Tetrad: Actively Secure 4PC for Secure Training and Inference

Nishat Koti
Indian Institute of Science, Bangalore

Arpita Patra
Indian Institute of Science, Bangalore

Rahul Rachuri
Aarhus University, Denmark

Ajith Suresh
Indian Institute of Science, Bangalore

1 INTRODUCTION

Increased concerns about privacy coupled with policies such as European Union General Data Protection Regulation (GDPR) make it harder for multiple parties to collaborate on machine learning computations. The emerging field of privacy-preserving machine learning (PPML) addresses this issue by offering tools to let parties perform computations without sacrificing privacy of the underlying data. PPML can be deployed across various domains such as healthcare, recommendation systems, text translation, etc., with works like [2] demonstrating practicality.

One of the main ways in which PPML is realised is through the paradigm of secure outsourced computation (SOC). Clients can outsource the training/prediction computation to powerful servers available on a 'pay-per-use' basis from cloud service providers. Of late, secure multiparty computation (MPC) based techniques [5, 8, 9, 26, 29, 30, 33, 34, 37] have been gaining interest, where a server enacts the role of a party in the MPC protocol. MPC [17, 39] allows mutually distrusting parties to compute a function in a distributed fashion while guaranteeing *privacy* of the parties' inputs and *correctness* of their outputs against any coalition of t parties.

The goal of PPML is practical deployment, making *efficiency* a primary consideration. Functions such as comparison, activation functions (e.g., ReLU), are heavily used in machine learning. Instantiating these functions via MPC naively turns out to be prohibitively inefficient due to their non-linearity. Hence, there is motivation to design specialised protocols that can compute these functions efficiently. We work towards this goal in the 4-party (4PC) setting, assuming honest majority [5, 9, 18, 21]. 4PC is interesting because it buys us the following over 3PC (which is threshold optimal): (1) *independence from broadcast:* broadcast can be achieved by a simple protocol in which the sender sends to everyone and residual parties exchange and apply a majority rule (2) *efficient dot-product:* 4PC offers a more efficient dot-product protocol (which is an important building block for several ML algorithms) with communication complexity independent of feature size (3) *simplicity and efficiency:* protocols are vastly more efficient and simpler in terms of design. To enhance practical efficiency, many recent works [9, 13, 20, 33] resort to the preprocessing paradigm, which splits the computation into two phases; a preprocessing phase where input-independent (but function-dependent) computationally heavy tasks can be computed, followed by a fast online phase. Since the same functions in ML are evaluated several times, this paradigm naturally fits the case of PPML, where the ML algorithm is known beforehand. Further, recent works [12–14] propose MPC protocols over 32 or 64 bit rings to leverage CPU optimizations.

MPC protocols can be categorized as high-throughput [1, 3, 4, 8, 9, 16, 21, 29, 32, 33] and low-latency [6, 7], where the former, based on secret-sharing, requires less communication compared to the latter (garbled circuits). High-throughput protocols typically work over the boolean ring $\mathbb{Z}_2$ or an arithmetic ring $\mathbb{Z}_{2^\ell}$ and aim to minimize communication overhead (bandwidth) at the expense of non-constant rounds. While high-throughput protocols enable efficient computation of functions such as addition, multiplication and dot-product, other functions such as division are best performed using garbled circuits. Activation functions such as ReLU used in neural networks (NN) alternate between multiplication and comparison, wherein multiplication is better suited to the arithmetic world and comparison to the boolean world. Hence, MPC protocols working over different representations (arithmetic/boolean/garbled circuit based) can be mixed to achieve better efficiency. This provided motivation for mixed protocols where each subprotocol is executed in a world where it performs best. Mixed-protocol frameworks [9, 14, 15, 29, 30, 32, 34, 35] have support for efficient ways to switch between the worlds, thereby getting the best from each of them. This work proposes a mixed-protocol PPML framework via MPC with four parties and honest majority with active security.

Works such as [27, 29, 37] typically go for active security with abort, where the adversary can act maliciously to obtain the output and make honest parties abort. The stronger notion of fairness guarantees that either all or none of the parties obtain the output. This incentivizes the adversary to behave honestly in resource-expensive tasks such as PPML, as causing an abort will waste its resources. Trident [9] showed that fairness can be achieved at the cost of security with abort. In cases where the risk of failure of the system is too high, for instance, when deploying PPML for healthcare applications, participants might want to avoid the case when none of them receive the output. The way to tackle this issue is to modify protocols to guarantee that the correct output is always delivered to the participants irrespective of an adversary's misbehaviour. This is provided by guaranteed output delivery (GOD) or robustness. A robust protocol prevents the adversary from repeatedly causing the computations to rerun, thereby upholding the trust in the system. We propose two variants of the framework – one with fairness and the other with robustness. Our contributions are listed next.

2 OUR CONTRIBUTIONS

We make several contributions towards designing a practically efficient 4PC mixed-protocol framework, Tetrad [22], tolerating at most one active corruption. It operates over the ring $\mathbb{Z}_{2^\ell}$ and provides *end-to-end* conversions to switch between arithmetic, boolean and garbled worlds. We assume a one-time key setup phase and work in the (function-dependent) preprocessing model which paves the way for a fast online phase.

Depending on the sensitivity of the application and the underlying data, one might want different levels of security. For this, we propose two variants of the framework, covering fairness (Tetrad)

# Parties	Reference[a]	#Active Parties[b]	Security	Dot Product[c]		Dot Product with Truncation		Conversions[d]		
				Comm_{pre}[e]	Comm_{on}	Comm_{pre}	Comm_{on}	A	B	G
3	ABY3 [29]	3	Abort	$12d\ell$	$9d\ell$	$12d\ell + 84\ell$	$9d\ell + 3\ell$	✓	✓	✓
	BLAZE [33]	2	Fair	3ℓ	3ℓ	15ℓ	3ℓ	✓	✓	✗
	SWIFT (3PC) [21]	2	Robust	3ℓ	3ℓ	15ℓ	3ℓ	✓	✓	✗
4	Mazloom et al. [27]	4	Abort	2ℓ	4ℓ	2ℓ	4ℓ	✓	✓	✗
	Trident [9]	3	Fair	3ℓ	3ℓ	6ℓ	3ℓ	✓	✓	✓
	Tetrad [22]	2	Fair	2ℓ	3ℓ	2ℓ	3ℓ	✓	✓	✓
	SWIFT (4PC) [21]	2	Robust	3ℓ	3ℓ	4ℓ	3ℓ	✓	✓	✗
	Fantastic Four [11] (Best)	4	Robust	-	6ℓ	ℓ	9ℓ	✓	✓	✗
	Fantastic Four [11] (Worst)	3	Robust	-	$6(\ell + \kappa)$	$\approx 80\ell + 76\kappa$	$9\ell + 6\kappa$	✓	✓	✗
	Tetrad-R [22]	2	Robust	2ℓ	3ℓ	2ℓ	3ℓ	✓	✓	✓

[a] Amortized costs reported for 1 million operations [b] parties that carry out most of the computation during online phase [c] ℓ - size of ring in bits, κ - security parameter, d - length of vectors. [d] A, B, G: support for arithmetic, boolean, garbled worlds [e] 'Comm' - communication, 'pre' - preprocessing, 'on' - online

Table 1: Comparison of actively-secure MPC frameworks (3PC and 4PC) for PPML.

and robustness (Tetrad-R) guarantees. The fair variant improves upon the state-of-the-art *fair* framework of Trident [9]. Tetrad-R improves communication over the best robust protocols [11, 21], while also providing support for secure training of neural networks.

2.1 Improved Arithmetic/Boolean 4PC

In Tetrad, the multiplication protocol has a communication cost of only 5 ring elements as opposed to 6 in the state-of-the-art framework of Trident [9]. Security is elevated to robustness via Tetrad-R, which has a minimal overhead over the fair one, in the preprocessing. Concretely, for a 64-bit ring with 40-bit statistical security, the overhead per multiplication is 0.027 bits for a circuit containing 2^{20} multiplications. This means robustness essentially comes free in the case of large circuits. The multiplication protocol supports probabilistic truncation *without* overhead and multi-input multiplication gates.

Probabilistic truncation without any overhead. Multiplication (and dot product) with truncation forms an essential component while working with fixed-point values. Techniques for probabilistic truncation were proposed by [29, 30]. Recently, [27] gave an efficient instantiation of truncation for 4PC with abort, based on the technique of ABY3. Using that as a baseline, we show for the *first time*, how fair and robust multiplication (and dot-product) with truncation can be performed without any additional cost over a multiplication.

Multi-input multiplication. Inspired by [31, 32], we propose new protocols for 3 and 4-input multiplication, allowing multiplication of 3 and 4 inputs in one online round. Naively, performing a 4-input multiplication follows a tree-based approach, and the required communication is that of three 2-input multiplications and 2 online rounds. Our contribution lies in keeping the communication and the round of the online phase the same as that of 2-input multiplication (i.e. invariant of the number of inputs). To achieve this, we trade off the preprocessing cost. Looking ahead, multi-input multiplication, when coupled with the optimized parallel prefix adder circuit from [32], brings in a 2× improvement in online rounds. It also cuts down the online communication of secure comparison, impacting PPML applications.

2.2 4PC Mixed-Protocol Framework

In addition to relying on the improved arithmetic/boolean world, we observe that a large portion of the computation in most MPC-based PPML frameworks is done over these worlds. The garbled world is used only to perform the non-linear operations (e.g. softmax) that are expensive in the arithmetic/boolean world and switch back immediately after. Leveraging this observation we propose tailor-made GC-based protocols with *end-to-end* conversion techniques.

The tailor-made GC for the fair protocols, has the following advantages over Trident – i) no use of commitments for the inputs, and ii) no requirement of an explicit input sharing and output reconstruction phase. The overall communication cost remains the same as Trident with 1 GC and 2 online rounds. In addition, for time-constrained applications we offer a variant that trades off 1 GC at the expense of 1 lesser online round. When it comes to robustness, the state-of-the-art for GC protocols are [19], costing 12 GC and 2 rounds, and [7], costing 2 GC and 4 rounds. We propose robust GC conversions for the first time, and they cost 2 GC and have an amortized round complexity of 1.

As mentioned earlier, the framework operates over three domains - arithmetic, boolean, and garbled. For an operation that requires computing over the garbled domain, the standard approach is to first switch from *Arithmetic to Garbled* and evaluate the garbled circuit to obtain a garbled-shared output. These shares are brought back to the arithmetic domain using a *Garbled to Arithmetic* conversion. Our approach instead is to modify the garbled circuit such that the output is in the arithmetic domain. This eliminates the need for an explicit *Garbled to Arithmetic* conversion, saving in both communication and rounds in the online phase. End-to-end conversions are of the form "x-Garbled-x" where x can be either arithmetic or boolean, and need a single round for the garbled world.

Comparison of Tetrad with actively secure PPML frameworks in 3PC and 4PC is presented in Table 1. The dot product is chosen as a parameter as it is one of the most crucial building blocks in PPML applications.

3 IMPLEMENTATION AND BENCHMARKING

We benchmark training and inference phases of the following deep NNs with varying parameter sizes, and the inference phase for

Support Vector Machines (SVM) using MNIST [25] and CIFAR-10 [23] dataset. Training phase of SVM requires additional tools and primitives, and is out of scope of this work. We refer readers to [30, 38] for the architecture and description of training/inference steps for the ML algorithms.

– *SVM:* Consists of 10 categories for classification [12].

– *NN-1:* Fully connected network with 3 layers [29, 33].

– *NN-2:* Convolutional neural network comprising of 2 hidden layers, with 100 and 10 nodes [9, 29, 34].

– *NN-3:* LeNet [24], comprises of 2 convolutional and fully connected layers, followed by maxpool for convolutional layers.

– *NN-4:* VGG16 [36] has 16 layers in total and contains fully-connected, convolutional, ReLU activation and maxpool layers.

We evaluate NN-1, NN-3, SVM on MNIST dataset which is a collection of 28×28 pixel, handwritten digit images with a label between 0 and 9 for each. NN-2, NN-4 are evaluated on CIFAR-10 dataset which has 32×32 pixel images of 10 different classes such as dogs, horses, etc. Benchmarks are against the state-of-the-art 4PC of Trident [9] and SWIFT [21] 4PC (supports only inference).

3.1 Benchmarking Environment Details

The protocols are benchmarked over a Wide Area Network (WAN), instantiated using n1-standard-64 instances of Google Cloud[1], with machines located in East Australia (P_0), South Asia (P_1), South East Asia (P_2), and West Europe (P_3). The machines are equipped with 2.0 GHz Intel (R) Xeon (R) (Skylake) processors supporting hyper-threading, with 64 vCPUs, and 240 GB of RAM Memory. Parties are connected by pairwise authenticated bidirectional synchronous channels (e.g., instantiated via TLS over TCP/IP). We use a bandwidth of 40 MBps between every pair of parties and the average round-trip time (rtt)[2] values among P_0-P_1, P_0-P_2, P_0-P_3, P_1-P_2, P_1-P_3, and P_2-P_3 are 153.74ms, 93.39ms, 274.84ms, 62.01ms, 174.15ms, and 219.46ms respectively.

For a fair comparison, we implemented and benchmarked all the protocols, including the protocols of Trident and SWIFT, building on the ENCRYPTO library [10] in C++17. Primitives such as maxpool, which Trident and SWIFT do not support, have been run using our building blocks. We would like to clarify that our code is developed for benchmarking, is not optimized for industry-grade use, and optimizations like GPU support can further enhance performance. Our protocols are instantiated over a 64-bit ring ($\mathbb{Z}_{2^{64}}$), and the collision-resistant hash function is instantiated using SHA-256. We use multi-threading, and our machines are capable of handling a total of 64 threads. Experiments are run 10 times and average values are reported. Batch size of $B = 128$ for training and 1 KB = 8192.

3.2 Benchmarking Parameters

We evaluate the protocols across a variety of parameters as given in Table 2. In addition to parameters such as runtime, communication, and *online throughput* (TP) [3, 4, 9, 29], the cumulative runtime (sum of the up-time of all the hired servers) is also reported. This is because when deployed over third-party cloud servers, one pays for them by the communication and the uptime of the hired servers. To analyze the cost of deployment of the framework, *monetary cost* (Cost) [28] is reported. This is done using the pricing of Google Cloud Platform[3], where for 1 GB and 1 hour of usage, the costs are USD 0.08 and USD 3.04, respectively. For protocols with an asymmetric communication graph, communication load is unevenly distributed among all the servers, leaving several communication channels underutilized. Load balancing improves the performance by running several execution threads in parallel, each with the roles of the servers changed. Load balancing has been performed in all the protocols benchmarked.

Notation	Description
$\mathsf{T}_{\mathsf{on,i}}$	Online runtime of party P_i.
$\mathsf{T}_{\mathsf{tot,i}}$	Total runtime of party P_i.
$\mathsf{PT}_{\mathsf{on}}$	Protocol online runtime; $\max_i\{\mathsf{T}_{\mathsf{on,i}}\}$.
$\mathsf{PT}_{\mathsf{tot}}$	Protocol total runtime; $\max_i\{\mathsf{T}_{\mathsf{tot,i}}\}$.
$\mathsf{CT}_{\mathsf{on}}$	Cumulative online runtime; $\Sigma_i \mathsf{T}_{\mathsf{on,i}}$.
$\mathsf{CT}_{\mathsf{tot}}$	Cumulative total runtime; $\Sigma_i \mathsf{T}_{\mathsf{tot,i}}$.
$\mathsf{Comm}_{\mathsf{on}}$	Online communication.
$\mathsf{Comm}_{\mathsf{tot}}$	Total communication.
Cost	Total monetary cost.
TP	Online throughput (higher = better) (#iterations / #queries per minute in online)

Table 2: Benchmarking parameters (lower is better, except for TP)

3.2.1 Discussion. Broadly speaking, we consider two deployment scenarios – optimized for time (T), and for cost (C). In the first one ($\mathsf{Tetrad}_\mathsf{T}$), participants want the result of the output as soon as possible while maximizing the online throughput. In the second one ($\mathsf{Tetrad}_\mathsf{C}$), they want the overall monetary cost of the system to be minimal and are willing to tolerate an overhead in the execution time. Using multi-input multiplication gates and the 2 GC variant of the garbled world makes the online phase faster but incurs an increase in monetary cost. This is because they cause an overhead in communication in the preprocessing phase, and communication affects monetary cost more than uptime (in our setting).

We report only the numbers for the fair variant of Tetrad and not the robust variant. This is because the overhead of robust over its fair counterpart is very minimal for deep networks, like those considered in this work. Both variants are compared against Trident [9], and their relative performance is indicated in Table 3.

Protocol	Training & Inference[a]			Training	Inference
	$\mathsf{Time}_{\mathsf{on}}$[b]	$\mathsf{Com}_{\mathsf{tot}}$	$\mathsf{CT}_{\mathsf{tot}}$	Cost	$\mathsf{TP}_{\mathsf{on}}$
$\mathsf{Tetrad}_\mathsf{T}$	●	◐	●	◐	●
$\mathsf{Tetrad}_\mathsf{C}$	◐	●	◐	●	◐
Trident	○	○	○	○	○

[a] 'Com' - Communication, 'Time' - Runtime, 'CT' - Cumulative Runtime, 'Cost' - Monetary Cost, '$\mathsf{TP}_{\mathsf{on}}$' - Online Throughput, on - online, tot - total

[b] ○ - good, ◐ - better, ● - best, (w.r.t parameter considered)

Table 3: Comparison of Trident [9] with the versions of Tetrad [22] for deep neural networks (cf. NN-4 in §3).

Observe that Tetrad is better when compared to Trident across all the parameters considered. Within Tetrad, $\mathsf{Tetrad}_\mathsf{T}$ fares better

[1] https://cloud.google.com/

[2] Time for communicating 1 KB of data between a pair of parties

[3] See https://cloud.google.com/vpc/network-pricing for network cost and https://cloud.google.com/compute/vm-instance-pricing for computation cost.

when it comes to online run time for both training and inference, while Tetrad$_C$ does better in terms of communication. When it comes to inference, throughput is more relevant than the cost, and here, the time-optimized variant fares the best. Robust variants follow the same trends.

3.3 ML Training

For training we consider NN-1, NN-2, NN-3 and NN-4 networks. We report values corresponding to one iteration, that comprises of a forward and backward propagation.

Starting with the time-optimized variant, Tetrad$_T$ is $3-4\times$ faster than Trident in online runtime. The primary factor is the reduction in online rounds of our protocol due to multi-input gates. More precisely, we use the depth-optimized bit extraction circuit while instantiating the ReLU activation function using multi-input AND gates. Looking at the total communication (Comm$_{tot}$) in Table 4, we observe that the gap in Comm$_{tot}$ between Tetrad$_T$ vs. Trident decreases as the networks get deeper. This is justified as the improvement in communication of our dot product with truncation outpaces the overhead in communication caused by multi-input gates. The impact of this is more pronounced with NN-4, as observed by the lower monetary cost of Tetrad$_T$ over Trident. Another reason is that there are two active parties (P_1, P_2) in our framework, whereas Trident has three. Given the allocation of servers, the best rtt Trident can get with three parties (P_0, P_1, P_2) is $153.74ms$, as compared to $62.01ms$ of Tetrad, contributing to Tetrad being faster. However, if the rtt among all the parties were similar, this gap would be closed. Concretely, the online runtime (PT$_{on}$) of Trident will be similar to that of Tetrad$_C$.

Algorithm	Parameter	Trident	Tetrad$_T$	Tetrad$_C$
NN-2	PT$_{on}$	8.13	2.05	2.67
	PT$_{tot}$	11.47	5.79	6.14
	CT$_{tot}$	30.88	14.82	13.40
	Comm$_{tot}$	0.28	0.39	0.24
	Cost	70.00	75.67	49.16
	TP	428.16	652.75	644.69
NN-3	PT$_{on}$	21.79	5.67	8.40
	PT$_{tot}$	30.66	15.14	17.87
	CT$_{tot}$	91.68	40.01	42.76
	Comm$_{tot}$	1.59	1.94	1.28
	Cost	331.01	343.73	240.41
	TP	53.62	55.71	54.13
NN-4	PT$_{on}$	72.01	25.90	38.35
	PT$_{tot}$	283.89	182.13	194.58
	CT$_{tot}$	859.09	500.13	522.32
	Comm$_{tot}$	31.59	29.52	22.24
	Cost	5779.27	5146.10	3999.30
	TP	2.55	2.61	2.56

Table 4: Benchmarking of the training phase of ML algorithms. Time (in seconds) and communication (in GB) are reported for 1 iteration. Monetary cost (USD) is reported for 1000 iterations.

The cost-optimized variant Tetrad$_C$ on the other hand, is $1.5\times$ slower in the online phase compared to Tetrad$_T$. However, it is still faster than Trident owing to the rtt setup, as discussed above. For monetary cost, this variant is up to $20-40\%$ cheaper than it's time-optimized counterpart, and by around 30% over Trident.

3.4 ML Inference

We benchmark the inference phase of SVM and the aforementioned NNs. In addition to Trident [9], we also benchmark against the 4PC robust protocol of SWIFT [21] since it supports NN inference.

Algorithm	Parameter	Trident	Tetrad$_T$	Tetrad$_C$	SWIFT
SVM	PT$_{on}$	17.09	2.91	4.77	5.21
	PT$_{tot}$	17.37	3.19	5.05	6.04
	CT$_{tot}$	47.02	6.99	10.70	14.47
	Comm$_{tot}$	1.36	2.34	1.25	1.36
	Cost	39.92	6.26	9.23	12.43
	TP	898.80	5271.74	3221.29	2949.76
NN-3	PT$_{on}$	14.42	2.61	4.10	4.54
	PT$_{tot}$	14.71	2.91	4.39	5.39
	CT$_{tot}$	39.92	6.43	9.40	13.18
	Comm$_{tot}$	5.62	8.42	4.76	5.39
	Cost	34.59	6.74	8.68	11.97
	TP	1065.35	5882.44	3746.89	3384.51
NN-4	PT$_{on}$	47.05	7.85	12.69	13.13
	PT$_{tot}$	47.61	8.44	13.28	14.33
	CT$_{tot}$	129.41	17.77	27.46	31.35
	Comm$_{tot}$	85.69	124.09	71.27	81.33
	Cost	122.66	34.40	34.32	39.18
	TP	326.46	934.34	891.19	891.19

Table 5: Benchmarking of the inference phase of ML algorithms. Time (in seconds) and communication (in MB) are reported for 1 query. Monetary cost (USD) is reported for 1000 queries.

Similar to training, the time-optimized variant for inference is faster when it comes to PT$_{on}$, by $4-6\times$ over Trident. This is also reflected in the TP, where the improvement is about $2.8-5.5\times$, as evident from Figure 1. In inference, the communication is in the order of megabytes, while run time is in the order of a few seconds. The key observation is that communication is well suited for the bandwidth used (40 MBps). So unlike training, the monetary cost in inference depends more on run time rather than on communication. This is evident from Table 5 which shows that Tetrad$_T$ saves on monetary cost up to a factor of 6 over Trident.

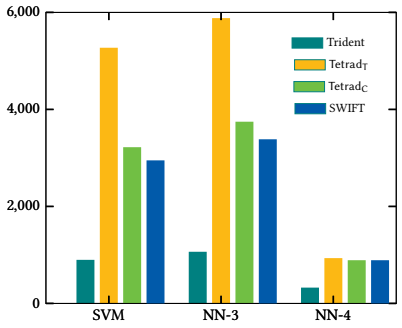

Figure 1: Inference of SVM, NN-3 and NN-4: in terms of TP (higher is better)

Note that the cost-optimized variant under performs in terms of monetary cost compared to Tetrad$_T$. This is because, as mentioned earlier, run time plays a bigger role in monetary cost than communication. Hence for inference, the time-optimized variant becomes the optimal choice.

REFERENCES

[1] Mark Abspoel, Anders Dalskov, Daniel Escudero, and Ariel Nof. 2019. An Efficient Passive-to-Active Compiler for Honest-Majority MPC over Rings. Cryptology ePrint Archive, Report 2019/1298. https://eprint.iacr.org/2019/1298.

[2] Javier Alvarez-Valle, Pratik Bhatu, Nishanth Chandran, Divya Gupta, Aditya V. Nori, Aseem Rastogi, Mayank Rathee, Rahul Sharma, and Shubham Ugare. 2020. Secure Medical Image Analysis with CrypTFlow. *CoRR* abs/2012.05064 (2020). https://arxiv.org/abs/2012.05064

[3] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. 2017. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 843–862. https://doi.org/10.1109/SP.2017.15

[4] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 805–817. https://doi.org/10.1145/2976749.2978331

[5] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PoPETs* 2020, 2 (April 2020), 459–480. https://doi.org/10.2478/popets-2020-0036

[6] Megha Byali, Carmit Hazay, Arpita Patra, and Swati Singla. 2019. Fast Actively Secure Five-Party Computation with Security Beyond Abort. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 1573–1590. https://doi.org/10.1145/3319535.3345657

[7] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. 2018. Fast Secure Computation for Small Population over the Internet. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 677–694. https://doi.org/10.1145/3243734.3243784

[8] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In *ACM CCSW@CCS*. https://eprint.iacr.org/2019/429

[9] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS 2020*. The Internet Society.

[10] Cryptography and Privacy Engineering Group at TU Darmstadt. 2017. ENCRYPTO Utils. https://github.com/encryptogroup/ENCRYPTO_utils.

[11] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In *USENIX Security'21*. https://eprint.iacr.org/2020/1330.

[12] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. 2019. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *2019 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1102–1120. https://doi.org/10.1109/SP.2019.00078

[13] Ivan Damgård, Claudio Orlandi, and Mark Simkin. 2018. Yet Another Compiler for Active Security or: Efficient MPC Over Arbitrary Rings. In *CRYPTO 2018, Part II (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.), Vol. 10992. Springer, Heidelberg, 799–829. https://doi.org/10.1007/978-3-319-96881-0_27

[14] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS 2015*. The Internet Society.

[15] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. 2020. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In *CRYPTO 2020, Part II (LNCS)*, Daniele Micciancio and Thomas Ristenpart (Eds.), Vol. 12171. Springer, Heidelberg, 823–852. https://doi.org/10.1007/978-3-030-56880-1_29

[16] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. 2017. High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In *EUROCRYPT 2017, Part II (LNCS)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.), Vol. 10211. Springer, Heidelberg, 225–255. https://doi.org/10.1007/978-3-319-56614-6_8

[17] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th ACM STOC*, Alfred Aho (Ed.). ACM Press, 218–229. https://doi.org/10.1145/28395.28420

[18] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. 2018. Secure Computation with Low Communication from Cross-Checking. In *ASIACRYPT 2018, Part III (LNCS)*, Thomas Peyrin and Steven Galbraith (Eds.), Vol. 11274. Springer, Heidelberg, 59–85. https://doi.org/10.1007/978-3-030-03332-3_3

[19] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. 2015. Secure Computation with Minimal Interaction, Revisited. In *CRYPTO 2015, Part II (LNCS)*, Rosario Gennaro and Matthew J. B. Robshaw (Eds.), Vol. 9216. Springer, Heidelberg, 359–378. https://doi.org/10.1007/978-3-662-48000-7_18

[20] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ Great Again. In *EUROCRYPT 2018, Part III (LNCS)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.), Vol. 10822. Springer, Heidelberg, 158–189. https://doi.org/10.1007/978-3-319-78372-7_6

[21] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. SWIFT: Superfast and Robust Privacy-Preserving Machine Learning. In *USENIX Security'21*. https://eprint.iacr.org/2020/592.

[22] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. 2021. Tetrad: Actively Secure 4PC for Secure Training and Inference. *IACR Cryptol. ePrint Arch.* (2021). https://eprint.iacr.org/2021/755.

[23] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. (2014). https://www.cs.toronto.edu/~kriz/cifar.html.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998), 2278–2324.

[25] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. (2010). http://yann.lecun.com/exdb/mnist/

[26] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. 2019. EPIC: Efficient Private Image Classification (or: Learning from the Masters). In *CT-RSA 2019 (LNCS)*, Mitsuru Matsui (Ed.), Vol. 11405. Springer, Heidelberg, 473–492. https://doi.org/10.1007/978-3-030-12612-4_24

[27] Sahar Mazloom, Phi Hung Le, Samuel Ranellucci, and S. Dov Gordon. 2020. Secure parallel computation on national scale volumes of data. In *USENIX Security 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2487–2504.

[28] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. 2020. Two-Sided Malicious Security for Private Intersection-Sum with Cardinality. In *CRYPTO 2020, Part III (LNCS)*, Daniele Micciancio and Thomas Ristenpart (Eds.), Vol. 12172. Springer, Heidelberg, 3–33. https://doi.org/10.1007/978-3-030-56877-1_1

[29] Payman Mohassel and Peter Rindal. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 35–52. https://doi.org/10.1145/3243734.3243760

[30] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 19–38. https://doi.org/10.1109/SP.2017.12

[31] Satsuya Ohata and Koji Nuida. 2020. Communication-Efficient (Client-Aided) Secure Two-Party Protocols and Its Application. In *FC 2020 (LNCS)*, Joseph Bonneau and Nadia Heninger (Eds.), Vol. 12059. Springer, Heidelberg, 369–385. https://doi.org/10.1007/978-3-030-51280-4_20

[32] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security'21*. https://eprint.iacr.org/2020/1225.

[33] Arpita Patra and Ajith Suresh. 2020. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS 2020*. The Internet Society.

[34] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *ASIACCS 18*, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM Press, 707–721.

[35] Dragos Rotaru and Tim Wood. 2019. MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security. In *INDOCRYPT 2019 (LNCS)*, Feng Hao, Sushmita Ruj, and Sourav Sen Gupta (Eds.), Vol. 11898. Springer, Heidelberg, 227–249. https://doi.org/10.1007/978-3-030-35423-7_12

[36] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[37] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *PoPETs* 2019, 3 (July 2019), 26–49. https://doi.org/10.2478/popets-2019-0035

[38] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *PoPETs* 2021, 1 (Jan. 2021), 188–208. https://doi.org/10.2478/popets-2021-0011

[39] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd FOCS*. IEEE Computer Society Press, 160–164. https://doi.org/10.1109/SFCS.1982.38