# ABY2.0: New Efficient Primitives for STPC with Applications to Privacy in Machine Learning (Extended Abstract)*

**Arpita Patra, Ajith Suresh**
Indian Institute of Science

**Thomas Schneider, Hossein Yalame**
TU Darmstadt

## Abstract

In this work, we improve semi-honest secure two-party computation (STPC) over rings, specially for privacy-preserving machine learning, with a focus on the efficiency of the online phase. We construct efficient protocols for several privacy-preserving machine learning (PPML) primitives such as scalar product, matrix multiplication, ReLU, and maxpool. The online communication of our scalar product is two ring elements *irrespective* of the vector dimension, which is a feature achieved for the first time in PPML literature. We implement and benchmark training and inference of Logistic Regression and Neural Networks over LAN and WAN networks. For training, we improve online runtime (both for LAN and WAN) over SecureML (Mohassel et al., IEEE S&P'17) in the range $1.5\times$–$6.1\times$, while for inference, the improvements are in the range of $2.5\times$–$754.3\times$.

## 1 Introduction

Secure Multi-Party Computation (SMPC) [2–4] allows $n$ mutually distrusting parties to jointly compute a function on their private inputs. The computation guarantees i) privacy– no set of $t$ corrupt parties can learn more information than the output, and ii) correctness– corrupt parties cannot force others to accept a wrong output. Due to its immense potential, SMPC can be used for solving real-life applications such as privacy-preserving auctions [5] and remote diagnostics [6], secure genome analysis [7, 8], and recently in the domain of privacy-preserving machine learning (PPML) [9–19].

While SMPC has the potential to address the privacy issues that arise in PPML, it involves multiple rounds of interaction between the parties. This work focuses on reducing round and communication complexity of secure two-party computation (STPC) [20] with mixed protocols over rings.

**Our Contributions.** We propose, ABY2.0, an efficient mixed-protocol STPC framework for PPML primitives. Our protocols are secure against a *semi-honest* adversary and use an *input-independent* setup [21, 22, 14, 23, 16, 15, 18]. We build several PPML building blocks with the focus on online efficiency. Our contributions can be summed up as follows:

**STPC protocols and conversions:** We propose an efficient STPC protocol over $\ell$-bit rings, requiring a communication of just 2 ring elements per multiplication in the online phase. Moreover, our protocol helps in realising efficient PPML primitives. For an $N$-input multiplication gate, our solution has a *constant* cost of 2 ring elements and one round of interaction. This is a massive improvement over [24], where they require communication of $2N$ ring elements. Round complexity wise, the naive method of multiplying $N$ elements by taking two at a time requires $\log_2(N)$ online rounds and overall communication of $4(N-1)$ ring elements for [20] and $2(N-1)$ for [22]. The mixed world conversions, that enable easy transition between Arithmetic ( [3]), Boolean ( [3]) and Yao ( [2]) sharing, are now celebrated in the PPML literature [9, 11, 10] due to their potential in building

*The full version of this paper published at USENIX'21 [1].

practically-efficient protocols. We propose a new set of conversions that outperform the state-of-the-art conversions of ABY[20] and SecureML [9] in the online phase. Our solution reduces the number of online rounds from 2 to 1 for the conversions.

**Building Blocks:** We propose efficient constructions for widely-used PPML building blocks that include Scalar Product, Depth-Optimized Circuits, Matrix Multiplication, Non-linear Activation functions, and Maxpool. The highlights include:

– Scalar Product: Our new protocol incurs an online communication that is *independent* of the vector dimension $\mathsf{n}$. This feature is achieved for the *first time* in the PPML literature. We require communication of just 2 ring elements as opposed to $4\mathsf{n}$ elements of [20, 9].

– Matrix Multiplication: Matrix multiplication is the fundamental building block in most ML algorithms. We extend the STPC multiplication protocol to support vector operations and provide an efficient matrix multiplication protocol.

– Depth Optimized Circuits: The Parallel Prefix Adder (PPA) [25, 26] used in the recent PPML literature [27] incurs a multiplicative depth of $\log_2(\ell)$ since it uses two-input AND gates only. We propose round efficient PPA constructions using a combination of two, three, and four input AND gates. Our solution has $2\times$ fewer rounds and less online communication compared to [27].

– Maximum of three elements: Our new protocol improves the online communication of [24] by $14\times$ while reducing the online rounds from 5 to 4.

**Application:** We implement the training and inference of Logistic Regression and Neural Networks in a LAN and a WAN setting and benchmarked over datasets with various feature sizes. For training, we obtain online runtime improvements over SecureML [9] in the range $2.7\times$–$6.1\times$ for LAN and $1.5\times$–$2.8\times$ for WAN. Our improvement for inference ranges from $7.9\times$–$754.3\times$ for LAN, while it ranges from $2.5\times$–$753.2\times$ for WAN.

**Beaver's technique [28] on gate inputs (cf. left of Fig. 1).** In STPC and PPML literature, there has been a lot of works [9–11] that use Beaver's[28] circuit randomization technique to compute the product $\mathsf{a} \cdot \mathsf{b}$. In this technique (cf. left side of Fig. 1), the inputs of the multiplication gate are randomized first and the corresponding correlated randomness is generated independently (preferably in a setup phase). In detail, parties interactively generate an additive sharing of the multiplication triple $(\delta_\mathsf{a}, \delta_\mathsf{b}, \delta_\mathsf{ab})$ with $\delta_\mathsf{ab} = \delta_\mathsf{a}\delta_\mathsf{b}$ during the setup phase before the actual inputs are known. Now, we can write: $\mathsf{a}\cdot\mathsf{b} = ((\mathsf{a}+\delta_\mathsf{a})-\delta_\mathsf{a})((\mathsf{b}+\delta_\mathsf{b})-\delta_\mathsf{b}) = (\mathsf{a}+\delta_\mathsf{a})(\mathsf{b}+\delta_\mathsf{b}) - (\mathsf{a}+\delta_\mathsf{a})\delta_\mathsf{b} - (\mathsf{b}+\delta_\mathsf{b})\delta_\mathsf{a} + \delta_\mathsf{ab}$.

Let $\Delta_\mathsf{a} = (\mathsf{a} + \delta_\mathsf{a})$ and $\Delta_\mathsf{b} = (\mathsf{b} + \delta_\mathsf{b})$ be the randomized versions of the input values of a multiplication gate. Then, during the online phase, parties locally compute an additive sharing of $\Delta_\mathsf{a}$ using additive shares of $\mathsf{a}$ and $\delta_\mathsf{a}$. Similarly, an additive sharing of $\Delta_\mathsf{b}$ is computed. This is followed by the parties mutually exchanging the shares of $\Delta_\mathsf{a}$ and $\Delta_\mathsf{b}$ to enable public reconstruction of $\Delta_\mathsf{a}$ and $\Delta_\mathsf{b}$. Then using the above equation, parties can locally compute a sharing of $\mathsf{a} \cdot \mathsf{b}$. Note that this method requires communicating 4 elements per multiplication (2 elements per reconstruction).

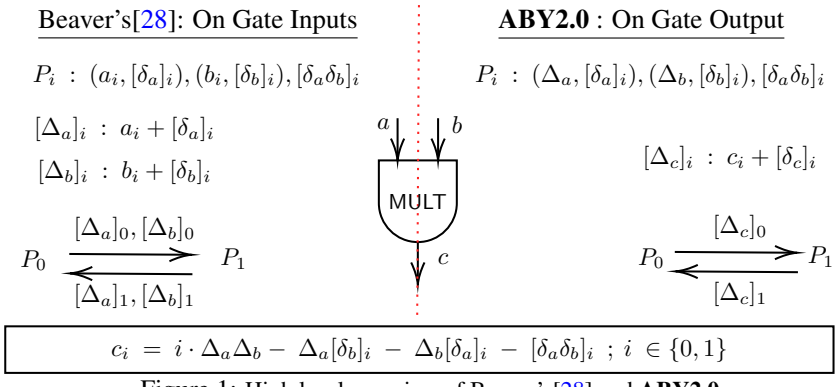


Figure 1: High level overview of Beaver's[28] and **ABY2.0**

**Our technique on gate outputs (cf. right of Fig. 1).** With this insight, we modify the sharing semantics so that the parties are ensured to have the $\Delta$ value as a part of their share, corresponding to every wire value (including the inputs of a multiplication gate). As a result, the reconstructions of $\Delta_\mathsf{a}$ and $\Delta_\mathsf{b}$ are no longer required. Our technique, in summary, shifts the need of reconstruction (which alone causes communication for a multiplication gate) from per input wire to the *output*

wire alone for a multiplication gate. For a traditional 2-input multiplication gate, we reduce the number of reconstructions (each involves sending 2 elements) from 2 to 1. As a result, we improve communication by a factor of $2\times$. The impact is much higher for an $N$-input multiplication gate and a scalar product of two $N$-dimensional vectors. For scalar product, Beaver's circuit re-randomization required $2N$ reconstructions, whereas our techniques need a *single* one, offering a gain of $2N\times$.

## 2 Protocols

**$[\cdot]$-sharing.** A value $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ is said to be $[\cdot]$-shared among $\mathcal{P}$, if party $P_i$ for $i \in \{0,1\}$ holds $[\mathsf{v}]_i$ such that $\mathsf{v} = [\mathsf{v}]_0 + [\mathsf{v}]_1$.

**$\langle\cdot\rangle$-sharing.** A value $\mathsf{v} \in \mathbb{Z}_{2^\ell}$ is said to be $\langle\cdot\rangle$-shared among $\mathcal{P}$, if there exist values $\delta_\mathsf{v}, \Delta_\mathsf{v} \in \mathbb{Z}_{2^\ell}$ such that i) $\delta_\mathsf{v}$ is $[\cdot]$-shared among $P_0, P_1$, ii) $\Delta_\mathsf{v} = \mathsf{v} + \delta_\mathsf{v}$, and iii) $\Delta_\mathsf{v}$ is known to both $P_0, P_1$ in clear. We denote the shares of individual parties as $\langle\mathsf{v}\rangle_i = ([\delta_\mathsf{v}]_i, \Delta_\mathsf{v})$ for $i \in \{0,1\}$.

**Linear Operations.** Our sharing scheme is linear in the sense that given $\langle\mathsf{a}\rangle, \langle\mathsf{b}\rangle$ and public constants $c_1, c_2$, parties can locally compute $\langle\mathsf{y}\rangle = c_1 \cdot \langle\mathsf{a}\rangle + c_2 \cdot \langle\mathsf{b}\rangle$.

**Multiplication Protocol.** Given the $\langle\cdot\rangle$-sharing of $\mathsf{a}, \mathsf{b}$, the goal of protocol $\mathsf{MULT}$ (cf. [1, Fig. 2]) is to generate $\langle\mathsf{y}\rangle$ where $\mathsf{y} = \mathsf{ab}$. To summarize, during the setup phase, parties first locally sample the $[\cdot]$-shares for $\delta_\mathsf{y}$. In parallel, parties execute the setup phase, using Oblivious Transfer (OT) [20, 29] or Homomorphic Encryption (HE) [30, 21, 31], on $[\delta_\mathsf{a}]$ and $[\delta_\mathsf{b}]$ to obtain $[\delta_\mathsf{ab}]$. During the online phase, the parties locally compute $[\Delta_\mathsf{y}]$ and subsequently reconstruct $\Delta_\mathsf{y}$.

**Multi-Input Multiplication gate.** We show how to compute a 3-input multiplication gate ($\mathsf{MULT3}$ ) with three inputs $\mathsf{a}, \mathsf{b}, \mathsf{c}$ with each input being $\langle\cdot\rangle$-shared (cf. [1, Fig. 5]). Here we need to generate the $[\cdot]$-sharing of four terms, namely $\delta_\mathsf{ab}, \delta_\mathsf{bc}, \delta_\mathsf{ac}$ and $\delta_\mathsf{abc}$ during the setup phase.

## 3 Primitives for PPML and Evaluation

**Scalar Product.** Given the arithmetic sharing of $\mathsf{n}$-element vectors $\vec{\mathbf{a}}, \vec{\mathbf{b}}$, the goal is to generate $\langle\mathsf{y}\rangle^{\mathbf{A}}$ where $\mathsf{y} = \vec{\mathbf{a}} \odot \vec{\mathbf{b}} = \sum_{j=1}^{\mathsf{n}} \mathsf{a}_i\mathsf{b}_i$. The parties first execute the preprocessing corresponding to each of the $\mathsf{n}$ multiplications in parallel. During the online phase, parties first locally compute the $[\cdot]$-sharing of value $\Delta_{\mathsf{y}_j}$ where $\mathsf{y}_j$ denotes $\mathsf{a}_j\mathsf{b}_j$. $P_i$ for $i \in \{0,1\}$ now locally computes $[\Delta_\mathsf{y}]_i = \sum_{j=1}^{\mathsf{n}} [\Delta_{\mathsf{y}_j}]_i$. This is followed by the parties mutually exchanging $[\Delta_\mathsf{y}]$-shares to reconstruct $\Delta_\mathsf{y}$. Compared with the state-of-the-art 2PC solutions in ABY [20] and SecureML [9] which require communication of $4\mathsf{n}$ elements in the online phase, our protocol requires an online communication of just 2 ring elements.

**Depth-optimized Adder.** We design a depth optimized adder and bit extraction circuit using two, three, and four input AND gates combined. Concretely, for a 64-bit ring, we achieve a $2\times$ improvement in depth over existing designs along with a reduction in online communication. **Matrix Multiplication.** We construct efficient protocol for matrix multiplication. Given two matrices $\mathbf{A}^{p\times q}, \mathbf{B}^{q\times r}$, our protocol improved the online communication from $\mathrm{O}(pqr)$ to $\mathrm{O}(pr)$ ring elements, eliminating the dependency on dimension $q$.

**ReLU.** To compute $\mathsf{ReLU}(\mathsf{v}) = \mathsf{max}(0, \mathsf{v})$, parties first execute the $\mathsf{LT}$ protocol on $\mathsf{v}$ to obtain $\langle\mathsf{u}\rangle^{\mathbf{B}}$, where $\mathsf{u} = 1$ if $\mathsf{v} < 0$ and 0 otherwise. Parties can then locally compute $\langle\overline{\mathsf{u}}\rangle^{\mathbf{B}}$, followed by executing an special conversion ($\langle\mathsf{a}\rangle^{\mathbf{B}}\langle\mathsf{b}\rangle^{\mathbf{A}} \rightarrow \langle\mathsf{ab}\rangle^{\mathbf{A}}$) on $\langle\overline{\mathsf{u}}\rangle^{\mathbf{B}}$ and $\langle\mathsf{v}\rangle^{\mathbf{A}}$ to obtain the desired result.

**Maxpool.** We construct maxpool building block using our new protocols. Here we provide an

| Ref. | Type | $\mathsf{n} = 1{,}024$ | | $\mathsf{n} = 65{,}536$ | |
|---|---|---|---|---|---|
| | | Comm [KB] | Rounds | Comm [KB] | Rounds |
| [9] | GC | 2,056 | 4 | 131,584 | 4 |
| **ABY2.0** | GC | 1,024 | **2** | 65,536 | **2** |
| [24] | MAX2 | 258 | 50 | 16,512 | 80 |
| **ABY2.0** | MAX2 | **53** | 40 | **3,408** | 64 |
| [24] | MAX3 | 492 | 35 | 31,679 | 55 |
| **ABY2.0** | MAX3 | 63 | 28 | 4,080 | 44 |

Table 1: Online communication and rounds of Maxpool protocols. $\mathsf{n}$ is the number of input elements.

empirical analysis of our Maxpool protocol and compare it with its competitors. We consider vectors with dimensions $\mathsf{n} \in \{1024, 65536\}$. We have evaluated both round-optimized and communication-optimized variants of the Maxpool protocol. In the round-optimized variant proposed by SecureML[9], a garbled circuit is used to evaluate the maximum among $\mathsf{n}$ elements. This method requires converting Arithmetic shares to Yao shares and back, which can be tackled using A2Y and Y2A conversions. In the communication-optimized variant, we use the tree-based approach where either two (MAX2) or three elements (MAX3) are compared at a time. Table 1 summarizes the cost for the online phase of the Maxpool protocol. It is evident from the table that our protocols outperform [9, 24] in both communication and rounds for the online phase in all three cases.

**Evaluation:** We implemented our protocols using the ENCRYPTO library[32] in C++17 over a 64-bit ring. In the domain of PPML [9, 27, 14, 15], we show that Logistic Regression and Neural Networks can be substantially improved with our building blocks. For the training phase, we follow [27, 15] and benchmark the *number of iterations per minute* (#it/min) over both LAN and WAN. The values are reported over batch sizes of $\{128, 256, 512\}$ and with feature sizes $\mathsf{n} \in \{100, 900\}$. For the inference, we report the online runtime as well as the *throughput* (TP) for the aforementioned feature sizes.

**Logistic Regression.** Tab. 2 gives our benchmarks for Logistic Regression training. Over SecureML[9], we have improvements in the range $4.4\times$-$6.1\times$ for LAN and in the range $1.5\times$-$2.0\times$ for WAN. Note that over WAN, the throughput of our protocol remains unchanged across feature sizes as well as batch sizes. This discrepancy is due to the effect of communication on the rtt. Tab. 3 gives our benchmarks for Logistic Regression inference. We improve the online runtime over SecureML [9] by $5.5\times$ for LAN and $1.6\times$ for WAN, and the online throughput by $7.9\times$-$35.5\times$ in LAN and $2.5\times$-$11.1\times$ in WAN.

**Neural Networks (NN).** In our work, we follow previous works [9, 27, 14] and consider a Neural Network with two hidden layers, each having 128 nodes followed by an output layer of 10 nodes. We use ReLU as the activation function over the nodes. Moreover, for training we use the MPC-friendly variant of the softmax function[9] which is defined as $f(\mathsf{v}_i) = \mathsf{ReLU}(\mathsf{v}_i)/\sum_{j=1}^{m} \mathsf{ReLU}(\mathsf{v}_j)$. Tab.2 gives our benchmarks for NN Training. Over SecureML[9], we have improvements in the range $2.7\times$-$3.46\times$ for LAN and $2.4\times$-$2.8\times$ for WAN.

| Batch Size | Ref. | Logistic Regression Training | | | | NN Training | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LAN (#it/min) | | WAN (#it/min) | | LAN (#it/min) | | WAN (#it/min) | |
| | | $\mathsf{n}=100$ | $\mathsf{n}=900$ | $\mathsf{n}=100$ | $\mathsf{n}=900$ | $\mathsf{n}=100$ | $\mathsf{n}=900$ | $\mathsf{n}=100$ | $\mathsf{n}=900$ |
| 128 | [9] | 29,112 | 27,273 | 108 | 104 | 3,593 | 3,559 | 17 | 17 |
| | **ABY2.0** | **176,471** | **149,626** | **162** | **162** | **12,448** | **12,343** | **42** | **42** |
| 256 | [9] | 25,829 | 24,058 | 107 | 97 | 3,578 | 3,521 | 17 | 17 |
| | **ABY2.0** | **163,043** | **117,188** | **162** | **162** | **9,259** | **9,156** | **42** | **42** |
| 512 | [9] | 23,292 | 22,247 | 104 | 83 | 3,330 | 3,323 | 15 | 15 |
| | **ABY2.0** | **110,906** | **98,847** | **162** | **162** | **9,177** | **9,146** | **42** | **42** |

Table 2: Comparison of the online throughput of ABY2.0 and SecureML [9] for Logistic Regression Training and NN Training. Best results in bold and larger is better. $\mathsf{n}$ is the number of features.

| Parameter | Ref. | Logistic Regression Inference | | | | NN Inference | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LAN | | WAN | | LAN | | WAN | |
| | | $\mathsf{n}=100$ | $\mathsf{n}=900$ | $\mathsf{n}=100$ | $\mathsf{n}=900$ | $\mathsf{n}=100$ | $\mathsf{n}=900$ | $\mathsf{n}=100$ | $\mathsf{n}=900$ |
| Runtime (ms) | [9] | 1.60 | 1.69 | 496.08 | 504.96 | 8.68 | 8.77 | 1,759.92 | 1,759.95 |
| | **ABY2.0** | **0.29** | **0.29** | **308.16** | **308.16** | **2.66** | **2.66** | **744.12** | **744.12** |
| TP (queries/min) | [9] | 5,342.61 | 1,193.01 | 16.08 | 3.58 | 62.02 | 40.89 | 0.19 | 0.12 |
| | **ABY2.0** | **42,372.41** | **42,371.11** | **39.88** | **39.88** | **30,796.99** | **30,795.17** | **92.39** | **91.57** |

Table 3: Comparison of the online runtime and throughput of ABY2.0 and SecureML[9] for Logistic Regression Inference and NN Inference. Best results in bold. $\mathsf{n}$ is the number of features.

Tab. 3 gives our benchmarks for NN Inference. Here we improve the online runtime of SecureML [9] by a factor of $3.3\times$ in LAN and $2.4\times$ in WAN. Regarding the online throughput, we observe huge improvements in the range $496\times$–$754\times$ for both LAN and WAN. This improvement is primarily due to our efficient dot product protocol which has a dimension-independent online communication.

**Setup Costs for PPML.** We incur a minimal overhead of just $1.6\%$ over SecureML [9] in terms of communication in the setup phase for Logistic Regression, while the overhead is $0.7\%$ for the case of Neural Networks. The overhead results from the expensive communication required by our activation functions (Sigmoid and ReLU) over the garbled circuit based solutions of SecureML [9].

## References


[1] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2. 0: Improved mixed-protocol secure two-party computation," in *USENIX Security*, 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. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *STOC*, 1988.

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

[6] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel, "Privacy-preserving remote diagnostics," in *CCS*, 2007.

[7] M. Blanton and F. Bayatbabolghani, "Efficient server-aided secure two-party function evaluation with applications to genomic computation," in *PETS*, 2016.

[8] O. Tkachenko, C. Weinert, T. Schneider, and K. Hamacher, "Large-scale privacy-preserving statistical computations for distributed genome-wide association studies," in *ASIACCS*, 2018.

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

[10] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *ASIACCS*, 2018.

[11] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *USENIX Security*, 2018.

[12] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint:1803.05961*, 2018.

[13] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *IEEE S&P*, 2019.

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

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

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

[17] A. Patra and A. Suresh, "BLAZE: Blazing Fast Privacy-Preserving Machine Learning," in *NDSS*, 2020.

[18] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: super-fast and robust privacy-preserving machine learning," *CoRR*, vol. abs/2005.10296, 2020. [Online]. Available: https://arxiv.org/abs/2005.10296

[19] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference," in *ARES*, 2020.

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

[21] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO*, 2012.

[22] A. Ben-Efraim, M. Nielsen, and E. Omri, "Turbospeedz: Double your online SPDZ! improving SPDZ using function dependent preprocessing," in *ACNS*, 2019.

[23] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider, "HyCC: Compilation of hybrid protocols for practical secure computation," in *CCS*, 2018.

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

[25] A. Beaumont-Smith and C. . Lim, "Parallel Prefix Adder Design," in *15th IEEE Symposium on Computer Arithmetic*, 2001.

[26] D. M. Harris, "A taxonomy of parallel prefix networks," in *Asilomar Conference on Signals, Systems and Computers*, 2003.

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

[28] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[29] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster malicious arithmetic secure computation with oblivious transfer," in *CCS*, 2016.

[30] W. Henecka, S. Kögl, A. R. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: Tool for automating secure two-party computations," in *CCS*, 2010.

[31] D. Rathee, T. Schneider, and K. Shukla, "Improved multiplication triple generation over rings via RLWE-based AHE," in *CANS*, 2019.

[32] ENCRYPTO Utils, https://github.com/encryptogroup/ENCRYPTO_utils.