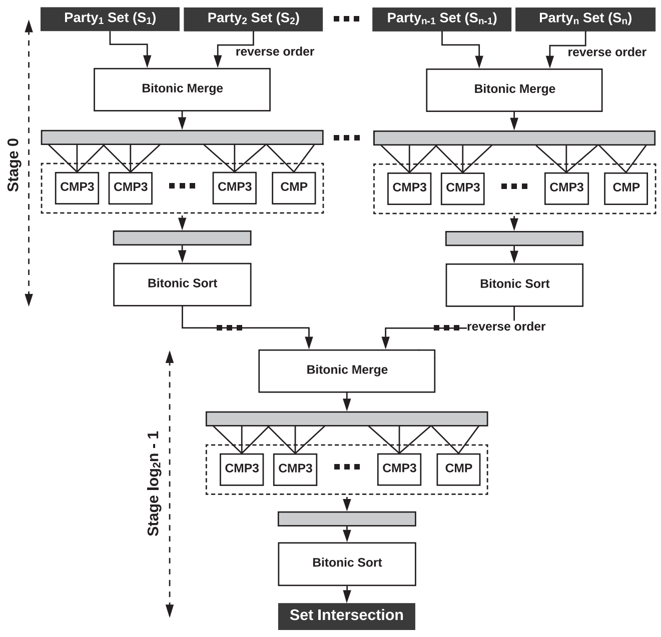

Fig. 4: High-level circuit description of the Sort-Merge-Compare-Shuffle for Private Set Intersection. Three operations are performed at each stage: merge, compare, and sort.

Sort-Merge-Compare-Shuffle (SMCS). In scenarios where $m << |\Omega|$, more efficient solutions than Bitwise-AND can be devised. Here, we present one of the most complicated circuits in our benchmarks which is the generalization of the approach presented in [19] from two-party setting to any n-party case. As the input to this circuit, each set is represented as a vector of m integers where each integer is b-bit. We will first explain the solution for two sets only. The intersection of two sets can efficiently be computed using three operations: sort, merge, and compare. First, each of these two sets should be sorted. Then by merging the two sorted sets, all elements in common will be brought together. Finally, by comparing adjacent elements, one can find the common elements in both sets. Since the set intersection is an associative operation, one can express the set intersection of n sets as a consecutive set intersection of two sets until reaching the final result. Therefore, the SMCS circuit has a binary tree structure where at each node, the intersection of two sets are computed. The final node computes the final intersection of all sets. Note that the first sort operation can locally be computed by each participant since it is independent of the other parties' private data. A final shuffle operation is needed in order to eliminate the information leakage which we describe later in this section. Without loosing any generality, assume that the number of sets (participants) is a power of two. If this is not the case, dummy nodes can be avoided in the tree structure. Please see Figure 4 for a high-level description of the SMCS circuit.

We now elaborate on each part of the SMCS circuit. The challenge is that the merger and sorter circuits should have a *fixed* structure and *non-random access* to the intermediate values since random access is a very costly operation in the MPC protocols. We rely on the *bitonic* merger and sorter circuits that satisfy this condition. Bitonic sort is one of the sorting networks that is an efficient circuit-based realization of a sorting algorithm. Input numbers are given to the circuit and after series of conditional swap operations, a sorted list is given as the output of the circuit. The only operation used in the circuit is conditional swap: given two input numbers, swap them if they are not sorted and do not swap them otherwise. The bitonic sort has a recursive structure. It first sorts each half of the input and then merges the two sorted lists. The base case is a circuit that sorts only two numbers which is equivalent to a conditional swap module. Our implementation of the bitonic sort circuit is also a recursive hardware description code.

The second half of the bitonic sorter represents the *bitonic merger* circuit. The input to the bitonic merger must be a bitonic sequence. A sequence x_i of numbers is called bitonic if for some k ($0 \leq k < m$):

$$x_0 \leq x_1 \leq ... \leq x_k \geq ... \geq x_{m-1} \geq x_m$$

or a circular shift of such sequence. Therefore, before merging the two sorted lists, one needs to reverse order the second list such that the concatenation of two lists be a bitonic sequence. This reverse-ordering should take place for input sets as well as for intermediate sets. Note that the reversing the order of a set does not incur any computation or communication cost and is realized as changing the order of wires in the circuit.

The second layer in the SMCS circuit is the *comparison* layer. After the merger layer, all identical elements in both sets are now beside each other. An intuitive solution is to have a series of comparison blocks that compare every two adjacent elements. However, it has been shown that having a 3-input comparison block as follows is more efficient [19]:

$$\text{CMP3 } (x_1, x_2, x_3) = \begin{cases} x_2 & if\ x_1 = x_2 \mid x_2 = x_3 \\ 0^b & otherwise \end{cases}$$

Given an array of $2m$ elements, we only need $m-1$ CMP3 blocks and one CMP block (compared to $2m$ CMP blocks).

The output of the comparison layer is an array of m numbers consisting of 0^b and the elements in the intersection of two sets. Before proceeding to the next stage (and similar to the first stage), the array has to be sorted. Note that the intermediate sets should not be revealed to any party since some information about the private input sets will be learned by other parties. Therefore, in contrast to the first stage, the sets should be sorted inside the MPC protocol.

At the end of all stages, the final set should be shuffled prior to be revealed in plaintext to all parties. This step is necessary because the final set potentially has a sequence of 0^b between two common elements. The position of zeros (0^b) reveal the distribution of elements that were not in the intersection and belong to one (or multiple parties) only.

The shuffling layer can be realized using Waksman permutation network [20] which takes as input an array and shuffles them based on the *control bits*. One of the parties is required to provide these control bits as well. However, this task makes one of the parties to have more control in the secure computation. For example, a dishonest party that is selected to provide the control bits can simply put all of them as zero which makes the shuffle layer ineffective and he can learn some information. As a result, we devise another solution that is secure but does not require more input from any party. The solution is to simply sort the final list before revealing it in plaintext. This approach is secure since all of the 0^b elements are brought together. More precisely, in all of the scenarios that the common elements are fixed, the final sorted set remains the same and an adversary cannot distinguish different scenarios. The overall complexity of the SMCS circuit is $\mathcal{O}(n\, m \lg^2 m\, b) = \mathcal{O}(n\, m \lg^2 m \lg|\Omega|)$ (compare with Bitwise-AND circuit with complexity $\mathcal{O}(n\,|\Omega|)$).

Modular Structure. One of the advantages of using a generic secure multi-party computation protocols such as BMR is its modular nature and flexibility. Unlike customized protocols, additional functionalities and computations can be augmented to the circuit seamlessly. For example, and auditing step can be added before releasing the final result: the intersection set is revealed if and only if the number of elements in common is less than a threshold. Such auditing steps are favorable especially when Ω is small and an adversary can easily put his input set as the universal set in which case, he clearly learns the intersection of all other sets. As another example, it is very straightforward to build other variants of PSI such as PSI-Cardinality which only outputs the size of the intersection and not the elements.

VIII. Stable Matching

Stable matching is the process of assigning the members of two groups to each other (one-to-one) where each person has a preference list. This assignment should satisfy the *stability* condition after the assignment: no two individuals should prefer to be matched with each other compared to their already assigned partners. In other words, the assignment is stable in a sense that no rematching will occur even if individuals are free to do so. Stable matching is one of the most complicated task in secure computation because of the complex and data-dependent memory accesses during the computation [21], [22].

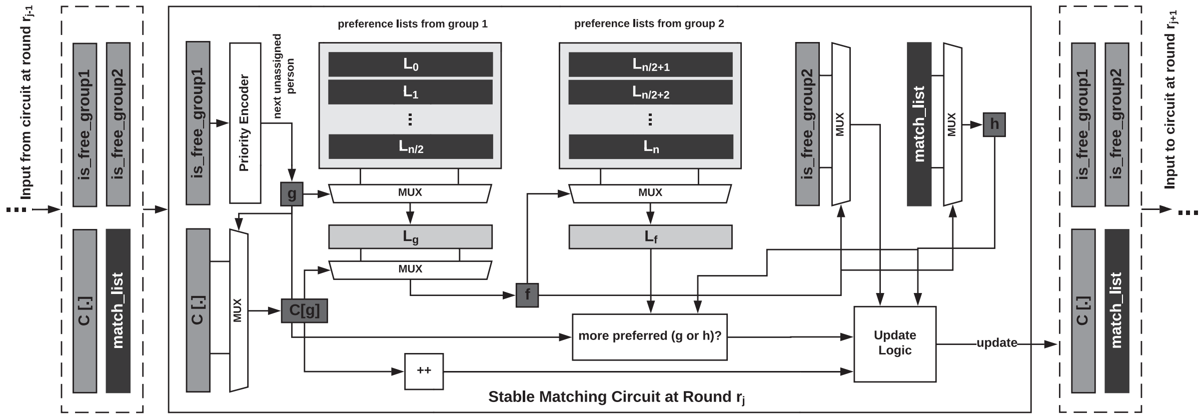

Fig. 5: The circuit for stable matching unrolled for round r_j. The circuit takes as input the intermediate values from previous round r_{j-1}, processes the current round based on the preference lists, and outputs the updated values.

Accessing the memory when the address is a secret value is a very costly operation in secure computation since the actual value of the address should remain private. In order to realize this constraint, random access to the memory is implemented using multiplexers inside the garbled circuit. Hence, the address as well as the accessed data remain private since they are processed inside the garbled circuit.

In secure stable matching, the match list is computed while keeping the preference lists private to their respective owners. This problem has been studied in the recent literature [21], [22] where the secure stable matching problem is reduced to a two-party secure computation scenario. Each individual XOR-shares her preference list and sends it to two non-colluding servers who perform the secure computation. However, stable matching is inherently a *multi-party* problem and the assumption of two non-colluding servers may not be feasible in practice. To the best of our knowledge, we provide the *first* solution for multi-party secure stable matching.

Input: Party P_i holds a preference list L_i with size of m. Each list is an array of $\lg(\frac{n}{2})$-bit numbers (IDs of the other group's members) sorted from the most preferred to the least.
Output: The match list: an array of size $\frac{n}{2}$ where each number is $\lg(\frac{n}{2})$-bit.
Parameter(s): The size of the preference list m.

Circuit Design: Gale and Shapley [23] were first to formalize the stable matching problem and proposed an algorithm that can find the matching list. During the computation, the matching list stores the temporary assignment of individuals. The algorithm works as multiple rounds. In each round r_j, an unassigned individual from group 1, say g, is selected. The circuit identifies if this individual can be assigned to the most preferred person given the preference list of g. Note that in this algorithm, each element of the preference list is accessed only once. If the match is not accepted, next preferred ID is selected in future rounds. The number of attempts for each individual from group 1 is stored as an array of counters C, i.e., $C[i]$ denotes the number of attempts for individual i.

Assume that at round r_j, unassigned individual g from group 1 is selected. The circuit first accesses the C array and finds the next preferred individual that is not already processed, let's call that ID f, $f = L_g[C[g]]$. The circuit increases the number of attempts for g by incrementing $C[g]$. If f is unassigned, the circuit assigns f from group 2 to g from group 1. Otherwise, it is necessary to determine whether f prefers g or his already assigned person h from group 1. The decision can be made by comparing the index of h and g in the preference list of f. That is, comparing $index\_of(h)$ and $index\_of(g)$ in L_f. If $index\_of(g)$ is less than $index\_of(h)$ in L_f, it means f prefers g over h. In this case, f is assigned to g and h gets unassigned and nothing happens otherwise.

In the next round, another unassigned individual from group 1 is selected and the same computation is performed. The process continues until all elements in the preference lists are processed. Note that this algorithm can be realized using a sequential circuit [21] but since there is currently no methodology for making the BMR protocol compatible with sequential circuits, we need to unroll the circuit as depicted in Figure 5. The overall complexity of the combinational circuit for secure stable matching is $\mathcal{O}(n^3\ m\ lg(n))$.

IX. Nearest-Neighbor Search (NNS)

In this benchmark, each party holds an attribute value v_i and one of the parties (P^*) is interested to learn which party (or parties) has the most similar (closest) attribute to her, given a certain similarity metric. NNS has many applications in classification, data mining, recommender systems, and proximity search. A privacy-preserving solution enables a client to find the most similar profiles without revealing her attribute. For example, consider an online dating website where each person creates a profile containing sensitive information about his/her age, personal preferences, and the zipcode of where she/he lives. Today, a centralized server knows all of the clients' information and provides the match result to each party. This computation model is also prone to internal and external attacks where clients' sensitive information is revealed to the attacker. In contrast, we propose a privacy-preserving method that is decentralized and is provably secure even if all other parties collude. Our methodology is modular and can be realized for any similarity metric since only the comparison module has to be modified.

Input: Party P_i holds an attribute value v_i.
Output: The set of k_n closest (most similar) attributes to party P^* input (v^*) given a similarity function $sim(.,.)$.
Parameter(s): Number of bits b required to represent each attribute. Number of nearest neighbors to be found (k_n).

Circuit Design: Our implementation for NNS is the generalization of the combinational circuit in [24] that computes 1-nearest neighbor search. We generalize the combinational circuit for any value of k_n. The core block of the design is a module that takes as input k_n distance values along with a new distance value and outputs the corresponding k_n minimum distances from the total $k_n + 1$ inputs. This module is instantiated n times where each instance processes the input from one party. Please note that in contrast to the two-party garbled circuit protocol, there is no known solution to use the sequential circuit in the general multi-party variant. Therefore, sequential circuits provided for k-nearest neighbors cannot be used. The overall complexity of the combinational circuit for NNS is $\mathcal{O}(n\ k_n\ b)$. `MPCircuits` supports the distance computation module to be any distance metric, e.g., the Hamming distance, euclidean distance, edit distance, or taxicab distance. In the following, the reported experimental results correspond to the Hamming distance.

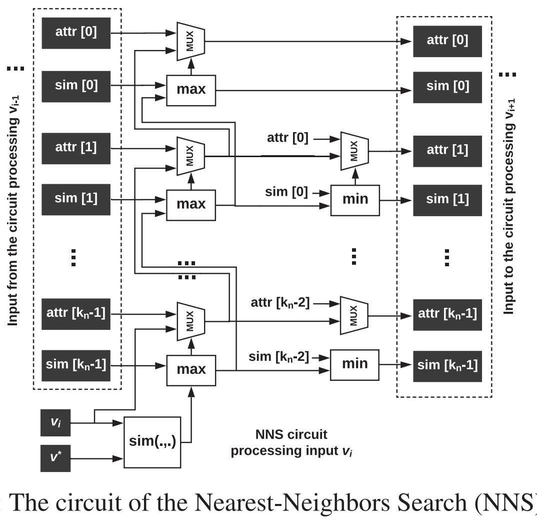

Fig. 6: The circuit of the Nearest-Neighbors Search (NNS) that finds the k_n most similar attributes to v^*. In the experimental results, the circuit is unrolled for n times.

X. Experimental Results and Related Work

We first discuss the metrics by which we characterize each application. We outline the metrics and the reason for their importance in practical realization of the MPC protocols.

- Execution time (T): The total execution time of the protocol comprises the time required for garbling/evaluating the circuit (T_{GE}) as well as time spent on the communication T_C. In a general case, these two can overlap in time depending on whether the implementation is pipelined/multi-threaded or not and hence, $T \leq T_{GE} + T_C$. The distinction between the two timing parameters is important since T_{GE} mostly depends on the computational power, whereas, T_C depends on the network quality (delay and bandwidth).
- Communication ($Comm$): Maximum number of bytes exchanged between any two parties. The "maximum" is required for protocols in which communication between parties are asymmetric. In the BMR protocol, the communication between each two parties can be computed as the multiplication of number of non-`XOR` gates, a constant factor (=9), number of parties minus one ($n-1$), and the bit-length of each wire label (usually 128).
- Memory footprint and scalability (Mem): One of the important characteristics for each MPC protocol is the amount of memory allocated in the end-to-end execution. Protocols/frameworks that consume a high volume of memory have limited scalability in real-world scenarios where the input size from each party is large.

Experimental Setup. The experiments are performed on a server equipped with 24 core Intel(R) Xeon(R) E5-2650 v4 @2.20GHz CPU with 256GB of RAM. We run all n parties in the same LAN network with 20ms round-trip latency and 10Gbps bandwidth. Synopsys Design Compiler 2015.06-SP2 is used to synthesize the Boolean circuits. RC is constant in all of our benchmarks for different values of parameters since our prototype implementation is based on [12] which has a constant round complexity. The SCAPI library utilizes Advanced Encryption Standard (AES) encryption and naturally benefits from the AES-NI which is supported by our machine. In our experimental results, we have used built-in Ubuntu `time` tool with `-f '%M'` flag to determine the memory footprint.

Auction. We perform experiments for different numbers of participants (n) in the auction for two values of b. Table I shows the results. As can be seen, the optimized Boolean circuits using `MPCircuits` technology libraries reduce the number of `AND` gates by 3.3×. Bogetoft et al. [16] have proposed a solution for secure auction. Their solution is based on multiple "Trusted Third Parties (TTPs)". TTPs compute the true outcome of the auction on behalf of the bidders. In this computation model, if all TTPs collude, the real input of all parties are revealed, whereas, in our approach, all parties securely process the auction and even if all other parties collude, nothing is revealed. The approach of [25] also requires a separate party called "Auction Issuer". The methodology in [26] additionally requires outsourcing the computation to two TTPs. Larson et al. [27] design a method based on a verifiable secret sharing scheme. The drawback of their approach is that not all participants in the auction are involved in the secure computation protocol and the security relies on the evaluators. Therefore, our solution is the *only* solution that (i) has constant round complexity and (ii) guarantees security even for cases where all other parties are corrupted.

TABLE I: Secure Auction.

		Non-optimized		**Optimized**						
b	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
16	4	69	324	261	97	0.74	0.62	2.39	0.04	10.25
	8	140	761	600	228	1.69	1.91	6.62	0.22	10.29
	16	281	1638	1281	492	3.51	4.48	15.06	1.01	18.14
32	4	133	660	534	194	0.74	0.66	3.41	0.08	10.31
	8	269	1547	1229	454	1.66	1.83	6.50	0.44	10.36
	16	539	3324	2621	975	3.48	4.34	16.85	2.01	30.65

Voting. Table II shows the experimental results for different number of parties (voters) and candidates. As can be seen, `MPCircuits` is between 1.4-2.7× more efficient compared to standard utilization of logic synthesis tools. Civitas [17] is a secure voting system which is verifiable and coercion-resistant but requires five different type of agents for its execution. Fujioka et al. [28] also propose a solution for secure auctions but it requires two additional entities called administrator and the counter conspire. In contrast, our solution does not involve any additional agents or entities.

TABLE II: Secure Voting.

		Non-optimized		**Optimized**						
n_c	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
2	8	7	17	18	8	1.57	1.77	9.35	3.3 KB	10.09
	16	19	43	45	16	3.29	4.29	13.76	0.02	10.09
4	4	17	50	23	37	0.71	0.54	3.25	0.02	10.09
	8	49	128	105	79	1.64	1.80	6.46	0.08	10.08
	16	123	294	249	147	2.99	4.11	14.23	0.30	10.08
8	16	250	739	545	388	3.40	4.01	15.40	0.80	15.30

Private Set Intersection. Table III shows the experimental results of Bitwise-AND circuit for different sizes of the universal set and different numbers of parties. For all PSI experiments, parameter m is set to 16. The corresponding results for the SMCS circuit are shown in Table IV. As can be seen, the optimized Boolean circuits using `MPCircuits` technology libraries reduce the number of `AND` gates by 4.2×.

There has been an extensive research focus on the Private Set Intersection (PSI) problem for a two-party situation [29], [30], [19]. In [19], authors propose a method for two-party PSI based on garbled-circuit approach. To the best of our knowledge, the only solution that is proposed for secure multi-party private set intersection is a recent work by Kolesnikov et al. [31]. Their approach is a customized solution that is optimized only to perform PSI in an identical security model as this work. Their computation platform is comparable but more powerful than ours. In the LAN setting, for a set size of 2^{16} and 10 parties, their total running time is 12 seconds with 23MB of communication. Whereas, for a universal set of size 10^5 ($\sim 2^{17}$) and 8 number of parties, our running time is 24 seconds with 314MB of communication. Although our solution is less optimized, we want to emphasize that we have proposed a generic solution to create any functionality, whereas, their solution is specially optimized for PSI. In addition, our solution has a very modular structure and can easily be modified to support other variants of the PSI, e.g., PSI cardinality in which only the *number* of mutual elements is revealed. Moreover, in Bitwise-AND circuit, the actual size of each party's set is not revealed since the inputs are fixed-length binary vectors.

TABLE III: Private set intersection (Bitwise-AND variant).

		Non-optimized		**Optimized**						
$\lvert\Omega\rvert$	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
10^4	4	0	3.00E+04	0	3.00E+04	0.94	0.69	3.89	12.36	65.24
	8	0	7.00E+04	0	7.00E+04	2.73	1.99	9.46	67.29	403.94
	16	0	1.50E+05	0	1.50E+05	12.61	4.74	30.46	308.99	2835.59
10^5	4	0	3.00E+05	0	3.00E+05	1.99	0.88	6.80	123.60	584.44
	8	0	7.00E+05	0	7.00E+05	11.82	2.89	24.05	672.91	3892.61

TABLE IV: Private set intersection (SMCS variant).

		Non-optimized		**Optimized**						
b	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
16	4	1.05E+04	7.77E+04	5.02E+04	1.86E+04	0.82	0.56	3.52	7.66	52.57
	8	2.42E+04	1.81E+05	1.16E+05	4.30E+04	2.42	1.76	6.59	41.37	280.42
	16	5.15E+04	3.88E+05	2.48E+05	9.19E+04	9.65	4.40	41.50	189.34	1843.72

Stable Matching. Table V shows the circuit size as well as the experimental results for different group sizes and preference list lengths. As can be seen, the optimized Boolean circuits generated by `MPCircuits` technology libraries have 1.6-2.4× lower number of `AND` gates. To the best of our knowledge, there has been no prior solution for multi-party secure stable matching. State-of-the-art solutions reduce the task to two-party secure computation problem [21], [22]. All parties outsource the computation to two servers which are assumed to not collude. While these solutions can scale to bigger set sizes, they rely on additional servers to find the match list on their behalf. If two servers collude, they can learn the preference list of all individuals in plaintext. In contrast, our security model is much stronger where *any* number of corrupted parties cannot learn the preference list of other individuals and the solution does not require additional servers for the computation.

TABLE V: Secure stable matching.

		Non-optimized		**Optimized**						
m	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
2	8	1.83E+02	1.11E+03	7.28E+02	5.35E+02	1.35	1.54	6.96	0.51	10.42
4		7.51E+02	4.55E+03	2.95E+03	2.24E+03	1.45	1.54	7.05	2.16	21.39
3	12	1.69E+03	9.61E+03	5.21E+03	6.03E+03	2.35	2.52	10.94	9.11	84.50
6		5.48E+03	3.10E+04	1.74E+04	1.90E+04	2.94	2.57	11.83	28.63	230.88
4	16	4.22E+03	3.70E+04	2.38E+04	1.67E+04	4.13	3.74	15.35	34.32	341.76
8		1.18E+04	1.11E+05	7.32E+04	4.66E+04	5.97	4.04	18.23	95.95	920.10

Nearest-Neighbor Search. Due to the space limitation, we report the results for $b = 32$ in Table VI. The distance function is Hamming Distance (HD). However, the circuit can be instantiated for any value of b. As can be seen, `MPCircuits` customized libraries result in 3-3.2× performance improvement compared to standard utilization of logic synthesis tools. Songhori et al. [24] propose a solution based on Garbled Circuits [2]. However, their approach is limited to the two-party setting only. Similarly, Qi et al. [32] create a scheme based on Homomorphic encryption for two-party settings. Perhaps the most similar work to ours is [33] where they support a multi-party setting. Nevertheless, they have not implemented their scheme.

TABLE VI: Secure k-nearest neighbor search.

		Non-optimized		**Optimized**						
k_n	n	#XOR	#AND	#XOR	#AND	OT (s)	T_{GE} (s)	T (s)	$Comm$ (MB)	Mem (MB)
1	8	7.64E+02	1.77E+03	1.79E+03	5.56E+02	1.61	1.77	7.40	0.53	10.75
	16	1.50E+03	3.68E+03	3.73E+03	1.16E+03	3.26	3.76	16.07	2.39	38.27
2	8	8.66E+02	3.31E+03	2.73E+03	1.08E+03	3.26	1.78	7.40	1.04	16.06
	16	1.67E+03	7.25E+03	5.82E+03	2.37E+03	3.48	1.79	16.32	4.88	66.54
3	8	9.77E+02	4.64E+03	3.62E+03	1.52E+03	1.59	1.70	9.26	1.46	20.42
	16	1.85E+03	1.06E+04	8.20E+03	3.50E+03	3.52	4.01	14.85	7.21	98.21

XI. Conclusion

We present MPCircuits, the first automated methodology to generate optimized Boolean circuits for secure multi-party computation (MPC). The Boolean circuit generation is a key step to employing the MPC protocols. We leverage industrial logic synthesis tools and transform the problem of generating optimized circuits for MPC to a logic synthesis problem. Our solution is modular and generic and can be adopted by different MPC protocols and implementations. To illustrate the practicality of our approach, we design and implement Boolean circuits for five compelling tasks in MPC. Namely, we consider auction, voting, private set intersection, stable matching, and nearest neighbor search. We perform extensive experimental evaluation of all five benchmarks based on the Beaver-Micali-Rogaway (BMR) protocol and show that MPCircuits automatically generates optimized circuits that require up to $4.2\times$ less garbled gates.

Acknowledgment

We would like to thank our reviewers for their valuable comments. This work is supported in part by NSF/SRC (1619261/2016-TS-2690), ONR N00014-17-1-2500, and MURI (FA9550-14-1-0351) grants to ACES lab at UCSD.

References

[1] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 218–229.
[2] A. C.-C. Yao, "How to generate and exchange secrets," in *Annual Symposium on Foundations of Computer Science*. IEEE, 1986.
[3] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 1990, pp. 503–513.
[4] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor, "Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 112–127.
[5] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "Oblivm: A programming framework for secure computation," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 359–376.
[6] S. Zahur and D. Evans, "Obliv-C: A language for extensible data-oblivious computation." *IACR ePrint Archive*, vol. 2015, p. 1153, 2015.
[7] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *S&P*. IEEE, 2013.
[8] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "TinyGarble: Highly compressed and scalable sequential garbled circuits," in *IEEE S & P*, 2015.
[9] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of Asia Conference on Computer and Communications Security (AsiaCCS)*. ACM, 2018.
[10] M. S. Riazi, E. M. Songhori, and F. Koushanfar, "PriSearch: Efficient search on private data," in *Design Automation Conference*. IEEE, 2017.
[11] M. S. Riazi, N. K. Dantu, L. V. Gattu, and F. Koushanfar, "GenMatch: Secure DNA compatibility testing," in *IEEE Hardware Oriented Security and Trust (HOST)*, 2016.
[12] A. Ben-Efraim, Y. Lindell, and E. Omri, "Optimizing semi-honest secure multiparty computation for the internet," in *CCS*. ACM, 2016.
[13] S. G. Choi, K.-W. Hwang, J. Katz, T. Malkin, and D. Rubenstein, "Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces," in *CryptographersâĂŹ Track at the RSA Conference*. Springer, 2012, pp. 416–432.
[14] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: a system for secure multi-party computation," in *ACM conference on Computer and communications security (CCS)*. ACM, 2008.
[15] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Automata, Languages and Programming*. Springer, 2008.
[16] P. Bogetoft, I. Damgård, T. P. Jakobsen, K. Nielsen, J. Pagter, and T. Toft, "A practical implementation of secure auctions based on multiparty integer computation," in *Financial Cryptography*. Springer, 2006.
[17] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a secure voting system," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 354–368.
[18] SCAPI, "the Secure Computation API," https://github.com/cryptobiu/libscapi, 2017.
[19] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.
[20] A. Waksman, "A permutation network," *Journal of the ACM (JACM)*, vol. 15, no. 1, pp. 159–163, 1968.
[21] M. S. Riazi, E. M. Songhori, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Toward practical secure stable matching," *Proceedings on Privacy Enhancing Technologies*, 2017.
[22] J. Doerner, D. Evans *et al.*, "Secure stable matching at scale," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1602–1613.
[23] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, 1962.
[24] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar, "Compacting privacy-preserving k-nearest neighbor search using logic synthesis," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 36.
[25] Z. Huang, "Privacy preserving auction." 2016.
[26] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004.
[27] M. Larson, C. Hu, R. Li, W. Li, and X. Cheng, "Secure auctions without an auctioneer via verifiable secret sharing," in *Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing*. ACM, 2015, pp. 1–6.
[28] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992, pp. 244–251.
[29] E. De Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity." in *Financial Cryptography*, vol. 10. Springer, 2010, pp. 143–159.
[30] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing." in *USENIX Security Symposium*, 2015, pp. 515–530.
[31] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical multi-party private set intersection from symmetric-key techniques," in *CCS*. ACM, 2017.
[32] Y. Qi and M. J. Atallah, "Efficient privacy-preserving k-nearest neighbor search," in *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*. IEEE, 2008, pp. 311–319.
[33] M. Shaneck, Y. Kim, and V. Kumar, "Privacy preserving nearest neighbor search," in *Machine Learning in Cyber Trust*. Springer, 2009.