Arithmetic and Boolean Secret Sharing MPC on FPGAs in the Data Center

Rushi Patel*, Pierre-François Wolfe†, Robert Munafo‡, Mayank Varia§, and Martin Herbordt¶
* † ‡ ¶ Dept. of Electrical and Computer Engineering & §Dept. of Computer Science,
Boston University, Boston, USA
Email: *ruship@bu.edu, †pwolfe@bu.edu, ‡rmunafo@bu.edu, §varia@bu.edu, ¶herbordt@bu.edu

***Abstract*—Multi-Party Computation (MPC) is an important technique used to enable computation over confidential data from several sources. The public cloud provides a unique opportunity to enable MPC in a low latency environment. Field Programmable Gate Array (FPGA) hardware adoption allows for both MPC acceleration and utilization of low latency, high bandwidth communication networks that substantially improve the performance of MPC applications. In this work, we show how designing arithmetic and Boolean Multi-Party Computation gates for FPGAs in a cloud provide improvements to current MPC offerings and ease their use in applications such as machine learning. We focus on the usage of Secret Sharing MPC first designed by Araki et al [1] to design our FPGA MPC while also providing a comparison with those utilizing Garbled Circuits for MPC. We show that Secret Sharing MPC provides a better usage of cloud resources, specifically FPGA acceleration, than Garbled Circuits and is able to use at least a 10× less computer resources as compared to the original design using CPUs.**

***Index Terms*—Multiparty Computation, Secret Sharing, Secure Computation, FPGA, Data Center, Cloud Service, Machine Learning, Matrix Multiplication**

I. Introduction

Confidential data is everywhere around us, personal, financial, medical, and government. We hear about data leaks regularly and lack trust in others when it comes to maintaining privacy and security in our own data. The problem, however, is that we may greatly benefit from sharing data and performing joint computations, but are unable to do so easily with confidential data. Multi-party computation (MPC) allows parties to securely share confidential data and compute collective information without ever releasing one's own personal data. Cloud systems for confidential data have been limited to privately owned systems and are less widely utilized by those requiring more computation power at any given moment. Thus, the public cloud is rarely considered by those interacting with confidential data.

MPC has been an active area of research for the last 40 years [2]–[5], and it has been deployed to protect data in the healthcare [6], [7], education [8], [9], finance [10]–[12], and technology [13], [14]. In addition, the use of MPC for machine learning applications is actively being researched [15]. Existing work shows that general-purpose MPC can be used in viable systems [16].

MPC currently focuses on two main approaches. Secret Sharing allows for a balanced arithmetic intensity between compute and network bandwidth, but it relies on a low latency environment to achieve high performance results. In contrast, Garbled Circuits typically perform better in higher latency situations, due to fewer communication rounds, but require high bandwidth to support large circuits. Because Garbled Circuit approaches are typically compute-bound, they appear more amenable to hardware acceleration, and as such have been the subject of significant prior research with FPGAs [17]–[27]. The evolving data center has seen a considerable rise in accelerators to further improve both computation and communication. This change suggests the current approach to MPC on FPGAs may need to be reconsidered to better utilize a growing and adaptive cloud environment.

Prior work [28] shows the first findings about the performance of Boolean only Secret Sharing on an FPGA. We build upon this work by extending the protocol to include arithmetic Secret Sharing and by providing a roofline plot to justify the advantage of Secret Sharing in the data center. The roofline model analysis of MPC using CPUs and FPGAs finds a compelling argument for hardware acceleration of MPC via Secret Sharing vs Garbled Circuits when deployed in a data center. Using a low latency and high bandwidth environment, powered with high performance transceivers, we show an effective MPC implementation with FPGA accelerators in a cloud data center performing more multi-party computations than alternatives providing a compelling argument for the usage of MPC in high performance applications such as machine learning.

In this paper, we examine the trade-offs between Garbled Circuits and Secret Sharing in cloud data centers using a roofline model, implement both arithmetic and Boolean Secret Sharing in hardware, test this hardware design, and assess its scalability for machine learning applications by analyzing its usage in matrix multiply. We conclude by proposing directions for future work towards a complete Secret Sharing MPC machine learning application using FPGAs running in a cloud.

We summarize the contributions in this work:

- We extend Secret Sharing MPC on FPGA hardware to include both arithmetic and Boolean protocols.
- We develop a roofline model to show the effectiveness of Secret Sharing utilizing FPGAs in a data center.
- Using 5.5% of FPGA fabric in a consumer cloud environment, we match the throughput of an optimized 20-core CPU implementation saturating a typical 10Gbps network connection. This result scales with available bandwidth allowing for a single FPGA to saturate a 200Gbs link.

II. Background

A. Data Center Model

The target data center model used here is the adoption of a MPC-as-a-service which can utilize the various configurations of accelerators found in different cloud and cluster setups. The MPC use case requires multiple computing and communicating parties for security and low latency networking for performance. Thus we consider scenarios where data center processing hardware is owned by different parties and co-located within a single physical location. This helps to limit the additional bottlenecks found when traversing physical distances to perform low-latency operations.

FPGA hardware acceleration has seen increasing adoption in data centers due to their very high bandwidth and low-latency communication used by means of high speed transceivers. As described in Section II-B, FPGA hardware properties, especially co-location of compute and communication logic on the same device, yield high throughput for MPC protocols based on Secret Sharing, which makes the most effective use of available bandwidth.

Current and projected offerings in commercial cloud systems include FPGAs as co-accelerators (connected only to the CPU), FPGAs connected to CPU but interconnected with each other through their own dedicated secondary network [29]–[32], or as network attached devices in various configurations. In this last configuration, the FPGA is connected directly to the network, either as a stand-alone leaf node [33], on a network interface card (NIC) [34], or as an inline accelerator (Bump-in-the-Wire) between the NIC and the network [35]–[37].

The interconnected cluster and network-attached configurations provide low-latency network connected systems which helps to benefit MPC secret sharing computation. The former co-accelerator model can be utilized with parties co-located on a single FPGA. We utilize this model as our experiment platform as it is readily available on current commercial cloud offerings. We plan to investigate an FPGA network model in our future work as it will help to address some security and confidentiality concerns when co-locating parties on a single hardware resource. Maximizing throughput is a focus for this work as this metric determines how efficiently multiple client tasks can be completed.

B. MPC Paradigms

In general, MPC protocols allow an arbitrary number of compute parties N to perform a joint computation while resisting a subset of T 'bad' parties who wish to breach the confidentiality of other people's data or tamper with the integrity of the calculation. In this work, we examine a 3-party protocol which tolerates 1 adversarial party who "semi-honestly" follows the protocol and only tries to break confidentiality. This matches a scenario in which a small number of FPGAs owned by different parties are co-located within a data center.

General-purpose MPC designs often represent the agreed-upon computation as an arithmetic or Boolean circuit, and follow the Garbled Circuit or Secret Sharing approaches. Garbled Circuits rely on one compute party generating a (large) encoded version of the entire circuit, which it then transmits to a second party who can evaluate the encoded circuit on encoded inputs in order to obtain the answer. On the other hand, Secret Sharing-based MPC systems have the compute parties evaluate each gate of the circuit in parallel on their own pieces or *shares* of the data, with a small amount of network communication required for each multiplication or AND gate (none is required for addition or XOR gates).

The computation and communication overhead of MPC manifests itself differently for Garbled Circuits and Secret Sharing. Even with optimizations [38]–[42], Garbled Circuits have a small number of communication rounds but a large communication size (80-128$\times$ the size of the original data), rendering them beneficial in high-latency scenarios but detrimental when processing large datasets. Conversely, Secret Sharing approaches require a low-latency environment because they involve many rounds of communication, however they consume substantially less bandwidth per computational step.

To date, most MPC implementations are in software, and thus rely on general-purpose processing hardware and commodity networking equipment. In this scenario, Secret Sharing tends to be network latency-bound whereas Garbled Circuits are often compute-bound. Consequently, most of the prior focus in hardware acceleration has been directed toward Garbled Circuits. Our work specifically considers MPC implementations in the data center, where Secret Sharing systems offer higher maximum throughput and the network latency can be low enough to realize meaningful performance benefits by optimizing the computation with FPGAs.

C. Selected MPC Protocol

The original Secret Sharing implementation [1], [43] differentiates between arithmetic and Boolean operations. The design by Araki et al. requires that 3 parties agree to perform a computation, however it only requires each party to communication with one other party at a time (aside from initial secret sharing and revealing the final result). This means for most actions 1 adversarial party is tolerated, but breaks the trust in the system when employed in higher application scenarios.

We focus on the implementation of both types of Secret Sharing circuits, Boolean and arithmetic. The main difference between these two categories is based around the use of a ring modulo 2^n. For all $n > 1$ operations are categorized into arithmetic gates either addition or multiplication. We opted to use $n = 128$ for our arithmetic implementation. For the special case $n == 1$, addition is the same as XOR and multiplication is the same as AND.

The process of MPC secret sharing involves 3 phases. First, confidential data is securely split into three *shares* and given to each party member. Each party's share does not contain enough information to deduce the original confidential data. The protocol dictates that only 2 parties' shares are necessary to reconstruct the final result. Computation is performed using either the Boolean XOR and AND gates or arithmetic addition, and multiplication gates. The decision to use either form of

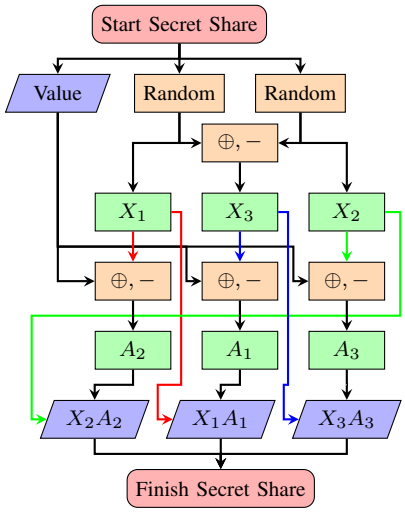


Fig. 1: Initial construction of three party secret shares

Secret Sharing is made as a group between all parties either before the share splitting phase occurs or after a transformation is done between secret sharing protocols. Reconstruction of the final value is done with the help of another party member's final share information. We explore the details of each step below for the protocol implemented.

1) Data Split - Share construction: Each share consists of two pieces: one uniformly random value, and one value based on data passed from another party. We denote each share as a pair (x_i, a_i) and v as our confidential data being shared among the party. Shares are generated and distributed based on the computation being performed, either arithmetic or Boolean logic. Shares are generated using uniformly random values x_1, x_2, x_3 which follow the rule stated in Eq. 1. Based on the protocol form of Secret Sharing, values for (x_i, a_i) are created using either Eq. 2 or Eq. 3, where if $i = 1$ then $i - 1 = 3$. The process of generating three unique shares for the confidential value v is visualized in Figure 1.

$$x_1, x_2, x_3 \in \mathbb{Z}_{2^n} \tag{1}$$

Arithmetic:

$$x_1 + x_2 + x_3 = 0 \text{ then } a_i = x_{i-1} - v \tag{2}$$

Boolean:

$$x_1 \oplus x_2 \oplus x_3 = 0 \text{ then } a_i = x_{i-1} \oplus v \tag{3}$$

2) Gate operations - Computation phase: To perform any operation two shares are passed to each party, one for each data value 'v_1, v_2'. We denote an input pair for v_1 as (x, a), v_2 as (y, b) and the output of the operation as (z, c).

- *XOR*: Each party can compute the XOR of their individual shares simply by performing a local xor of the individual parts in the pair.

$$z = x \oplus y \text{ and } c = a \oplus b \tag{4}$$

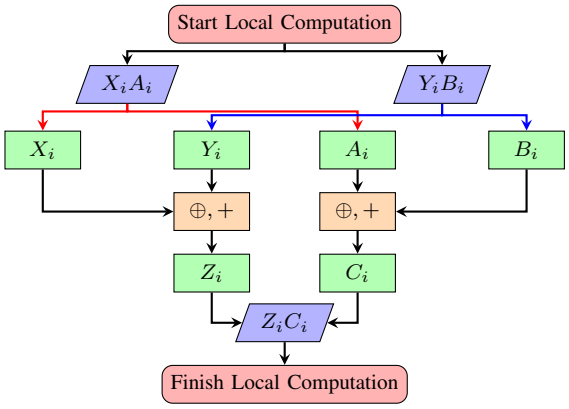


Fig. 2: Local computation of XOR, addition gate

- *Addition*: Similar to XOR, individual parties can perform a simple addition of arithmetic shares using their local information without the need for communication.

$$z = x + y \text{ and } c = a + b \tag{5}$$

- *AND*: Performing an AND operation requires parties to communicate with one another and thus is more complex. First, each party i produces a correlated random value $\alpha_i \in \{0, 1\}$ which follows Eq. 6. From here we calculate r_i using Eq. 7. Lastly, each party passes their r_i value to one other party member and finally calculates the output share (z, c).

$$\alpha_1 \oplus \alpha_2 \oplus \alpha_3 = 0 \tag{6}$$

$$r = x\&y \oplus a\&b \oplus \alpha \tag{7}$$

$$z = r_i \oplus r_{i-1} \text{ and } c = r_i \tag{8}$$

- *Multiply*: Much of the computation for multiply is similar to the AND gate previously described. Parties select correlated random values that hold $\alpha_i \in \mathbb{Z}_{2^n}$, following Eq. 9. Calculation of each r_i is done using Eq. 10. Due to the unique nature of the shares and their set associativity, we take advantage of the modular multiplicative inverse for our q value. Similar to the communication necessary in the AND gate, each party passes their generated r_i and calculates their individual shares (z, c).

$$\alpha_1 + \alpha_2 + \alpha_3 = 0 \tag{9}$$

$$r = (a \cdot b - x \cdot y + \alpha) \cdot q \text{ where } q \cdot 3 \equiv 1 \pmod{2^n} \tag{10}$$

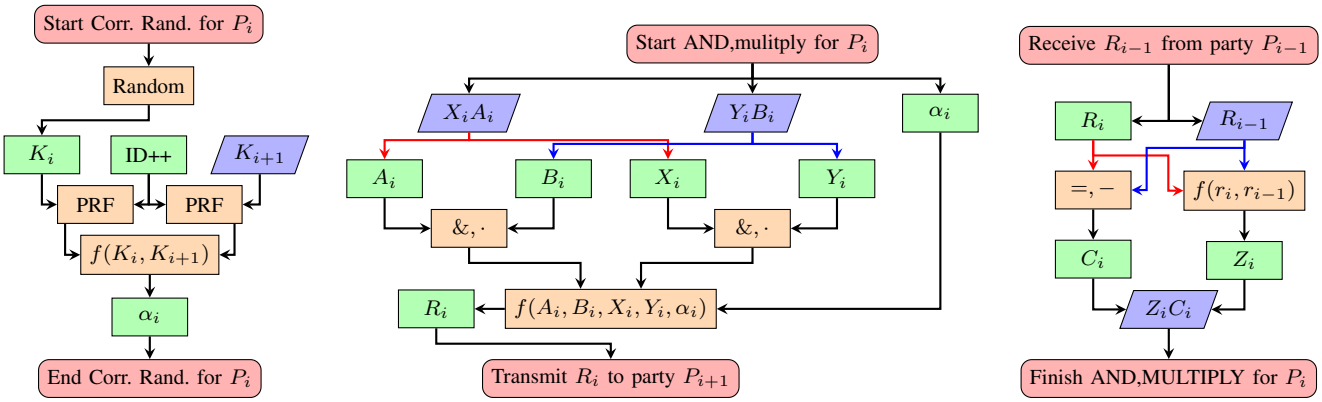
$$z = r_{i-1} - r_i \text{ and } c = -2r_{i-1} - r_i \tag{11}$$

3) Final results - Data reconstruction: Finally once all computation is performed on the party shares, each member can reconstruct the final value by requesting information from one other party member.
Arithmetic:

$$v' = z_{i-1} - c_i \tag{12}$$

Boolean:

$$v' = z_{i-1} \oplus c_i. \tag{13}$$



(a) correlated random value (b) initial computation and exchange (c) final computation

Fig. 3: Party i's contribution toward computing an AND,multiply gate

We focus on the Araki et al. Secret Sharing using 3 parties as our starting point. Extensions of this protocol can provide more security over malicious individuals or allow for more party members [44]–[46]. Our goal is to provide evidence toward the benefits of accelerating MPC in the data center while leaving room to adapt the protocol for future use cases.

III. Design & Implementation

A. FPGA Implementation

Our designs for the arithmetic and Boolean versions of Secret Sharing contain many common details and features. This provides the opportunity to customize the allocation and utilization of gates in the design at a higher level without the need to reconfigure the hardware. We will discuss in depth the process involved when utilizing AND and multiply gates as those require different calculation and communication actions. XOR and addition gates are done locally and do not require much additional design choices as seen in Figure 2.

We rely on OpenCores projects [47], [48] for our design when generating pseudo-random numbers. This design choice was made out of convenience to eaily adapt the design to any hardware provided. Our future plans are to utilize vendor specific Random Number Generator (RNG) hard cores or reference designs created by FPGA vendors to provide a better optimized version for targeted hardware. Security of the OpenCores RNG module was not considered for this application but will be examined in the future when making a selection between different vendor specific tools and designs. We use the selected random number during initial key generation, share splitting phase, and the correlated random requirements in both AND and multiply gates.

The usage of a correlated random number between each of the three parties is important to both following the protocol listed in Section II-C2 as well as providing a method for all parties to obtain random values without the need for additional communication. Figure 3a shows the process we developed for our protocol. Prior to performing any MPC computations each party generates an initial key through the use of the RNG block and maintains a copy of the key value and shares this same value with one other party member in round robin fashion. These keys are then used to generate a pseudo-random value of 128-bits using both keys in possession. The function $f(K_i, K_{i+1})$ is pre-determined when the party selects to use Boolean or arithmetic operations.
Arithmetic:

$$\alpha_1 + \alpha_2 + \alpha_3 = 0 \text{ and } \alpha_i = PRF(K_i) - PRF(K_{i+1}) \quad (14)$$

Boolean:

$$\alpha_1 \oplus \alpha_2 \oplus \alpha_3 = 0 \text{ and } \alpha_i = PRF(K_i) \oplus PRF(K_{i+1}) \quad (15)$$

Calculation of α must be performed for each use of an AND or multiply gate to provide additional security to the protocol and prevent replay attacks on the system.

B. Analysis of FPGA Implementation

The FPGA implementation uses Amazon Web Services (AWS) FPGAs available through its Elastic Compute Cloud (Amazon EC2). Specifically, Xilinx Virtex UltraScale+ VU9P FPGAs are accessible via a virtual machine in EC2 F1 instances. Amazon includes a hardware shell for software/hardware co-design between the node CPU (Intel Xeon E5-2686 v4) and FPGA. Software control for F1 instances rely on provided DMA functions and PCIe function templates which communicate directly with the hardware shell [49]. This furnishes the mechanism for loading data, controlling operations, and retrieving results.

The PCIe packets are translated through the Amazon shell and utilize multiple AXI bus configurations to send and receive data with the software system. We use the general purpose AXI bus supporting a 512 bit data packet to provide a single message containing two secret share vectors (4×128-bits) prior to starting the hardware operations. The HDL design takes each AXI bus message, parses the information, and relays data to the desired module.

Since all gates are routed individually, connecting gates to form a specific circuit or equation can be done either through the software host process or with an agreed upon hardware change between all party members. This provides us

the opportunity to route multiply and addition gates to function together for matrix multiply as individual connections or as a joint multiply accumulate (MAC) module. Previous work done with matrix multiply shows efficient methods of using Garbled Circuits to accelerate the generation of MPC circuitry before passing the garble tables over to the participating party member [21], [26], [27]. Our approach to matrix multiply using secret sharing performs all computation jointly during run-time. We discuss later in Section IV-B how our FPGA implementation can fully utilize the network bandwidth found in the data center for larger applications such as machine learning.

IV. Results

A. Testing and Data

MPC Secret Sharing modules are assessed in terms of total FPGA resource utilization and total throughput with all gates running in parallel.

Our final implementation requires only a single cycle to perform either an XOR or addition gate and a total of 4 cycles to calculate an AND or multiply gate. This is obtainable after an initial 21 cycles of pre-computation of party keys and initial correlated random values. The computation for each additional correlated random value is performed during the utilization of the current peusdo-random value during AND or multiply gate evaluation.

At the time of this publication, Amazon F1 does not currently offer their "FPGA-Link" and thus only provides FPGA to FPGA connections through PCIe or over their 25GbE network through a host NIC. Our current design focuses on obtaining the most utilization on a single FPGA and thus is not limited based on network speed. For verification and to fully utilize our design without network bottlenecks, we designed a test environment allowing for 3 parties on a single FPGA. This is a possible use case in a real world environment through the use of a trusted third party to perform calculations on behalf of all three party members. This design included the necessary routing and control logic which enabled the system to perform calculations between all parties from start to end without the need of software intervention. The test environment allocates 3 AND or multiply gates, one for each party member, then each group is duplicated to fully utilize the FPGA; summarized in Table I."

We include the number of bits in Table I as a measure of the amount of data communicated during the exchange part of each computation cycle.

Amazon F1 instances provide a few choices in clock speed when synthesizing FPGA designs. As a conservative estimate the default choice of 125Mhz is selected as we are not currently limited by a computation bottleneck. We determine that the current design utilizing 4 cycles per AND operation and running at 125Mhz has the potential to perform 160 Gbps of computations when being fully utilized with only 20% utilization exceeding the current Amazon F1 network bandwidth.

Comparing with the Araki et al. design running on 20 Xeon E5-2686 v4 cores, their theoretical MPC op./sec can reach a total of 10 billion operations, while our fully utilized FPGA design surpasses it by over 10×.

TABLE I: AWS Implementation Result Analysis

AND Cores	Bits	MPC (billions op.)/sec
1	128	2.67
3	384	8.00
12	1536	32.0
24	3072	64.0
48	6144	128
60	7680	160

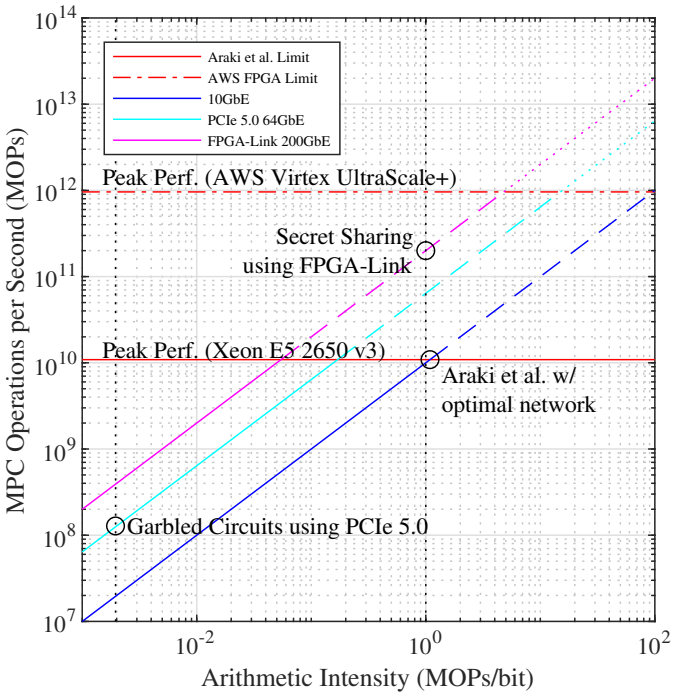


Fig. 4: Roofline model comparing Secret Sharing performance against data center bandwidth. We denote limitations in the Araki et al. design and our FPGA Secret Sharing implementation, and FPGA implementations based on Garbled Circuits.

B. Analysis

Based on the minimal data dependencies and flow in Figure 3, in principle all MPC Secret Sharing gates can execute one operation per clock cycle. This can be done by utilizing a fully pipelined implementation of the AND and multiply gate. Such a design would saturate a 10Gbps network connection when operated at 78.13 MHz. Operating at the higher frequencies used commonly by FPGAs would require higher bandwidth.

Our tests and analysis of different quantities of MPC blocks on Amazon AWS provide sufficient data points to establish that Secret Sharing MPC can be competitive when implemented on FPGA hardware in the data center.

The original work utilized a software implementation of the protocol executed on general purpose processors [1], [43]. Specifically, each party used varying numbers of cores from one or two Xeon processors. The authors were able to nearly saturate their 10Gbps link (7.38Gbps) between parties when

using all cores in each node. While the authors were limited by multiple cores causing queuing congestion at the Network Interface Card (NIC) the use of a CPU appears to have more limited scaling potential. Using the reported number of AES/sec and network communication for 1 core, scaling from 73.3% CPU usage to 100% would appear to show 1 core being capable of saturating a $\sim$0.780Gbps connection. Multiplying for 20 cores that would indicate a peak of $\sim$15.6Gbps.

In comparison, the FPGA block we tested for performing Secret Sharing only requires 3 AND cores to exceed the 7.38Gbps reached with 20 CPU cores, reaching a full 8.00Gbps. This uses $\sim$5% of the fabric available on the FPGA targeted, a $10\times$ improvement vs the CPU utilization. Attempting to fully employ the available fabric it is possible to implement 60 AND cores based on our design which would permit saturation of a 160Gbps link.

We demonstrate the peak performance obtainable using the roofline model in Figure 4. Figure 4 shows the limiting factors and constraints of Secret Sharing designs: compute bandwidth (horizontal lines) and network bandwidth (diagonal lines). We first show that the peak performance is demonstrated in number of MPC operations per second (MOPs), while arithmetic intensity (AI) is designated as MOPs/bit of data transferred over the network. We utilize the AND gate as our common Secret Sharing gate and assume a pipelined design of one MOP per cycle. Each Secret Sharing gate requires exactly one bit of communication thus an arithmetic intensity of 1 is achievable. Therefore the Araki et al. design has the potential to hit a peak performance of 10^{10} MOPs if they did not encounter any drawback due to network congestion. Comparatively we show that a fully utilized FPGA running all AND gates can outperform by $\sim$2 orders of magnitude. This analysis shows that the Araki et al system provides an almost perfect balance between network and MOPs when utilizing their 10Gbps interface. However, if a larger bandwidth network is available, such as 40 GbE, then their application will be limited by computational performance and thus will not effectively utilize the bandwidth available.

By contrast, using a fully pipelined version of our design, we show a theoretical peak performance of the FPGA accelerated Secret Sharing design able to achieve 10^{12} MOPs. This indicates that the design of Secret Sharing on FPGAs remains limited by the current bandwidth offered in the data center. This limitation allows for the extra FPGA resources to be distributed for other tasks such as more local computations of XOR and addition gates or control logic for high performance applications. We compare this against the arithmetic intensity of Garbled Circuits with the assumption that each garbling table can be created each clock cycle. Traditionally each MPC gate contains a table of 4 SHA hashes. We consider the arithmetic intensity (AI) of Garbled Circuits to be $1/(4 \times 128) = 0.002$, which when translated to the roofline model shows that Garbled Circuits are heavily network limited due to the low arithmetic intensity and high bandwidth requirements.

These results demonstrate preferable scaling properties supporting the selection of FPGAs for acceleration. Based on the results, targeting the anticipated 200Gbps links in the Amazon F1 would require less than 25% fabric utilization to reach full saturation utilizing the pipeline improvement described earlier. With a frequency improvement, adjustable through AWS, even less fabric would be required. The remaining available fabric is beneficial as it allows for work distribution, additional secure computations, and higher application control logic such as matrix multiply.

V. Related Work

There exists earlier research exploring hardware accelerated MPC, but the efforts have focused on Garbled Circuits rather than Secret Sharing. The hardware considered has included GPUs [19], [50]–[52]; most efforts, however, employ FPGAs. The earliest of these efforts dates to 2010 [17], [18], with more work recently [23]–[25], and some considering Amazon AWS [21], [25], [27]. Other work explored garbling entire processors [20], [22] and specialized problem acceleration [21].

The usage of MPC in data centers [24], [26] matches our decisions. Their architecture and usage of individual blocks for AND and XOR operations guided our choice to allow greater software control of the FPGA without the hardware needing to be recreated for each use.

VI. Conclusion

In this paper, we describe one approach to implementing all MPC gates described by Araki et al. [44] in hardware. We demonstrate the viability of Secret Sharing MPC in a low latency environment and test the design on an FPGA in the cloud highlighting greater potential scalability of the design compared to alternatives. With these insights, we plan to pursue improvements to this design to increase the performance further and to implement the higher level controls both in software and hardware to use the Secret Sharing building block in a complete MPC cloud service for higher applications and specifically for machine learning.

Some future work includes HDL implementation optimizations allowing for a fully pipelined scheme, FPGA to FPGA communication using future AWS offering or with an in-house design, and implementation of share conversion between both forms of Secret Sharing. Additional research directions include different viable MPC security models and hardware security considerations on FPGAs including but not limited to the usage of vendor specific random number generation, secure data at rest on FPGAs, and applying secure multi-party computation for machine learning applications.

VII. Acknowledgements

Supported by Red Hat and by NSF Grants 1618303, 1718135, 1739000 1915763, 1919130, 1925504, and 1931714. DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the United States Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

REFERENCES

[1] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 805–817. [Online]. Available: https://doi.org/10.1145/2976749.2978331

[2] D. Evans, V. Kolesnikov, and M. Rosulek, *A Pragmatic Introduction to Secure Multi-Party Computation*. NOW Publishers, 2018.

[3] A. C. Yao, "Protocols for Secure Computations." *Annual Symposium on Foundations of Computer Science - Proceedings*, pp. 160–164, 1982.

[4] A. C. C. Yao, "How To Generate and Exchange Secrets." *Annual Symposium on Foundations of Computer Science (Proceedings)*, no. 1, pp. 162–167, 1986.

[5] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[6] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From keys to databases - real-world applications of secure multi-party computation," *Comput. J.*, vol. 61, no. 12, pp. 1749–1771, 2018.

[7] T. Giannopoulos and D. Mouris, "Privacy preserving medical data analytics using secure multi party computation. an end-to-end use case." Ph.D. dissertation, National and Kapodistrian University of Athens, 09 2018.

[8] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, "Students and taxes: a privacy-preserving social study using secure computation," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1159, 2015.

[9] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean, "Secure computation of surveys," in *EU Workshop on Secure Multiparty Protocols*, 2004, pp. 2–14. [Online]. Available: https://www.cs.yale.edu/homes/jf/SMP2004.pdf

[10] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft, "Secure multiparty computation goes live," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, vol. 5628. Springer, 2009, pp. 325–343.

[11] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, "Confidential benchmarking based on multiparty computation," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, vol. 9603. Springer, 2016, pp. 169–187.

[12] A. Abidin, A. Aly, S. Cleemput, and M. A. Mustafa, "An mpc-based privacy-preserving protocol for a local electricity trading market," in *CANS*, ser. Lecture Notes in Computer Science, vol. 10052, 2016, pp. 615–625.

[13] 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 *ACM Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[14] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, M. Raykova, S. Saxena, K. Seth, D. Shanahan, and M. Yung, "On deploying secure computing commercially: Private intersection-sum protocols and their business applications," *IACR Cryptology ePrint Archive*, vol. 2019, p. 723, 2019.

[15] P. Mohassel and P. Rindal, "Aby^3: A mixed protocol framework for machine learning," in *ACM Conference on Computer and Communications Security*. ACM, 2018, pp. 35–52.

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

[17] K. Järvinen, V. Kolesnikov, A. R. Sadeghi, and T. Schneider, "Embedded SFE: Offloading server and network using hardware tokens," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6052 LNCS, pp. 207–221, 2010.

[18] ——, "Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6225 LNCS, pp. 383–397, 2010.

[19] T. K. Frederiksen, T. P. Jakobsen, and J. B. Nielsen, "Faster maliciously secure two-party computation using the GPU," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8642, no. grant 61061130540, pp. 358–379, 2014.

[20] E. M. Songhori, S. Zeitouni, G. Dessouky, T. Schneider, A. R. Sadeghi, and F. Koushanfar, "GarbledCPU: A MIPS processor for secure computation in hardware," *Proceedings - Design Automation Conference*, vol. 05-09-June, 2016.

[21] S. U. Hussain, B. D. Rouhani, M. Ghasemzadeh, and F. Koushanfar, "MAXelerator: FPGA Accelerator for Privacy Preserving Multiply-Accumulate (MAC) on Cloud Servers," *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[22] E. M. Songhori, M. S. Riazi, S. U. Hussain, A. R. Sadeghi, and F. Koushanfar, "ARM2GC: Succinct garbled processor for secure computation," *Proceedings - Design Automation Conference*, 2019.

[23] S. U. Hussain and F. Koushanfar, "FASE: FPGA acceleration of secure function evaluation," *Proceedings - 27th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019*, pp. 280–288, 2019.

[24] X. Fang, S. Ioannidis, and M. Leeser, "Secure function evaluation using an FPGA overlay architecture," *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 257–266, 2017.

[25] ——, "SIFO: Secure computational infrastructure using FPGA overlays," *International Journal of Reconfigurable Computing*, vol. 2019, 2019.

[26] K. Huang, M. Gungor, X. Fang, S. Ioannidis, and M. Leeser, "Garbled circuits in the cloud using FPGA enabled nodes," *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019*, pp. 1–6, 2019.

[27] M. Leeser, M. Gungor, K. Huang, and S. Ioannidis, "Accelerating large garbled circuits on an FPGA-enabled cloud," *Proceedings of H2RC 2019: 5th International Workshop on Heterogeneous High-Performance Reconfigurable Computing - Held in conjunction with SC 2019: The International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 19–25, 2019.

[28] P.-F. Wolfe, R. Patel, R. Munafo, M. Varia, and M. Herbordt, "Secret Sharing MPC on FPGAs in the Datacenter," in *IEEE Conference on Field Programmable Logic and Applications*, 2020.

[29] J. Sheng, C. Yang, and M. Herbordt, "Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study," in *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, 2015, https://pdfs.semanticscholar.org/832d/c69145f5ba0ed6a951583201b1b20dd2096e.pdf.

[30] A. George, M. Herbordt, H. Lam, A. Lawande, J. Sheng, and C. Yang, "Novo-G#: A Community Resource for Exploring Large-Scale Reconfigurable Computing Through Direct and Programmable Interconnects," in *2016 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA*, 2016, pp. 1–7, doi: 10.1109/ HPEC.2016.7761639.

[31] C. Plessl, "Bringing FPGAs to HPC Production Systems and Codes," in *H2RC'18 workshop at Supercomputing (SC'18)*, 2018, doi: 10.13140/RG.2.2.34327.42407.

[32] A. Mondigo, T. Ueno, K. Sano, and H. Takizawa, "Comparison of Direct and Indirect Networks for High-Performance FPGA Clusters," in *ARC 2020. Lecture Notes in Computer Science, vol 12083*, F. Rincon, J. Barba, H. So, P. Diniz, and J. Caba, Eds. Springer, 2020, 10.1007/978-3-030-44534-8_24.

[33] B. Ringlein, F. Abel, A. Ditter, B. Weiss, C. Hagleitner, and D. Fey, "System architecture for network-attached fpgas in the cloud using partial reconfiguration," in *IEEE Conference on Field Programmable Logic and Applications*, 2019.

[34] V. Krishnan, O. Serres, and M. Blocksome, "COnfigurable Network Protocol Accelerator (COPA)," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020.

[35] A. Putnam, et al., "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *International Symposium on Computer Architecture*, 2014, pp. 13–24, doi: 10.1109/ISCA.2014.6853195.

[36] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *49th IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–13.

[37] J. Sheng, C. Yang, A. Caulfield, M. Papamichael, and M. Herbordt, "HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics," in *27th International Conference on Field Programmable Logic and Applications*, 2017, doi: 10.23919/ FPL.2017.8056853.

[38] D. H. U. Beaver, S. M. Micali, and P. M. Rogaway, "The Round Complexity of Secure Protocols," *ACM*, 1990.

[39] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," *ACM International Conference Proceeding Series*, pp. 129–139, 1999.

[40] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5126 LNCS, no. PART 2, pp. 486–498, 2008.

[41] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole - reducing data transfer in garbled circuits using half gates," in *EUROCRYPT (2)*, ser. Lecture Notes in Computer Science, vol. 9057. Springer, 2015, pp. 220–250.

[42] S. Yakoubov, "A Gentle Introduction to Yao's Garbled Circuits," 2017, http://web.mit.edu/sonka89/www/papers/2017ygc.pdf.

[43] T. Araki, A. Barak, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "Demo: High-throughput secure three-party computation of kerberos ticket generation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1841–1843. [Online]. Available: https://doi.org/10.1145/2976749.2989035

[44] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein, "Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate per Second Barrier," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 843–862, 2017.

[45] Furukawa, Jun and Lindell, Yehuda and Nof, Ariel and Weinstein, Or, "High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority," in *Advances in Cryptology – EUROCRYPT 2017*, Coron, Jean-Sébastien and Nielsen, Jesper Buus, Ed. Cham: Springer International Publishing, 2017, pp. 225–255.

[46] J. Furukawa and Y. Lindell, "Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1557–1571, 2019.

[47] H. Hsing, "tiny_aes," https://opencores.org/projects/tiny_aes, 2012. [Online]. Available: https://opencores.org/ocsvn/tiny_aes/tiny_aes/trunk

[48] J. Castillo, "systemc_rng," https://opencores.org/projects/systemc_rng, 2004. [Online]. Available: https://opencores.org/ocsvn/systemc_rng/systemc_rng/trunk

[49] Amazon Web Services, "aws_fpga," https://github.com/aws/aws-fpga, 2016. [Online]. Available: https://github.com/aws/aws-fpga.git

[50] S. Pu, P. Duan, and J.-C. Liu, "Fastplay-A Parallelization Model and Implementation of SMC on CUDA based GPU Cluster Architecture," *IACR Cryptology ePrint Archive*, vol. 2011, p. 97, 2011.

[51] S. Pu and J. Liu, "Computing Privacy-Preserving Edit Distance and Smith-Waterman Problems on the GPU Architecture." *IACR Cryptology ePrint Archive*, 2013. [Online]. Available: http://eprint.iacr.org/2013/204.pdf

[52] N. Husted, S. Myers, A. Shelat, and P. Grubbs, "GPU and CPU parallelization of honest-but-curious secure two-party computation," *ACM International Conference Proceeding Series*, pp. 169–178, 2013.