SynCirc: Efficient Synthesis of Depth-Optimized Circuits for Secure Computation (Full Version)*

Arpita Patra[1], Thomas Schneider[2], Ajith Suresh[1], Hossein Yalame[2]
[1] Indian Institute of Science, Bangalore, India
{arpita, ajith}@iisc.ac.in
[2] TU Darmstadt, Germany
{schneider, yalame}@encrypto.cs.tu-darmstadt.de

***Abstract*—Secure Multi-party Computation (MPC) allows to securely compute on private data. To make MPC practical, logic synthesis can be used to automatically translate a description of the function to be computed securely into optimized and error-free boolean circuits. TinyGMW (Demmler et al., CCS'15) used industry-grade hardware synthesis tools (DC, Yosys) to generate depth-optimized circuits for MPC. To evaluate their optimized circuits, they used the ABY framework (Demmler et al., NDSS'15) for secure two-party computation. The recent ABY2.0 framework (Patra et al., USENIX Security'21) presented round-efficient constructions using multi-input AND gates and improved over ABY by at least $6\times$ in online communication for 4-input AND gate evaluation.**

In this work, we propose *SynCirc*, an efficient hardware synthesis framework designed for MPC applications. Our framework is based on Verilog and the open-source tool Yosys-ABC. It provides custom libraries and new constraints that accommodate multi-input AND gates. With this, we improve over TinyGMW by up to $3\times$ in multiplicative depth with a corresponding improvement in online round complexity. Moreover, we provide efficient realizations of several new building blocks including comparison, multiplexers, and equality check. For these building blocks, we achieve improvements in multiplicative depth/online rounds between 22.3% and 66.7%. With these improvements, our framework makes multi-round MPC better-suited for high-latency networks such as the Internet. With respect to the look-up table based approach of Dessouky et al. (NDSS'17), our framework improves the online communication by $1.3\times - 18\times$.

***Index Terms*—Secure Function Evaluation, Hardware Synthesis, Multi-party Computation, Depth Optimization, Logic Design, ABY2.0**

I. Introduction

Secure multi-party computation (MPC) [2], [3], one of the prominent techniques of modern cryptography, allows a set of n mutually distrusting parties to jointly compute a function on their private inputs. In addition to the correctness of the function output, MPC guarantees privacy which ensures that no set of t corrupt parties can learn more information than the output. MPC has showcased its potential in real-world applications such as private auctions [4], secure aggregation [5], [6], private clustering [7], [8], and recently in the domain of privacy-preserving machine learning (PPML) [9]–[17].

The area of MPC research was kickstarted mainly by two protocols, both evaluating a Boolean circuit representing the desired functionality – i) Yao's garbled circuits (GC) [2] approach and ii) the Goldreich-Micali-Wigderson (GMW) [3] paradigm. The former has a constant number of communication rounds, yet incurs a high communication. The latter has a better communication cost, yet incurs a depth-proportional round complexity. Since then, enormous literature has been exploring the design of practically efficient circuit-based secure computation protocols.

While experts manually designed circuits for each of the specific functions in the initial days [18]–[23], the task became tedious with the introduction of large and complex functions. Moreover, errors in such hand-made circuits can even lead to a breach of privacy by leaking additional information on the party's private inputs. This paved the way for automated generation of circuits [18], [19], [21], [23]–[32] wherein high-level function descriptions are automatically compiled to efficient circuit representations using logic synthesis. The optimization criteria within the logic synthesis are derived based on the protocol that uses the circuit. For instance, Yao-based protocols prefer to have circuits with as less as possible AND gates (and as high as possible XOR gates due to free-XOR technique that allows XOR gates to be evaluated locally [33]), whereas the GMW-style approach prefers circuits with optimized multiplicative depth. The most widely accepted methods for logic synthesis include Circuit Compilers (CC) and Hardware Synthesis Tools (HST).

In the Circuit Compilers approach, Fairplay [18], [19] and PAL [30] compiled a domain-specific language (DSL) into a size-optimized Boolean circuit. The CBMC-GC compiler [21] used an SAT model checker to generate size-optimized Boolean circuits from the ANSI C language. The PCF compiler [23] generates a compact assembler-like intermediate representation while the KSS compiler [20] generates Boolean circuits from a DSL description. ShallowCC [31] takes ANSI C as input and creates depth-optimized Boolean circuits by introducing new optimized building blocks and proposing multiple depth minimization techniques. In [27], [28], a logic synthesis toolbox was proposed for security applications. They consist of resubstitution, refactoring, and rewriting algorithms and consider the minimization of the number of AND gates as their primary goal.

In the Hardware Synthesis Tools approach, TinyGarble [24] proposed a method using sequential circuits and already estab-

*Please cite the version of this paper published at 14. IEEE International Symposium on Hardware Oriented Security and Trust (HOST'21) [1].

lished powerful hardware logic synthesis tools to synthesize size-optimized circuit descriptions. TinyGMW [25], in contrast to TinyGarble, focused on automatically synthesizing low-depth combinational circuits for use in protocols that require a low multiplicative depth. Dessouky et al. [26] went beyond Boolean gates and replaced the 2-input Boolean gates representation by more compact lookup tables (LUTs) and utilized FPGA LUT-based synthesis tools to transform a description in a hardware description language (HDL) into a LUT representation for LUT-based cryptographic protocols. MPCircuits [29] generates size-optimized Boolean circuits using hardware synthesis tools and designing new technology libraries for security applications with multiple parties. Recently, Heldmann et al. [34] proposed an automated circuit compilation suite based on the compiler toolchain LLVM. Their LLVM output is further processed with the ABC tool [35] to produce size-optimized circuits.

In this work, we improve upon the approach of TinyGMW [25] where industry-grade hardware synthesis tools were tweaked for logic synthesis. TinyGMW focused on depth-optimized circuits for the GMW paradigm owing primarily to the following reasons – i) it allows to pre-compute the communication-intensive input independent operations in a setup phase offering a high-speed online phase, and ii) GMW supports better parallelization of the same circuit using SIMD operations leading to high throughput [22], [36]. Moreover, the circuits generated by their toolchain were compatible with the ABY framework [36], which was the best known MPC framework for two parties (2PC) back in 2015. Recently, the ABY2.0 protocols [37] improved upon those in ABY by a factor $6\times$ in online communication for 4-input AND gate and introduced MPC protocols to evaluate multi-input AND gates efficiently. In this work, we use HW synthesis tools to accommodate for multi-input AND gates to obtain circuits with a better multiplicative depth that can then be evaluated using the ABY2.0 protocols [37]. Moreover, we implement efficient circuits for several building blocks with a focus on secure computation as the end goal.

A. Outline and Our Contributions

In this work, we present **SynCirc**, an efficient framework for synthesizing boolean circuits with the end goal of being deployed in a secure computation framework. For this, we re-purpose logic synthesis tools [24]–[26], [29], to automatically and efficiently compile a function written in a Hardware Description Language into a multiplicative depth-optimized representation. The generated circuit comprises basic boolean gates and multi-input AND gates with a fan-in of up to four. SynCirc is the first hardware synthesis framework for MPC to accommodate multi-input AND gates.

Our SynCirc framework, built along the lines of TinyGMW [25], is designed to minimize the communication and rounds in the online phase. The framework consists of several building blocks and improves the multiplicative depth over previous circuit building blocks between 22.3% and 66.7%. Thus, SynCirc clubbed with the state-of-the-art 2PC protocols of ABY2.0 [37] makes MPC better suited for high-latency networks such as the Internet. Our contributions are summarized next.

a) Our three-layer framework: Our framework, depicted in Fig. 1 consists of three layers, with the 3rd and final layer consisting of the privacy-preserving realization of various functionalities and forming the end goal.

Layer I consists of basic boolean gates such as AND, XOR, and NOT. In addition, we incorporate 3 and 4 input AND gates (AND3 and AND4) at this layer by re-engineering the synthesis tool with proper parameters. Layer I forms the basis for Layer II, which comprises the depth-optimized variants of basic functionalities. This includes ℓ-bit boolean adder/subtractor, bit extraction, multiplexer, equality test, etc., and forms the building blocks for many secure computation tasks. To accommodate multi-input AND gates during synthesis, we developed a customized synthetic library in Verilog with the depth-optimized circuit descriptions of the functionalities in Layer II. With the proper mapping of functionalities, the synthesis tool uses our custom descriptions over the built-in ones. We can assemble the building blocks in Layer II to construct several advanced functionalities, which would otherwise be impractical to do by hand. In Layer III, we present a list of such functionalities: division, sorting, private set intersection, and privacy-preserving machine learning blocks such as Rectified Linear Unit (ReLU) and Argmax.

Though we consider only a limited set of functionalities in this work, we can leverage the modular design of our framework to enhance the support for more advanced functionalities later. For instance, as shown in MPCircuits [29], a more advanced Layer IV for functions like auctions, voting, stable matching, and nearest neighbour search can be constructed using existing layers, and we leave this as future work.

TABLE I
COMPARISON OF OUR CONSTRUCTIONS WITH SHALLOWCC [31] IN TERMS OF MULTIPLICATIVE DEPTH.

Operation	**ShallowCC** [31]	**SynCirc (this work)**
n-bit Addition	$\log_2(\mathsf{n})+1$	$0.5\ \log_2(\mathsf{n})+1$
n-bit Subtraction	$\log_2(\mathsf{n})+2$	$0.5\ \log_2(\mathsf{n})+2$
n-bit Multiplication	$2\ \log_2(\mathsf{n})+3$	$1.5\ \log_2(\mathsf{n})+2$
n-bit Comparison	$\log_2(\mathsf{n})+1$	$0.5\ \log_2(\mathsf{n})$+1
n : 1 Multiplexer	$\log_2(\log_2(\mathsf{n})+1)$	$0.25\ \log_2(\mathsf{n})$

b) Comparison with Circuits Compiler approach: In Table I, we compare the multiplicative depth of the building blocks of SynCirc with the variants proposed in ShallowCC [31], the state-of-the-art circuit compiler for depth optimized boolean circuits. Concretely, for 64-bit inputs, our constructions improve the multiplicative depth of ShallowCC by $1.25\times$ - $2.0\times$. Note that reducing the depth of a block by just one can impact the overall performance by a large amount in several applications. The comparison operation is one such example that contributes to more than 93% of the rounds for neural network training/inference [17].

c) Implementation and Benchmarking: We implemented the functionalities in Verilog and synthesized the netlists using

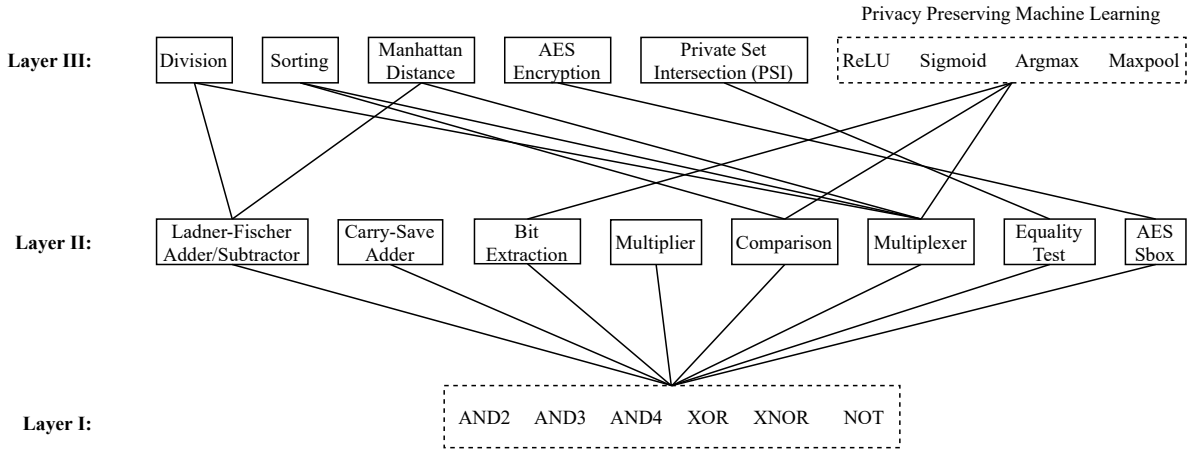

Fig. 1. The three-layer SynCirc architecture

the Yosys-ABC tool [38], [39]. As XOR gates are free in MPC protocols, we primarily considered the multiplicative depth and the number of non-free gates (AND2, AND3, AND4) as the parameters for benchmarking. We report the values over different bit-widths ranging from 8 to 64. For obtaining the secure variant of the functionalities, we evaluated the circuits generated by our synthesis tool using the state-of-the-art two-party protocols of ABY2.0 [37]. As evident from Table VI in §V-B, our improvement in multiplicative depth ranges from $1.3\times$ to $3\times$ over the various functionalities considered in this work. Note that our framework is independent of the secure protocol being executed and can be used in any MPC framework that supports multi-input AND gates.

d) Improvement over Lookup Table (LUT) Based Approaches: The lookup table-based approach of Dessouky et al. [26] is an orthogonal line of work, though it has an end goal similar to ours. The approach keeps a balance between the communication and rounds by trading off the computation. They propose two variants with different trade-offs: i) OP-LUT, which optimizes online communication, and ii) SP-LUT, which optimizes setup/total communication. When compared with the OP-LUT variant of [26], our circuits evaluated using the ABY2.0 [37] protocols improve the online communication in the range $1.3\times$ - $1.6\times$ keeping the multiplicative depth on par with the constructs in [26] (cf. Table VIII) for most of the cases. Moreover, we improve the overall communication of the circuits mentioned earlier in the range $70\times$ - $208\times$ while simultaneously eliminating the need for heavy computations required in [26]. Hence, for practical applications where a balance between computation and communication is needed, our framework improves over [26]. More details are provided in §V-C.

II. Preliminaries and Background

A. Secure computation over Boolean circuits

In this work, we focus on evaluating Boolean circuits using the GMW paradigm. As pointed out in [22] and TinyGMW [25], the GMW paradigm has several advantages over the constant round approach of Yao. Apart from reducing communication, the GMW approach allows to balance the workload among the parties, to provision to parallelize evaluations of the same circuit using SIMD operations [22], [36], and to precompute all symmetric cryptographic operations in a preprocessing phase leading to a very efficient online phase. Another approach that improves over GMW is the Lookup Table (LUT) method of [26]. It further reduces the round complexity over GMW and yields communication better than Yao. However, this improvement comes at the expense of an increase in computation. Hence, the LUT-based approach can be considered as a middle ground between Yao and GMW-based solutions.

We use the ABY2.0 [37] protocols for the secure evaluation of the circuits generated by our framework. ABY2.0 improved the online communication of its predecessor ABY [36] using an input-independent preprocessing, which now depends on the circuit and a new secret sharing scheme. Moreover, ABY2.0 allows to improve the online round complexity of ABY by incorporating protocols for multi-input AND gates (AND gates with three and four inputs, to be specific). ABY2.0 is secure against semi-honest corruption, preventing data leakage to a passively corrupted party.

B. HDL Synthesis Tools for Secure Computation

Generating a hand-optimized error-free Boolean circuit for secure computation is a tedious and time-consuming task. The problem becomes more challenging with the recent advancements in the area where multi-input Boolean gates are taken into account in addition to the standard two-input ones [37], [40]. Instead of 'reinventing the wheel' and building a new compiler [18], [19], [21], [23], [27]–[31], [41], works like [24]–[26], [29] showcased the potential of re-engineering logic synthesis tools to accomplish this task. A logic synthesis tool takes a function description, written in a hardware description language (HDL) as input and transforms this function into a suitable output for the standard target technologies. For instance, the target can be either Lookup Tables (LUTs) for Field Programmable Gate Arrays (FPGAs) or Boolean gates for Application-Specific Integrated Circuits (ASICs).

In SynCirc, we use the open-source Yosys-ABC tool [38], [39] for ASIC synthesis following TinyGMW [25]. SynCirc integrates well with the regular ASIC design flow. This is because the proposed solution instructs the ASIC synthesis to use our customized circuit descriptions instead of the standard cells, and the rest of the workflow is untouched. For this reason, it is possible to use tools like the commercial Design Compiler (DC) by Synopsis [42] and we leave the industry-grade realization of SynCirc using commercial synthesis tools as future work.

III. Global Flow Overview

A. Challenges of Logic Synthesis for MPC protocols

The generation of circuits with hardware synthesis for secure computation has two main challenges. First, hardware synthesis tools target hardware platforms like FPGAs and ASICs and involve technology constraints different from the ones needed for Boolean circuits. Moreover, these tools use the layout as a synthesis parameter, whereas Boolean circuits for secure computation do not have such layout constraints. The generated circuits are evaluated 'virtually' using a secure protocol and not through a physical evaluation. Second, the cost of a gate varies significantly from the viewpoint of a hardware synthesis and secure computation. For example, XOR gates are essentially free in MPC protocols as they can be evaluated locally. In contrast, in the logic synthesis tools, Boolean NAND gates are favored over XOR gates because of their placement costs. For this reason, the logic synthesis tools need to be repurposed to achieve our objectives for security applications, especially in generating multiplicative depth-optimized Boolean circuits for ABY2.0.

B. Customizing Synthesis

Fig. 2 depicts the high-level flow of our SynCirc synthesis framework which is similar to the toolchain in TinyGMW [25]. Given the user input in the form of a hardware description language, the goal of the synthesis tool is to generate its logical representation in the form of a Boolean circuit that best fits the user's constraints. For the secure evaluation, we feed the Boolean circuit generated as an input to the ABY2.0 protocol. The first step is synthesizing a set of basic arithmetic and logic operations such as addition, subtraction, multiplication, division, comparator, and multiplexers. Each of these operations can be instantiated from different implementations, depending on the provided constraints in the synthesis tool. For instance, we can obtain different circuits for an l-bit comparison by optimizing for area [32] or delay [43]. The basic operations can be used as building blocks for advanced operations such as sorting and private set intersection. In the hardware synthesis tool, the arithmetic and logic operations are mapped to standard cells given in a cell library by default. Moreover, standard libraries [44], [45] or libraries proposed by TinyGarble [24], TinyGMW [25], and MPCircuit [29] cannot be used directly in our framework. This is mainly because of two reasons - i) the cost metric in the ABY2.0 protocol is the multiplicative depth instead of online communication in the previous works, and ii) ABY2.0 provides multi-input AND gates (AND3, AND4) in the protocol that are absent in previous libraries.

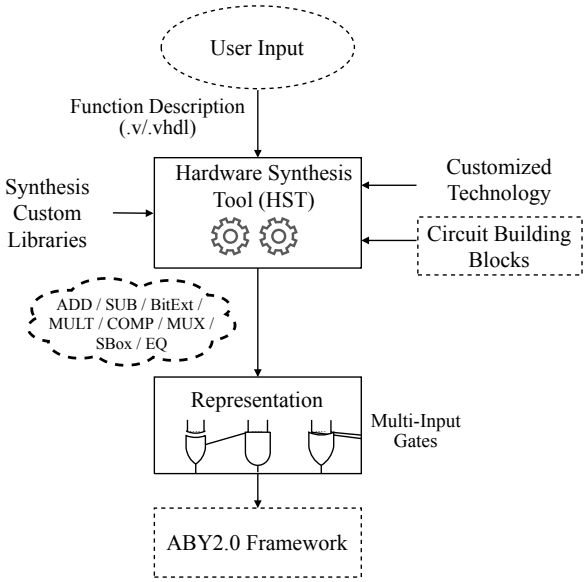

Fig. 2. Global Flow of SynCirc framework

Since the goal is to reduce the multiplicative depth of the circuit, we have implemented a special synthesis technology library that includes the circuits minimized for multiplicative depth of the basic arithmetic and logic operations considering XOR, AND2, AND3, and AND4 gates. We added our multiplicative depth-optimized blocks to the library of the hardware synthesis tool [38], [39]. We then re-engineered the toolchain to enable automated mapping to our customized circuit descriptions rather than the standard cells.

The technology library consists of the functional description, the Boolean function they represent, and their parameters like the delay and area in a semiconductor vendor's library. For this, we use an ASIC cell library in liberty format [44, Fig. 3] that specifies the cells that can be used along with their cost functions. Our customized technology library has no manufacturing or technology rules, similar to the approach in TinyGarble [24] and TinyGMW [25]. The cells in our library, which are used in the ASIC mapping for the ABY2.0 protocol, contain Boolean logic gates AND2, AND3, AND4, NOT, XOR, and XNOR. The cost function of these gates is manipulated as follows: the cost parameters of XOR, XNOR, and NOT gates are set to 0, while those for AND2, AND3, and AND4 gates are set to reasonably high values. Note that setting the cost of a Boolean gate to very high values will result in the gate getting excluded from the synthesis. Thus, to exclude OR gates from the synthesis, their costs are set to very high values.

Yosys-ABC [38], [39], the open-source toolchain used in our work, uses gates from the provided technology library to generate an implementation of the design at a gate level. We perform a multi-level combinational logic optimization on gate-level netlists using the external Berkeley ABC [35] tool integrated with the Yosys [44] toolchain. The "abc" pass over

Yosys extracts the combinational gate-level parts of the design, passes it through ABC and re-integrates the results. Yosys-ABC is configured to minimize the multiplicative depth considering up to 4-input AND gates. As a result, the synthesis process minimizes the multiplicative depth in the final netlist, and these settings give the most desirable mapping results. This way, we gain up to $3\times$ improvement in the multiplicative depth for our benchmark circuits compared to TinyGMW [25] thanks to our multi-input AND gates.

We need to ensure that the toolchain uses our multiplicative-depth optimized circuits from the customized library that we added for realizing the advanced functionalities. For this, we add the customized library's path in the toolchain and we instruct the tool to optimize depth in the synthesis script.

IV. Building Blocks Library

In this section, we provide high-level details of the depth-optimized circuits obtained using our toolchain. We classify the circuits into two types: i) Basic – that form the building blocks for most of the secure computation tasks, and ii) Advanced – that use the basic circuits to build circuits for complex functionalities. All of the below circuits outperform the state-of-the-art circuits in multiplicative-depth. To verify the correctness, test benches are used to simulate our building blocks without any physical hardware which is not needed in MPC protocols.

A. Layer II - Basic Functionalities

Customized Ladner-Fischer Adder/Subtractor ($\mathsf{ADD_{CLF}}$/$\mathsf{SUB_{CLF}}$). To perform the addition of two ℓ-bit values, the traditional Ripple Carry Adder (RCA), in which the carry out of one stage is fed directly to the carry-in of the next stage, yields a multiplicative depth of ℓ [46], [47]. To improve the speed, Carry Look Ahead (CLA) adders were introduced, where the carry bits are computed in advance, and thus the depth is reduced. The Ladner-Fischer Adder [22] is one among the widely used CLA and has a multiplicative depth of $2\lceil\log_2(\ell)\rceil + 1$. ShallowCC [31] proposed a construction with a depth of $\lceil\log_2(\ell)\rceil + 1$ for another CLA named Sklansky Adder [48]. ABY2.0 [37] used Parallel-Prefix Adders (PPA) using up to 4-input AND gates to build a hand-optimized adder with ℓ-bit inputs and ℓ-bit output with multiplicative depth $\lfloor\log_4(\ell)\rfloor$ for input sizes $\ell \in \{8, 64\}$. For $(\ell+1)$-bit outputs, their construction has multiplicative depth $\lfloor\log_4(\ell)\rfloor + 1$ which matches the depth of our automatically generated addition circuit in SynCirc.

Fig. 3 shows a depth-optimized 8-bit PPA architecture obtained by the combined use of AND2, AND3, and AND4 gates.

TABLE II
Multiplicative depth of Adder Circuits for bitwidth ℓ.

Work	ℓ	**8**	**16**	**32**	**64**
Ripple-Carry [47]	$\ell - 1$	7	15	31	63
Ladner-Fischer [22], [49]	$2\lceil\log_2(\ell)\rceil + 1$	7	9	11	13
Sklansky [31]	$\lceil\log_2(\ell)\rceil + 1$	4	5	6	7
ABY2.0 [37]	$\lfloor\log_4(\ell)\rfloor + 1$	**2**	–	–	**4**
SynCirc	$\lfloor\log_4(\ell)\rfloor + 1$	**2**	**3**	**3**	**4**
% reduction in depth		40%	40%	50.0%	42.8%

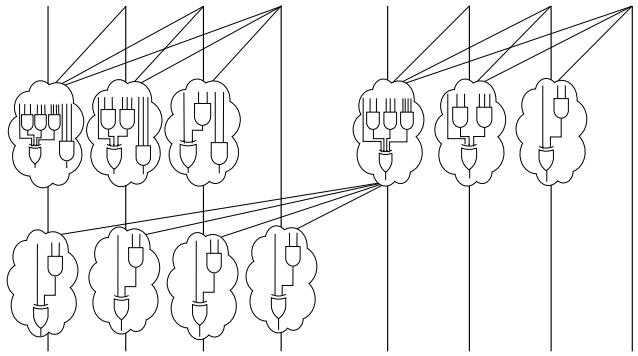

Fig. 3. Parallel Prefix Adder Architecture for 8-bit Customized Ladner-Fischer Adder ($\mathsf{ADD_{CLF}}$) [37].

The customized Ladner-Fischer Adder produced by our toolchain has a depth of $\lfloor\log_4(\ell)\rfloor + 1$ and works for any value of ℓ. This amounts to a reduction of $\approx 42\%$ in multiplicative depth when compared with the ShallowCC compiler [31] for the case of 64-bit inputs. Table II provides a comparison of our circuit with other circuits in the literature. A subtractor is a special case of adder as the subtraction of two values a and b and can be represented as $a + \bar{b} + 1$ where $\bar{b}$ denotes the negated binary representation of b. As a result, we achieve similar improvements for subtraction.

Carry-Save Adder ($\mathsf{ADD_{CSA}}$). To perform addition of $\mathsf{n} > 2$ values, the sequential method of adding two values at a time will require $\mathsf{n}-1$ sequential additions and hence a multiplicative depth of $(\mathsf{n}-1)\mathsf{d_{ADD}}$, where $\mathsf{d_{ADD}}$ denotes the depth of the adder circuit used. However, a tree-based approach can reduce the depth to $\lceil\log_2(\mathsf{n})\rceil \cdot \mathsf{d_{ADD}}$. For adding three ℓ-bit values, both of the aforementioned approaches require a multiplicative depth of $2\mathsf{d_{ADD}}$. A Carry-Save Adder (CSA) on the other hand provides efficient constructions for multiple successive additions with better multiplicative depth. For instance, the CSA used in ShallowCC [31] has a multiplicative-depth of $\lceil\log_2(\ell)\rceil + 2$ for adding three ℓ-bit values. Moreover, they proposed efficient constructions for a Carry-Save Network (CSN) that evaluates n sequential additions with a depth of $\lceil\log_2(\mathsf{n})\rceil + \lceil\log_2(\ell)\rceil$. Using our toolchain, we obtain a CSA ($\mathsf{ADD_{CSA}}$) with a depth of $\lfloor\log_4(\ell)\rfloor + 2$ and a CSN ($\mathsf{ADD_{CSN}}$) with a depth of $\lceil\log_2(\mathsf{n})\rceil + \lfloor\log_4(\ell)\rfloor$. Concretely, for a bit-width of 64 and $\mathsf{n} = 64$, we improve the multiplicative depth of $\mathsf{ADD_{CSA}}$ by $1.6\times$ and $\mathsf{ADD_{CSN}}$ by $1.3\times$ over ShallowCC.

Bit Extraction (BitExt). The bit extraction circuit computes the Most Significant Bit (MSB) of the sum of two ℓ-bit values [16], [37]. For this, we can tweak an ℓ-bit adder by removing all the unnecessary gates. The transition from adder to bit extraction circuit is easy in our framework. This is due to the hardware synthesis tool automatically removing all those gates whose outputs are neither assigned directly to the output nor used later as inputs to other gates. Our bit extraction circuit follows our adder circuit and has a multiplicative depth of $\lceil\log_4(\ell)\rceil + 1$ which matches with the hand-optimized circuit in ABY2.0 [37]. Fig. 4 shows the structure of BitExt circuit with 8-bit inputs.

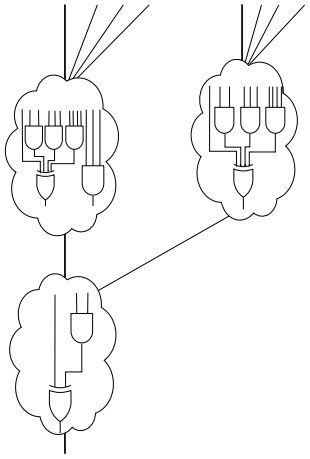

Fig. 4. BitExt PPA architecture for 8-bit inputs [37].

Multiplier (MUL$_{\mathsf{CLF}}$). A multiplier computes the 2ℓ-bit product of two ℓ-bit inputs. The standard textbook method uses bitwise multiplication followed by shifted addition resulting in a depth of $2\ell - 1$ [22]. In this approach, a total of ℓ partial products of length ℓ-bit are computed and then added. However, we can achieve a faster addition of these partial products using a tree structure and ADD$_{\mathsf{CLF}}$.

TABLE III
MULTIPLICATIVE DEPTH OF MULTIPLIER CIRCUITS FOR BITWIDTH ℓ.

Work	ℓ	16	32	64
Textbook [47]	$2\ell-1$	45	93	189
MulCSN [22]	$3\lceil\log_2(\ell)\rceil+4$	16	19	22
ShallowCC [31]	$2\lceil\log_2(\ell)\rceil+3$	11	13	15
SynCirc	$\lceil\log_2(\ell)\rceil+\lfloor\log_4(2\ell-1)\rfloor+2$	**8**	**10**	**12**
% reduction in depth		28.0%	23.0%	20.0%

Fig. 5 shows a Wallace-tree based multiplier which consists of three main steps - i) computing ℓ partial products with depth 1, ii) aggregating partial products in a tree structure with depth $\lceil\log_2(\ell)\rceil$, and ii) adding two $2\ell - 1$-bit values using ADD$_{\mathsf{CLF}}$. Thus, a multiplier with a depth of $\lceil\log_2(\ell)\rceil+\lfloor\log_4(2\ell-1)\rfloor+2$ can be designed by combining all the three steps. We provide a comparison of our circuit with existing works in Table III. Our circuit reduces the depth of existing solutions by at least 20.0% for any bit-width.

Comparison (COMP). To compare two ℓ-bit values x and y ($x > y$), the standard approach [32] demands a depth ℓ while the recursive approach [43] requires only a depth of $\lceil\log_2(\ell)\rceil+1$. Using multi-input ANDs, our COMP circuit reduces the depth to $\lfloor\log_4(\ell)\rfloor+1$. The circuit for 8-bit values is given in Fig. 6. Table IV provides a comparison of our construction with existing works, and we observe a minimum improvement of 40% in multiplicative depth for any bit-width over existing solutions.

Multiplexer (MUX). A multiplexer (MUX) is the most important building block for the control and data flow [33]. MUXes are used to evaluate conditionals and array accesses. [33] provided a construction of a 2-to-1 MUX that only requires one AND2 gate for every pair of input bits and has a depth of 1. The tree architecture for 8-to-1 MUX has a depth of 3 [25]. Our toolchain could generate circuits of depth 1 for 2-to-1 MUX, 4-to-1 MUX, and 8-to-1 MUX using multi-input AND gates. Fig. 7 shows the structure for 4-to-1 MUX. Table V details

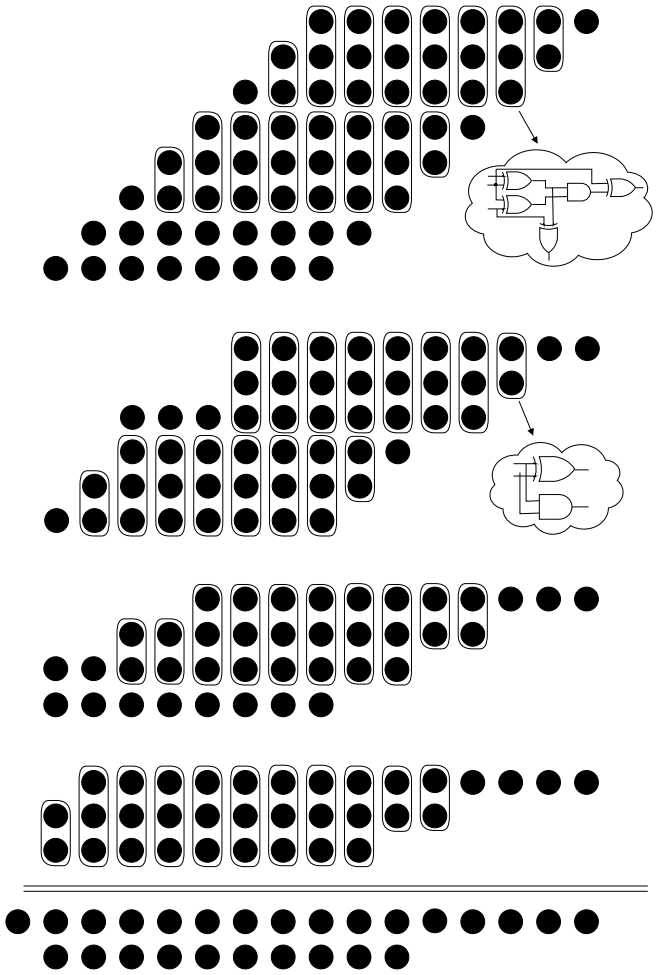

Fig. 5. Stages of an 8-bit Wallace tree multiplier [50].

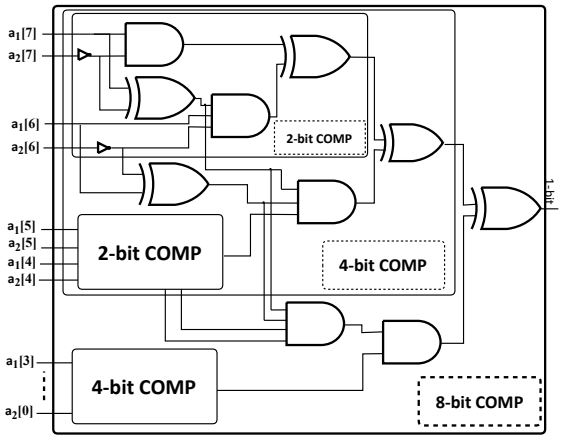

Fig. 6. Multiplicative depth-optimized 8-bit comparison circuit [43].

TABLE IV
MULTIPLICATIVE DEPTH OF COMPARISON CIRCUITS FOR BITWIDTH ℓ.

Work	ℓ	16	32	64
Sequential GT [47]	ℓ	16	32	64
Recursive GT [43]	$\lceil \log_2(\ell) \rceil$+1	5	6	7
SynCirc	$\lfloor \log_4(\ell) \rfloor$+1	**3**	**3**	**4**
% reduction in depth		40.0%	50.0%	43.0%

the multiplicative depth for different numbers of inputs (n). Concretely, for an 8-to-1 MUX, this amounts to a reduction of 50% in multiplicative depth over ShallowCC [31].

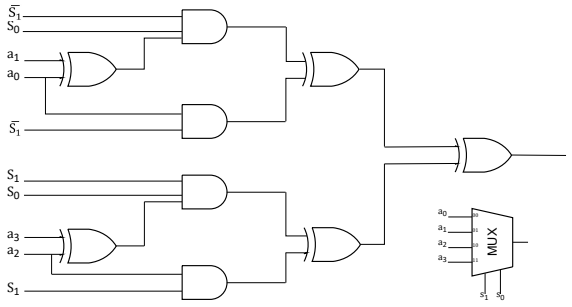

Fig. 7. Multiplicative depth-optimized 4-to-1 multiplexer (MUX).

Equality Test (EQ). The equality test circuit is a very common building block used in circuit-based private set intersection (PSI) [51]–[55] and Data Mining [56]. As shown in Fig. 8, this circuit can be built from XNOR and AND gates [22], [32]. For ℓ-bit inputs, we need ℓ XNOR and a tree of $(\ell - 1)$ AND2 gates of depth $\lceil \log_2(\ell) \rceil$. The toolchain simply replaces three AND2 gates with an AND4 gate, which improves the round complexity over [51] by $2\times$ from $\log_2(\ell)$ to $\log_4(\ell)$.

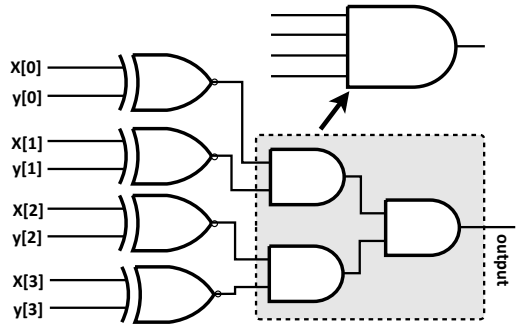

Fig. 8. 4-bit Equality check circuit [32].

TABLE V
MULTIPLICATIVE DEPTH OF MULTIPLEXER CIRCUITS FOR BITWIDTH n.

Work	n	8	16	32
MUX-Tree [22]	$\lceil \log_2(n) \rceil$	3	4	5
MUX-DNFs [31]	$\lceil \log_2(\lceil \log_2(n) \rceil) \rceil + 1$	3	4	4
MUX-DNFd [31]	$\lceil \log_2(\lceil \log_2(n) + 1 \rceil) \rceil$	2	3	3
SynCirc	$\lceil \log_8(n) \rceil$	**1**	**2**	**2**
% reduction in depth		50.0%	33.0%	33.0%

Sbox (Sbox). Enabling a party to encrypt the message x using a key k held by the other party is the goal of privacy-preserving AES [57], which has several applications in private set intersection [58] and encrypted databases [59], [60]. Since AES operations like MixColumns and AddRoundKey can be evaluated using only free XOR gates [61], the focus is to building efficient circuits for its core block Sbox, which has 8 input bits and 8 output bits. While [62] have used a special Greedy-approach to identify a small Boolean circuit with 34 AND2 gates and multiplicative depth 4, using our toolchain, we obtain an Sbox with a multiplicative depth of 3, matching the depth of the hand-optimized Sbox circuit proposed in ABY2.0 [37]. This is achieved by replacing some of the AND2 gates in the optimized Sbox constructions of [62], [63] with AND3 gates. The generated circuit has 30 AND2 and 4 AND3 gates.

B. Layer III - Advanced Functionalities

Division (DIV). Integer DIV, which computes the quotient for a division of two binary integer numbers, is a complex operation that is not trivially implementable by hand [64]. This fundamental operation finds application in several ML algorithms such as the softmax function [65] and k-means clustering [66]. The standard approach named "long division" works similar to the textbook multiplication. We can build an ℓ-bit DIV using ℓ subtractors and ℓ multiplexers, each of bit-width ℓ [67]. We synthesised DIV circuits in our framework for bit-widths $\{\ell = 16, \ell = 32\}$ using $\mathsf{SUB_{CLF}}$ and MUX. Listing 1 shows the details for a 16-bit division based on the circuit in TinyGarble [24].

```
module Division16(x,y,o);
input [15:0] x,y;
output[15:0] o;
wire [31:0] temp1[16:0];
wire [31:0] temp2[15:0];
assign  temp1[16] = {{16{1b0}}, x};
genvar i;
generate
for(i = 15; i >= 0;i = i - 1)
    begin:MyDIV
    if (i > 0)
  SUB_CLF _SUB(.x_1(temp1[i+1]),.x_2({{(16-i){1b0
    }}, y, {g{1b0}}}),.out({o[i],temp2[i]}));
  else
  SUB_CLF _SUB(.x_1(temp1[i+1]),.x_2({{(16-i){1b0
    }}, y}),.out({o[i],temp2[i]}));
  MUX _MUX(.x_1(temp1[i+1]),.x_2(temp2[i]),.s(o[i
    ]),.out(temp[i]));
  end
endgenerate
endmodule
```

Listing 1. 16-bit Division circuit using basic functionalities in SynCirc [24]

Sorting (SORT). Sorting is one of the core building blocks for many data analysis tasks such as private set intersection [29] and k-Nearest Neighbors [68]. Our circuit SORT uses the Bitonic Sort algorithm of [69] that is implemented by [29]. The circuit consists of multiplexer (MUX) and comparison (COMP) blocks. These blocks are used for the conditional swap operation that swaps two input numbers into sorted order.

Manhattan Distance ($\mathsf{DST_M}$). The Manhattan distance ($\mathsf{DST_M}$) between two points $a = (x_1^l, y_1^l)$ and

$b = (x_2^l, y_2^l)$ is the absolute distance along a two dimensional space where only horizontal and vertical paths are allowed and can be computed as $|x_1^l - x_2^l| + |y_1^l - y_2^l|$. This distance metric is useful in several applications like private localization [68] and k-Nearest Neighbors [70]. The $\mathsf{DST_M}$ value can be computed using four Subtraction ($\mathsf{SUB_{CLF}}$), two 2-to-1 multiplexer (MUX), and one addition ($\mathsf{ADD_{CLF}}$) block as shown in Listing 2.

```
module ManhattanDistance16(x1,x2,y1,y2);
input [15:0] x1, y1, x2, y2;
output [17:0] distance;
wire [16:0] dist_x1_x2, dist_x2_x1,
dist_abs_x1x2;
wire [16:0] dist_y1_y2, dist_y2_y1,
dist_abs_y1y2;
SUB_CLF x1_x2 (.x_1(x1),.x_2(x2),.out(
dist_x1_x2));
SUB_CLF x2_x1 (.x_1(x2),.x_2(x1),.out(
dist_x2_x1));
SUB_CLF y1_y2 (.x_1(y1),.x_2(y2),.out(
dist_y1_y2));
SUB_CLF y2_y1 (.x_1(y2),.x_2(y1),.out(
dist_y2_y1));
MUX abs_x1x2 (.x_1(dist_x1_x2),.x_2(
dist_x2_x1),.s(dist_x1_x2[15]),.out(
dist_abs_x1x2));
MUX abs_y1y2 (.x_1(dist_y1_y2),.x_2(
dist_y2_y1),.s(dist_y1_y2[15]),.out(
dist_abs_y1y2));
ADD_CLF _dist(.x_1(dist_abs_x1x2),.x_2(
dist_abs_y1y2), .out(distance));
endmodule
```

Listing 2. Manhattan Distance using basic functionalities in SynCirc [70]

Private Set Intersection (PSI). Circuit-based PSI [51]–[55] allows two parties to privately compute a function on the intersection of their private input sets. This has several applications like measuring ad conversion rates and data mining. We use the Bitwise-AND implementation of [29], [51]. Each set is represented as a binary vector, and the set intersection is calculated using a bit-wise AND between the sets provided by the parties. The core building block for performing bit-wise AND is the private equality circuit (EQ). Also, the more advanced linear-complexity circuit-based PSI protocols [52]–[55] use equality circuits after mapping the sets to bins. Our toolchain uses our depth-optimized EQ circuit for PSI.

Privacy-Preserving Machine Learning (PPML). In privacy preserving machine learning (PPML) algorithms, since the underlying values are real numbers, Fixed-Point Arithmetic (FPA) semantic is used to embed real values to ℓ-bit algebraic structures combined with two's complement representation [9], [37]. Here, the least significant d bits denote the fractional part and the most significant bit (MSB) denotes the sign. An MSB of 1 denotes a negative value and a 0 denotes a positive value. We use the FPA semantic details while constructing the circuits for PPML.

*Rectified Linear Unit (*ReLU*):* ReLU is one of the most widely used non-linear activation functions in several PPML algorithms. The ReLU function on a value v is defined as $\mathsf{ReLU}(v) = \mathsf{max}(0, v)$. For this, first compute the sign of the value v, say bit b, by extracting the MSB using the BitExt circuit. Given b, $\mathsf{ReLU}(v)$ can be computed using a 2-to-1 multiplexer (MUX) circuit with inputs $((v, 0); b)$.

*Sigmoid (*Sigmoid*):* Sigmoid is another widely-used non-linear activation function in PPML and is given as:

$$\mathsf{Sigmoid}(v) = \begin{cases} 0 & \text{if} \quad v < -\frac{1}{2} \\ v + \frac{1}{2} & \text{if} \quad -\frac{1}{2} \leq v \leq \frac{1}{2} \\ 1 & \text{if} \quad v > \frac{1}{2} \end{cases}$$

Let the bits b_1 and b_2 denote whether $v < -\frac{1}{2}$ and $v < \frac{1}{2}$ respectively. If yes, the bits are set to 1 and 0 otherwise. The bits b_1 and b_2 can be computed as MSB $(v + \frac{1}{2})$ and MSB $(v + \frac{1}{2})$ respectively using the BitExt circuit. Then, $\mathsf{Sigmoid}(v)$ can be computed using a 4-to-1 MUX with inputs $((1, v + \frac{1}{2}, 0, 0); (b_1, b_2))$.

*Maxpool (*Maxpool*):* Maxpool is a popular technique used mainly in Convolutional Neural Networks (CNN) to reduce the complexity and extract low-level features from the neighbourhood of the nodes. Given n values, Maxpool selects the maximum among them. As shown in [9], [32], a n-to-1 Maxpool circuit can be built using $(\mathsf{n} - 1)$ MAX-2 circuits following a tree-based approach. A MAX-2 circuit consists of a comparison (COMP) followed by a 2-to-1 MUX. In SynCirc, in addition to MAX-2, we use the MAX-3 circuit that computes the maximum of three values in one shot. The circuit consists of three COMP and one 8-to-1 MUX as shown in Fig. 9. Our n-to-1 Maxpool circuit has a depth of $(\log_4(\ell) + 2)\log_3(\mathsf{n})$ as opposed to $(\log_2(\ell) + 2)\log_2(\mathsf{n})$ in [9], [32]. For PPML applications, with values represented in FPA semantics, we can replace the comparison (COMP) with a bit extraction (BitExt) circuit.

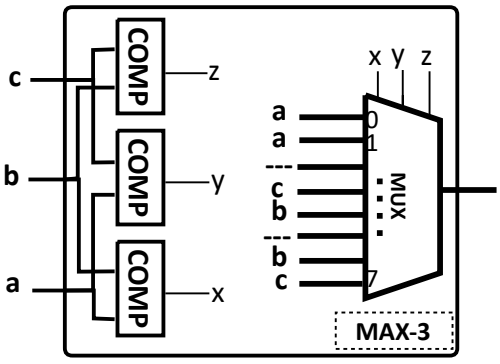

Fig. 9. MAX-3 circuit to find the maximum of 3 numbers [37].

*Argmax (*Argmax*):* The Argmax circuit is an extension to Maxpool where the index of the maximum element is also computed. For this, we associate an additional 2-to-1 MUX with every MAX-2 and 8-to-1 MUX with every MAX-3 block in the Maxpool circuit, which keeps track of the index of the maximum element [32].

V. Evaluation

A. Experimental Setup

We implemented all functionalities in Verilog and synthesized the netlists using the open-source logic synthesis Yosys-

TABLE VI
SYNTHESIS RESULTS OF IMPROVED BUILDING BLOCKS COMPARED TO BEST CIRCUITS IN THE LITERATURE FOR INPUTS OF BITWIDTH ℓ.

Circuit	Bitwidth	Literature		SynCirc					Depth Improvement
		AND2	Depth	AND2	AND3	AND4	#AND (Total)	Depth	
Layer II - Basic Functionalities									
$\mathsf{ADD}_{\mathsf{CLF}}$ [31]	$\ell = 8$	24	4	8	8	5	21	**3**	1.3×
	$\ell = 16$	64	5	21	15	11	47	**3**	1.7×
	$\ell = 32$	160	6	50	43	54	147	**3**	2.0×
	$\ell = 64$	384	7	109	133	342	584	**4**	1.8×
$\mathsf{SUB}_{\mathsf{CLF}}$ [31]	$\ell = 16$	129	6	21	19	19	59	**4**	1.5×
	$\ell = 32$	311	7	49	56	69	174	**4**	1.8×
	$\ell = 64$	705	8	109	156	378	593	**5**	1.6×
BitExt [16]	$\ell = 8$	22	5	2	5	2	9	**4**	1.3×
	$\ell = 16$	46	6	6	6	9	21	**4**	1.5×
	$\ell = 32$	94	7	14	15	20	49	**4**	1.8×
	$\ell = 64$	190	8	26	27	54	107	**5**	1.6×
$\mathsf{MUL}_{\mathsf{CLF}}$ [31]	$\ell = 8$	152	9	109	15	11	135	**7**	1.3×
	$\ell = 16$	576	11	466	43	54	563	**8**	1.4×
	$\ell = 32$	2 208	13	1 933	133	342	2 408	**10**	1.3×
COMP [25]	$\ell = 16$	42	5	10	11	5	26	**3**	1.7×
	$\ell = 32$	89	6	20	21	16	57	**3**	2.0×
	$\ell = 64$	184	7	30	32	32	94	**4**	1.8×
4-to-1 MUX [25]	$\ell = 16$	48	2	32	32	–	64	**1**	2.0×
	$\ell = 32$	96	2	64	64	–	128	**1**	2.0×
	$\ell = 64$	192	2	128	128	–	256	**1**	2.0×
8-to-1 MUX [25]	$\ell = 16$	112	3	–	64	64	128	**1**	3.0×
	$\ell = 32$	224	3	–	128	128	256	**1**	3.0×
	$\ell = 64$	448	3	–	256	256	512	**1**	3.0×
EQ [25]	$\ell = 16$	15	4	–	–	5	5	**2**	2.0×
	$\ell = 32$	31	5	1	–	10	11	**3**	1.7×
	$\ell = 64$	63	6	–	–	21	21	**3**	2.0×
AES Sbox [62]		34	4	30	4	–	34	**3**	1.3×
Layer III - Advanced Functionalities									
DIV [25]	$\ell = 16$	1 542	93	697	672	1 563	2 932	**69**	1.4×
	$\ell = 32$	7 079	207	2 662	3 669	13 133	19 464	**189**	1.1×
SORT [68] ($\mathsf{n} = 16$)	$\ell = 16$	4 800	60	2 080	880	400	3 360	**40**	1.5×
$\mathsf{DST}_{\mathsf{M}}$ [68]	$\ell = 16$	241	13	133	91	87	311	**8**	1.7×
PSI [29], [51] ($\mathsf{n} = 2^{11}, \delta = 32$)		32 736	10	–	–	10 912	10 912	**5**	2.0×
ReLU [9]	$\ell = 16$	62	7	22	6	9	37	**5**	1.4×
	$\ell = 32$	126	8	46	15	20	81	**5**	1.6×
	$\ell = 64$	254	9	90	27	54	171	**6**	1.5×
Sigmoid [9]	$\ell = 16$	140	8	44	44	18	106	**5**	1.6×
	$\ell = 32$	284	9	92	94	40	226	**5**	1.8×
	$\ell = 64$	572	10	180	182	108	470	**6**	1.7×
Maxpool [25] ($\mathsf{n} = 16$)	$\ell = 16$	870	24	236	690	558	1 484	**12**	2.0×
	$\ell = 32$	1 815	28	472	910	1 248	2 630	**12**	1.8×
	$\ell = 64$	3 720	32	724	1 600	2 496	4 820	**15**	2.1×
Argmax [25] ($\mathsf{n} = 16$)	$\ell = 16$	1 110	24	252	1 138	1 006	2 396	**12**	2.0×
	$\ell = 32$	2 295	28	504	1 806	2 144	4 454	**12**	2.3×
	$\ell = 64$	4 680	32	788	3 392	4 288	8 466	**15**	2.1×

ABC [38], [39]. All the experiments have been carried out on a machine with an Intel Core i9-7960X CPU @ 2.80 GHz and 128 GB of RAM. We compared all the synthesized circuits in our framework with their state-of-the-art counterparts. The multiplicative depth that refers to the number of AND gates (including multi-input ones) in the circuit's critical path is the primary metric for our benchmarking as it determines the round complexity in MPC. We also report the circuit size in the number of non-free AND gates (AND2, AND3, and AND4). Since ABY2.0 (and most of the secure computation protocols) allows the computation of XOR gates locally without incurring any communication, we do not count XORs in our benchmarking.

B. Benchmark Evaluation

We report the details of all the circuits obtained using our synthesis framework in Table VI. As evident from the table, we improve the multiplicative depth of all the circuits under consideration, and the improvement ranges from 1.1× to 3×. This improvement is amplified for applications where these

TABLE VII
COMPARISON OF SYNCIRC WITH HAND-OPTIMIZED $\mathsf{ADD}_{\mathsf{CLF}}$ AND BitExt CIRCUITS IN ABY2.0 [37].

Circuit	Bitwidth	Approach	AND2	AND3	AND4	Depth	Setup (bits)	Online (bits)
$\mathsf{ADD}_{\mathsf{CLF}}$ [31]	$\ell = 8$	**ABY2.0 [37]**	15	6	1	3	3 956	44
		SynCirc	8	8	5		5 280	42
	$\ell = 16$	**ABY2.0 [37]**	216	184	179	4	153 888	1 158
		SynCirc	109	133	342		198 469	1 168
BitExt [16]	$\ell = 8$	**ABY2.0 [37]**	7	4	1	4	2 382	24
		SynCirc	2	5	2		2 403	18
	$\ell = 16$	**ABY2.0 [37]**	41	27	47	5	32 951	230
		SynCirc	26	27	54		34 021	214

TABLE VIII
COMPARISON OF SYNCIRC WITH DEPTH-OPTIMIZED LUT-BASED CIRCUITS IN [26] FOR 32-BIT VALUES. THE FIRST LINE FOR EACH CIRCUIT IS THE CIRCUIT OF [26] OPTIMIZED FOR SP-LUT AND THE SECOND ONE OPTIMIZED FOR OP-LUT.

Circuit	Lookup Table (SP-LUT [26])			Lookup Table (OP-LUT [26])			SynCirc (ABY2.0 [37])			Improvement over OP-LUT	
	Setup (bits)	Online (bits)	Round	Setup (bits)	Online (bits)	Round	Setup (bits)	Online (bits)	Round	Setup	Online
Adder	22 161	2 199	5	49 105	702	5	41 253	294	3	1.2×	2.4×
	6 174	15 487	3	3 657 974	382	3				88.7×	1.3×
Comparator	7 859	861	5	18 867	250	5	14 991	114	3	1.3×	2.2×
	2 691	4 322	3	1 065 309	164	3				71.1×	1.4×
Multiplier	356 433	46 691	14	955 649	11 326	14	442 885	4 816	10	2.2×	2.4×
	123 614	395 790	8	93 193 239	7 484	8				210.4×	1.6×
AES-Sbox	37 209	5 471	5	113 561	1 230	5	5 024	68	3	22.7×	18.1×
	247	2 056	1	524 288	16	1				104.4×	0.24×

circuits contribute to most of the online rounds. For instance, consider the ReLU circuit where we improve the depth by at least 1.5×. As shown in ABY2.0 [37], the ReLU circuit contributes to more than 90% of the online rounds for a two-layer deep neural network, showcasing the significance of our improvement. The comparison between the automatically generated circuits in SynCirc and hand-optimized circuits in ABY2.0 [37] is given in Table VII. As seen in the table, we can automatically generate multiplicative depth-optimized circuits with the same multiplicative depth and similar communication complexity.

For the analysis of the online communication, it is sufficient to compare the total number of non-free gates in the circuit as ABY2.0 has the same online communication for all AND gates irrespective of the fan-in of the gate. When evaluated using the ABY2.0 protocol, we observe our circuits incur less communication (by a factor of 1.1× - 3×) over the best previous work for almost all the cases, with some exceptions. Also, the overhead in online communication for some of the circuits like division is capped by a factor of at most 2.7×. However, as pointed out in ABY2.0, for the secure computation over a high latency network such as the Internet, the number of online rounds plays a crucial role in determining the protocol efficiency over online communication. Thus the evaluation of our circuits with ABY2.0 provides good online performance w.r.t. both communication and rounds.

C. Comparison of SynCirc with Lookup Table (LUT)-based MPC

Here we compare our circuits with those in Dessouky et al. [26] that deploys the LUT-based approach. They propose two variants with different trade-offs: i) OP-LUT, which optimizes online communication, and ii) SP-LUT, which optimizes setup/total communication. In Table VIII, we compare the efficiency of the circuits from our SynCirc compiler evaluated with ABY2.0 with LUT-optimized circuits and protocols of [26]. Since our work aims at online efficiency, the last column in that table shows our improvements over OP-LUT which also optimizes for that metric [26]. We outperform the OP-LUT in online communication for all the cases except the AES-SBox. This is expected as the AES-SBox is a highly non-linear 8×8 lookup table and hence can be efficiently realized using the LUT method by simply evaluating the S-box on all 2^8 possible inputs. For all other circuits, our improvement in online communication over OP-LUT ranges from 1.3× - 1.6×. The setup communication is improved between 71× - 210× while having the same number of online rounds. When compared with the SP-LUT variant that optimizes for setup/total communication, our circuits have 3.5× - 24× more setup communication. However, we improve the online communication over SP-LUT by 30× - 82×.

VI. CONCLUSION AND FUTURE DIRECTIONS

The generation of optimized error-free Boolean circuits is a crucial stage in deploying secure computation protocols. Towards this, we present *SynCirc*, the first hardware synthesis framework, accommodating multi-input AND gates, to generate depth-optimized circuits for secure computation. We enriched the framework with several advanced functionalities like division, sorting, Manhattan distance, private set intersection, and privacy-preserving machine learning tools. *SynCirc* outperforms state-of-the-art compilers for secure computation in circuit depth by up to 2.3×. Future work can extend to more building blocks like floating-point operations or combine multi-input ANDs with more costly but arbitrary lookup tables.

ACKNOWLEDGEMENTS

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) — SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE.

Arpita Patra would like to acknowledge financial support from SERB MATRICS (Theoretical Sciences) Grant 2020, Google India AI/ML Research Award 2020, and DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-CPS) 2020. Ajith Suresh would like to acknowledge financial support from Google PhD Fellowship 2019.

REFERENCES

[1] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "Syncirc: Efficient synthesis of depth-optimizedcircuits for secure computation," in *HOST*, 2021.

[2] A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986.

[3] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *STOC*, 1987.

[4] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *ACM Conference on Electronic Commerce*, 1999.

[5] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A. R. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "SAFELearn: Secure Aggregation for private FEderated Learning," in *DLS*, 2021.

[6] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017.

[7] H. Keller, H. Möllering, T. Schneider, and H. Yalame, "Balancing quality and efficiency in private clustering with affinity propagation." SECRYPT, 2021.

[8] A. Hegde, H. Möllering, T. Schneider, and H. Yalame, "SoK: Efficient privacy-preserving clustering." PETS, 2021.

[9] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *IEEE S&P*, 2017.

[10] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High throughput 3PC over rings with application to secure prediction," in *CCSW*, 2019.

[11] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC framework for privacy preserving machine learning," in *NDSS*, 2020.

[12] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning," in *PETS*, 2020.

[13] A. Patra and A. Suresh, "BLAZE: blazing fast privacy-preserving machine learning," in *NDSS*, 2020.

[14] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: super-fast and robust privacy-preserving machine learning," in *USENIX Security*, 2021.

[15] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *ARES*, 2020.

[16] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *CCS*, 2018.

[17] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security*, 2020.

[18] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay: Secure two-party computation system." in *USENIX Security*, 2004.

[19] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A system for secure multi-party computation," in *CCS*, 2008.

[20] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries," in *USENIX Security*, 2012.

[21] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith, "Secure two-party computations in ANSI C," in *CCS*, 2012.

[22] T. Schneider and M. Zohner, "GMW vs. Yao? Efficient secure two-party computation with low depth circuits," in *FC*, 2013.

[23] B. Kreuter, A. Shelat, B. Mood, and K. Butler, "PCF: A portable circuit format for scalable two-party secure computation," in *USENIX Security*, 2013.

[24] 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.

[25] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni, "Automated synthesis of optimized circuits for secure computation," in *CCS*, 2015.

[26] G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables." in *NDSS*, 2017.

[27] E. Testa, M. Soeken, H. Riener, L. Amaru, and G. De Micheli, "A logic synthesis toolbox for reducing the multiplicative complexity in logic networks," in *DATE*, 2020.

[28] E. Testa, M. Soeken, L. Amarù, and G. De Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *DAC*, 2019.

[29] M. S. Riazi, M. Javaheripi, S. U. Hussain, and F. Koushanfar, "MPCircuits: Optimized circuit generation for secure multi-party computation." in *HOST*, 2019.

[30] B. Mood, L. Letaw, and K. Butler, "Memory-efficient garbled circuit generation for mobile devices," in *FC*, 2012.

[31] N. Buescher, A. Holzer, A. Weber, and S. Katzenbeisser, "Compiling low depth circuits for practical secure computation," in *ESORICS*, 2016.

[32] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *CANS*, 2009.

[33] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *ICALP*, 2008.

[34] T. Heldmann, T. Schneider, O. Tkachenko, C. Weinert, and H. Yalame, "LLVM-Based circuit compilation for practical secure computation," in *ACNS*, 2021.

[35] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*, 2010.

[36] D. Demmler, T. Schneider, and M. Zohner, "ABY – A framework for efficient mixed-protocol secure two-party computation," in *NDSS*, 2015.

[37] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *USENIX Security*, 2021.

[38] "Berkeley logic synthesis. ABC: a system for sequential synthesis and verification," https://github.com/berkeley-abc/abc, 2010.

[39] "Yosys open synthesis suite," http://www.clifford.at/yosys/, 2013.

[40] S. Ohata and K. Nuida, "Communication-efficient (client-aided) secure two-party protocols and its application," in *FC*, 2020.

[41] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay – secure two-party computation system." in *USENIX Security*, 2004.

[42] "Synopsys Inc. design compiler," http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler, 2010.

[43] J. Garay, B. Schoenmakers, and J. Villegas, "Practical and secure solutions for integer comparison," in *PKC*, 2007.

[44] C. Wolf, J. Glaser, and J. Kepler, "Yosys – a free verilog synthesis suite," in *Austrian Workshop on Microelectronics*, 2013.

[45] "Synopsys Inc. design ware library - datapath and building block IP." https://www.synopsys.com/dw/buildingblock.php, 2015.

[46] M. S. Javadi, H. Yalame, and H. R. Mahdiani, "Small constant mean-error imprecise adder/multiplier for efficient VLSI implementation of MAC-Based applications," *IEEE Transactions on Computers*, 2020.

[47] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design," *Journal of Computer Security*, 2013.

[48] D. Harris, "A taxonomy of parallel prefix networks," in *Signals, Systems & Computers*, 2003.

[49] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," in *JACM*, 1980.

[50] C. S. Wallace, "A suggestion for a fast multiplier," in *TEC*, 1964.

[51] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[52] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient circuit-based PSI with linear communication," in *EUROCRYPT*, 2019.

[53] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, "Efficient circuit-based PSI via cuckoo hashing," in *EUROCRYPT*, 2018.

[54] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *USENIX Security*, 2015.
[55] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on ot extension," *TOPS*, 2018.
[56] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *International Journal of Information Security*, 2012.
[57] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *ASIACRYPT*, 2009.
[58] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *USENIX Security*, 2019.
[59] L. T. Brandão, N. Christin, and G. Danezis, "Toward mending two nation-scale brokered identification systems," in *PETS*, 2015.
[60] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *EUROCRYPT*, 2015.
[61] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits." in *USENIX Security*, 2011.
[62] J. Boyar and R. Peralta, "A small depth-16 circuit for the AES S-box," in *IFIP International Information Security Conference*, 2012.
[63] J. Boyar, P. Matthews, and R. Peralta, "Logic minimization techniques with applications to cryptology," *Journal of Cryptology*, 2013.
[64] F. Kerschbaum, T. Schneider, and A. Schröpfer, "Automatic protocol selection in secure two-party computations," in *ACNS*, 2014.
[65] N. Agrawal, A. Shahin Shamsabadi, M. J. Kusner, and A. Gascon, "QUOTIENT: Two-party secure neural network training and prediction," in *CCS*, 2019.
[66] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *CCS*, 2007.
[67] R. Pibernik, Y. Zhang, F. Kerschbaum, and A. Schropfer, "Secure collaborative supply chain planning and inverse optimization – The JELS model," *EJOR*, 2011.
[68] K. Järvinen, H. Leppäkoski, E.-S. Lohan, P. Richter, T. Schneider, O. Tkachenko, and Z. Yang, "PILOT: Practical privacy-preserving indoor localization using outsourcing," in *EuroS&P*, 2019.
[69] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of Spring Joint Computer Conference*, 1968.
[70] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, and F. Koushanfar, "Compacting privacy-preserving k-nearest neighbor search using logic synthesis," in *DAC*, 2015.