CryptoSPN: Expanding PPML beyond Neural Networks*

Amos Treiber
TU Darmstadt
treiber@encrypto.cs.tu-darmstadt.de

Alejandro Molina
TU Darmstadt
molina@cs.tu-darmstadt.de

Christian Weinert
TU Darmstadt
weinert@encrypto.cs.tu-darmstadt.de

Thomas Schneider
TU Darmstadt
schneider@encrypto.cs.tu-darmstadt.de

Kristian Kersting
TU Darmstadt
kersting@cs.tu-darmstadt.de

ABSTRACT

The ubiquitous deployment of machine learning (ML) technologies has certainly improved many applications but also raised challenging privacy concerns, as sensitive client data is usually processed remotely at the discretion of a service provider. Therefore, privacy-preserving machine learning (PPML) aims at providing privacy using techniques such as secure multi-party computation (SMPC).

Recent years have seen a rapid influx of cryptographic frameworks that steadily improve performance as well as usability, pushing PPML towards practice. However, as it is mainly driven by the crypto community, the PPML toolkit so far is mostly restricted to well-known neural networks (NNs). Unfortunately, deep probabilistic models rising in the ML community that can deal with a wide range of probabilistic queries and offer tractability guarantees are severely underrepresented. Due to a lack of interdisciplinary collaboration, PPML is missing such important trends, ultimately hindering the adoption of privacy technology.

In this work, we introduce CryptoSPN, a framework for privacy-preserving inference of sum-product networks (SPNs) to significantly expand the PPML toolkit beyond NNs. SPNs are deep probabilistic models at the sweet-spot between expressivity and tractability, allowing for a range of exact queries in linear time. In an interdisciplinary effort, we combine techniques from both ML and crypto to allow for efficient, privacy-preserving SPN inference via SMPC.

We provide CryptoSPN as open source and seamlessly integrate it into the SPFlow library (Molina et al., arXiv 2019) for practical use by ML experts. Our evaluation on a broad range of SPNs demonstrates that CryptoSPN achieves highly efficient and accurate inference within seconds for medium-sized SPNs.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; **Privacy protections**; *Usability in security and privacy*.

*This is an extended abstract of our paper at ECAI'20 [42]. Please cite the full version.

PPMLP'20, November 9, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8088-1/20/11…$15.00
https://doi.org/10.1145/3411501.3419417

KEYWORDS

privacy-preserving inference; sum-product networks; secure computation

ACM Reference Format:
Amos Treiber, Alejandro Molina, Christian Weinert, Thomas Schneider, and Kristian Kersting. 2020. CryptoSPN: Expanding PPML beyond Neural Networks. In *2020 Workshop on Privacy-Preserving Machine Learning in Practice (PPMLP'20), November 9, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3411501.3419417

1 INTRODUCTION

Machine learning (ML) is omnipresent in today's digital landscape. However, there persist a range of pressing issues in common ML-as-a-Service (MLaaS) scenarios: Since ML models usually constitute sensitive intellectual property of the service provider, they cannot be made directly available to clients. Instead, clients have to trust the service provider with their inputs. Concerns about the privacy violations implied by this practice were recently addressed from a legal perspective by regulations such as the European General Data Protection Regulation (GDPR). It affects ML applications in two ways: the aforementioned *privacy of client data* and the *capabilities of ML models*, as restrictions are placed on what decisions can be made, which can be seen as a "right to explanation" [18].

From a technical perspective, the issue of protecting client data has been heavily addressed by cryptographic research in recent years under the umbrella of privacy-preserving ML (PPML), leveraging techniques such as homomorphic encryption (HE) or secure multi-party computation (SMPC). The former allows for computation under encryption, while the latter allows multiple parties to interactively compute functions represented as *circuits* on their private inputs. PPML securely realizes MLaaS: The client has as input features $\boldsymbol{X}$, and the server has as input an ML model M. In the end, the client receives $M(\boldsymbol{X})$ without learning any other information, and the server learns nothing. Though said techniques are moving from theory towards practical implementations, reducing the overhead of cryptographic protection is still considered the main PPML research challenge. Thus, many PPML frameworks, e.g., [7, 17, 21, 25, 27, 29, 30, 38, 39], introduce a range of performance optimizations. However, they almost exclusively focus on neural networks (NNs), which are evaluated more and more efficiently via cryptographic trickery as well as model approximations.

In contrast, ML research has mainly focused on expanding the capabilities and trustworthiness of models: NNs can provide impressive accuracy, but generally cannot indicate uncertainty nor

handle general probabilistic reasoning. These concepts are correlated with trust, as an expression of uncertainty shows how suitable the model is to make a prediction. Therefore, they are sought after in real-world applications and fortunately can be achieved by new deep probabilistic models such as sum-product networks (SPNs) [37], a tractable probabilistic graphical model [24]. These models emerged as one of the principal theoretical and practical approaches to ML [16], as they provide an understanding of what inference and learning are, and can perform a wide range of different ML tasks. A challenge here is the trade-off between expressivity and tractability, and SPNs are considered to be at the sweet-spot: They allow for tractable, exact inference for a wide range of queries that benefit many real-world applications, e.g., [2, 20, 35, 46].

Obviously, there is a disparity between the NN-centric PPML literature and the capabilities as well as trustworthiness of the models sought after by ML experts and regulations. Additionally, most optimizations and approximations were performed by cryptographers[1], often ignoring the expertise and requirements of ML experts. Nowadays, more capable ML models are in demand, and the lack of support for such models in PPML frameworks hinders their adoption in the ML domain, where its intended users reside.

Therefore, we significantly broaden the PPML horizon by introducing support for deep probabilistic models via CryptoSPN [42], a framework for privacy-preserving SPN inference based on SMPC. It uses both cryptographic and ML techniques for its privacy guarantees, even protecting information encoded in the SPN structure. We implement and directly integrate it into the SPN library SPFlow [33], making it practically usable for ML experts without requiring any cryptographic knowledge. In a practical evaluation on standard SPNs, we show that we can privately evaluate medium-sized SPNs within seconds on commodity hardware.

2 RELATED WORK

The appearance of first practical SMPC frameworks [28] paved the way for initial considerations of private neural network evaluation [6, 34, 41]. Practical secure classification tasks for less demanding models such as hyper-plane decision, naive Bayes, and decision trees was provided in [9]. What followed is a race for the fastest NN-based privacy-preserving image classification that is still ongoing, with new works appearing rapidly. Starting with CryptoNets [17], notable works include MiniONN [27], Chameleon [39], Gazelle [21], ABY3 [30], XONN [38], and DELPHI [29]. Additionally, SecureML [31] even allows for neural network training using stochastic gradient descent (SGD). The way these frameworks operate is to find approximate NN computations that are more suitable for an efficient evaluation with HE and/or SMPC that involves clever tricks such as converting between protocols to reduce runtimes.

An orthogonal research subject is usability, as privacy-preserving frameworks are intended to be used by ML experts with little to no cryptographic knowledge and no experience with such tools. Thus, direct integrations with ML frameworks exist: For TensorFlow there are integrations for differential privacy [3], HE [44], and SMPC [5, 12, 25], for PyTorch there is a direct SMPC integration by Facebook's AI researchers [19] as well as PySyft [40], and there are HE [8] and SMPC [7] backends for Intel's nGraph compiler [11] that are compatible with PyTorch and TensorFlow.

In this interdisciplinary work, we extend the scope of PPML to the more capable sum-product networks (SPNs), by leveraging both cryptographic and ML tools for efficient, usable, and privacy-preserving SPN inference.

3 SUM-PRODUCT NETWORKS (SPN)

SPNs encode the joint probability $P(\boldsymbol{X}, \boldsymbol{Y})$ of random variables (RVs) $\boldsymbol{X}, \boldsymbol{Y}$ in the structure of arithmetic circuits (ACs) [10]. They guarantee exact, tractable answers for a range of probabilistic queries. An SPN is defined as a rooted directed acyclic graph, consisting of *sum*, *product*, and *leaf* nodes, and the set of RVs within the network is denoted by its *scope*. The leaves are tractable univariate distributions on the RVs with alternating sum and product nodes on top. A product node represents a factorization over independent distributions over different RVs ($P(\boldsymbol{X}, \boldsymbol{Y}) = P(\boldsymbol{X})P(\boldsymbol{Y})$) and is defined as the product of its children. A sum node encompasses a mixture of distributions defined over the same RVs ($P(\boldsymbol{X}, \boldsymbol{Y}) = wP_1(\boldsymbol{X}, \boldsymbol{Y}) + (1-w)P_2(\boldsymbol{X}, \boldsymbol{Y})$) and is defined as the weighted sum of its children. Therefore, SPNs can be built recursively by considering (i) a tractable univariate distribution, (ii) a product of SPNs over *different* scopes, and (iii) a weighted sum of SPNs over *the same* scope as an SPN.

3.1 Tractable Inference in SPNs

SPNs are evaluated bottom up for a probabilistic query given the input RVs, leaf distribution parameters, and sum weights. The output is the value of the root. Due to the SPN's structure, one can easily evaluate marginals ($P(\boldsymbol{X})$; partial RV inputs) by setting the corresponding leaf probabilities to 1. Therefore, marginals can be computed efficiently and a conditional query can be computed as $P(\boldsymbol{Y}|\boldsymbol{X}) = P(\boldsymbol{X}, \boldsymbol{Y})/P(\boldsymbol{X})$, allowing queries to be answered in linear complexity.

3.2 SPN Applications

SPNs are at a sweet-spot between expressivity and tractability with the sought-after advantages of tractable probabilistic reasoning. Therefore, they can indicate uncertainty and deal with missing data, which makes them more trustworthy than other models. This is reflected in increased research and usage in practice, with SPNs making appearances in real-world applications such as speech recognition [35], activity recognition [2], scene understanding [46], and even DB management systems [20].

Due to the popularity of research in such applications, software frameworks for SPNs emerged and can be used by ML experts for easy deployment and advancement of SPNs. A prominent example is SPFlow [33], an actively maintained open-source Python library with support for SPN learning, manipulation, and inference. SPNs are interfaced via a domain-specific language and internally represented as a functional graph structure. This allows for easy extension and adaption of the codebase by the community. It already provides compilation into TensorFlow, PyTorch, C, CUDA, and FPGA custom code for speeding up the computations. Furthermore, it encompasses visualization and serialization of SPNs to foster research on probabilistic modeling.

[1] An exception is DELPHI [29], which combines ML and cryptographic techniques.

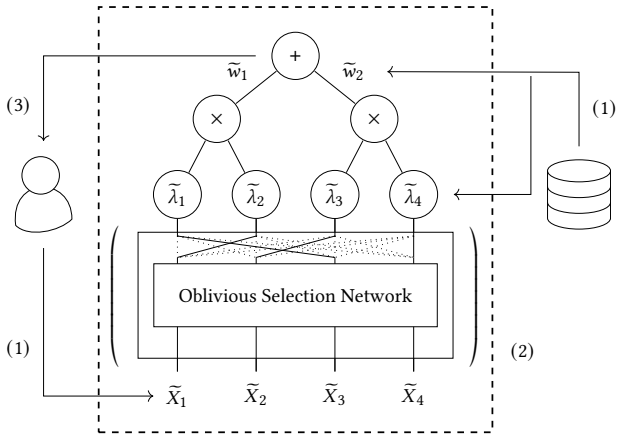

Figure 1: CryptoSPN protocol flow for a miniature SPN with Poisson leaves: (1) the client has private inputs $X_{1,\dots,4}$ and the server has private inputs $w_{1,2}, \lambda_{1,\dots,4}$; (2) private evaluation of leaf-, sum- and product nodes using Yao's GC protocol [45]; (3) client receives SPN inference result.

With the increased practical application of SPNs across domains, it is crucial to expand the PPML toolkit to SPNs and make such deep probabilistic models privacy-preserving. To encourage actual usage, results need to be integrated with familiar tools, such as SPFlow.

4 CRYPTOSPN ARCHITECTURE

With CryptoSPN, we provide privacy-preserving SPN inference in the common PPML setting: the client has as input RVs $\boldsymbol{X} = X_1, \dots, X_n$ and the server has as input an SPN P, with the output being $P(\boldsymbol{X})$ for the client. Our approach is to evaluate the SPN within SMPC, given that it already is in a circuit representation. An overall overview of the CryptoSPN workflow can be found in Fig. 1.

Evaluating an SPN within SMPC comes with a unique set of challenges that we have to face to allow for useful and practical private inference. In the following, we give an overview of these challenges and how we address them. For more technical details as well as detailed cost analyses we refer to the full paper [42].

4.1 Exactness Guarantees

One of the main advantages of performing probabilistic inference with SPNs is to have guarantees on tractability and exactness (cf. §3). Therefore, to obtain useful results, we cannot resort to fixed-point approximations, as is often done in the private NN literature. Instead, we have to incorporate Boolean sub-circuits for IEEE 754-compliant floating point operations [13] with either 32-bit or 64-bit precision. Since these sub-circuits introduce a lot of depth to the circuit, we use the constant-round SMPC protocol Yao's garbled citcuits (GC) [45] with state-of-the-art optimizations. Thus, the round complexity is independent of the circuit depth.

4.2 Efficiently Hiding the RVs and Parameters

With the evaluation in a GC, the RVs and model parameters are hidden according to the GC security guarantees. Our implementation provides semi-honest security, where the parties are assumed to honestly follow the protocol, but can be expanded to security against malicious adversaries with known techniques at some overhead, e.g., [26]. An advantage of our approach is that by using maliciously secure oblivious transfer, it can achieve security against a malicious client with very little overhead [4].

However, an issue we face is that due to the floating point sub-circuits, we obtain very large garbled circuits. We mitigate this in two ways: First, we use the *log-sum-exp* trick to compute everything in log2 space, which not only avoids underflows but also decreases circuit size, as the frequently appearing products are replaced by cheaper additions. Furthermore, circuits for log2 and exp2 operations have a significantly smaller size compared to the conventional natural operations. Second, we show how to compute leaf distributions within the log2 domain in a way that minimizes the amount of operations necessary to be performed in the garbled circuit.

Additionally, due to the weighted sums, the server has a lot more inputs than the client in a GC for an SPN. Therefore, the server acts as garbler whose inputs in the GC protocol are relatively cheap.

4.3 Efficiently Hiding the Training Data

Even though we efficiently hide all parameters with our construction, the underlying circuit and therefore the SPN structure has to be made public. This is an issue as sensitive training data might be encoded in the structure [15]. One could use further cryptographic techniques for *private function evaluation* that also hides the function from the client, e.g., via universal circuits (UCs) [1, 43]. However, this would significantly enlarge the overhead of CryptoSPN and make it unusable in practice.

Instead, we use a machine learning technique that allows for data-independent structure learning: One can learn a valid SPN structure with random tensorized (RAT-)SPNs [36] prior to seeing any training data. This structure only depends on non-sensitive hyperparameters, and is built by randomly splitting the RVs into region graphs that finally build the SPN. Then, only the parameters are learned using the training dataset.

Since RAT-SPN performance is comparable and sometimes even better than conventional learning [36], we use a RAT-SPN as structure for the SPNs evaluated in CryptoSPN. This is an example where ML techniques can be useful for other purposes and can be employed more efficiently than the cryptographic approach.

4.4 Efficiently Hiding the Scope

We also consider the case where one cannot re-train an existing SPN to a RAT-SPN. For this, we offer a mitigation technique that hides an SPN's scope, i.e., which X_j of the client's RVs $\boldsymbol{X} = X_1, \dots, X_n$ is an input to which leaf distribution. This is a well known problem in cryptography, and recent works [22] suggest that the practically most efficient way to achieve privacy is to evaluate a selection network in SMPC [23]. On a high level, this introduces an additional layer between the m distribution leaves and the n RVs that obliviously selects the RVs $X_{\phi(1)}, \dots, X_{\phi(m)}$ according to a server input $\phi\colon [m] \mapsto [n]$. This denotes which leaf $i \leq m$ has inputs $X_{\phi(i)}$ and is now hidden as an additional server input. Therefore, the scope is hidden for any SPN.

5 CRYPTOSPN IMPLEMENTATION

We integrate CryptoSPN into SPFlow [33] and make it available as open source at https://encrypto.de/code/CryptoSPN. Our goal is to achieve usability for ML experts familiar with SPFlow and its domain-specific language but without any knowledge about cryptography or SMPC. Therefore, we interface via a compilation of any SPFlow SPN instance into an executable that is privacy-preserving. This is similar to how SPFlow already provides TensorFlow, C, CUDA, and FPGA exports and therefore represents no disruption to the usage of SPFlow.

As the backend that invokes Yao's GC protocol, we use the state-of-the-art ABY framework [14]. ABY is implemented in C++ and allows for the secure computation of supplied Boolean or arithmetic circuits with support for *single instruction, multiple data* (SIMD) instructions, which enables CryptoSPN to batch multiple queries.

5.1 Interface and Workflow

The main challenge of the implementation is to align the high-level functional Python representation of an SPN in SPFlow with the required circuit-level C++ input of ABY. We do this by unrolling the SPN, identifying the structure and the inputs, and then converting the SPN into an equivalent Boolean circuit. The resulting circuit is transcribed into the format required by ABY and combined with necessary configurations and setup procedures to obtain an ABY program that performs CryptoSPN's inference on the given SPN. All of this is done in the background and, therefore, the user does not need to specify or interact with any domain-specific knowledge.

However, a simple export into an ABY C++ program still requires the user to deal with the compilation of an ABY program. This is not a problem for the existing exports for common and widely known codebases like library-free C and TensorFlow. But frameworks such as ABY come with some compilation peculiarities and a necessity to familiarize oneself with that might discourage interested experts in exploring CryptoSPN's privacy guarantees for their models and use cases. Consequently, to achieve our goals for ML usability, we provide a *direct* compilation into an executable without any interaction with or specification of ABY or its dependencies.

As a result, a service provider can use SPFlow to create a privacy-preserving executable that can then be executed between the client and the server with their respective inputs. So far, the inputs are supplied as text files to the program in CryptoSPN.

5.2 Sample Application

A very simple example of the workflow is demonstrated in Listing 1. First, some random training data is created using NumPy. Then, a parametric SPN is learned via `learn_parametric`. A `Context` has to be provided, in this case specifying the problem as containing four Bernoullis. This yields the `spn` instance.

The prior steps are regular SPFlow interaction. The user can then invoke the funtion `CryptoSPN.spn_to_aby_file` to export `spn` to an ABY program. Only the bit precision (32 or 64 bit), the filename, and whether an oblivious selection network shall be included has to be specified. To compile the `spn` into an executable, the user simply has to call `CryptoSPN.spn_to_aby_exec`, optionally specifying the ABY and CryptoSPN paths that are necessary for the compilation. The paths can also be set as a constant in CryptoSPN's configuration file. The output is the desired executable, which can be executed just by supplying the server's IP address, the party's respective inputs, and specifying whether one acts as client or as server. This private evaluation corresponds to the plaintext evaluation of calling SPFlow's `log_likelihood`.

Listing 1: Example of CryptoSPN code for creating an executable to be used for privacy-preserving SPN inference. An SPN with four Bernoulli distribution leaves is trained and modified as in SPFlow, and then the executable can be generated without any domain-specific knowledge.

```
# Create example SPN
train_data = np.random.binomial(1, [0.1, 0.8, 0.9, 0.1], size=(100, 4))

ds_context = Context(parametric_types=[Bernoulli, Bernoulli, Bernoulli, Bernoulli]).add_domains(train_data)

spn = learn_parametric(train_data, ds_context, min_instances_slice=20)

# Create file for ABY inputs
np.savetxt("all_data.txt", train_data)

# Create ABY .cpp from SPN instance.
spn_to_aby_file(spn, bitlen=32, filename="example.cpp", sel=Selection.NONE)

# Automatically create ABY executable from SPN instance
spn_to_aby_exec(spn, bitlen=64, name="example", sel=Selection.OSELECTION)

# Plaintext output
train_ll = log_likelihood(spn, train_data)
np.savetxt("example_out_data.txt", train_ll)
```

6 EVALUATION

The integration with SPFlow allows us to easily evaluate a broad range of SPNs. This stands in contrast to many private NN works, where the evaluation is often limited to a few datasets.

Our evaluation is performed on random SPNs trained in SPFlow for standard datasets of [15]. For investigating regular SPNs fortified with an oblivious selection network, we use the `nips` dataset of [32]. Both 32- and 64-bit precision are evaluated. The experiments are performed on two machines with Intel Core i9-7960X CPUs and 128 GB of RAM. We use a symmetric security parameter of $\kappa = 128$ bits according to current recommendations. The connection between both machines is restricted to 100 Mbit/s bandwidth and a round-trip time of 100 ms to simulate a realistic wide-area network (WAN) for a client-server setting. We differentiate between two phases: The *setup* phase performs pre-processing prior to the actual inference necessary for the cryptographic operations in the *online* phase that performs private inference once the inputs are known. An excerpt of the results is given in Tab. 1 for 32-bit precision. We refer to [42] for the full results and a detailed discussion thereof.

In short, the results show that for medium-sized SPNs, private inference is possible within seconds. While this can be a substantial overhead in some applications, it might already be feasible for

Table 1: Benchmark summary for private SPN inference with CryptoSPN in a WAN network with 32-bit precision. All SPNs are RAT-SPNs with Bernoulli leaves except the one for `nips`, which is a regular SPN with Poisson leaves. † indicates selection network usage for hiding RV assignments.

dataset	#edges	setup (min)	online (s)
[15] (all but `nltcs`)	10k-45k	2-7	8-30
[15] (`nltcs`)	3k	0.6	2.7
[32] (`nips`)	3k	1.1	4.6
		$1.2^{\dagger}$	$4.8^{\dagger}$

highly private use cases such as medical diagnostics that are impossible otherwise. We observe that 64-bit precision roughly doubles the cost of 32-bit precision and that, while no single parameter is decisive, runtime performance often scales according to density (especially the amount of edges). Sums introduce more complexity than products, but are much rarer. For the regular SPN for `nips`, we note that the effects of hiding the network's scope are minimal.

We also compare the log-probability results of the CryptoSPN evaluation to the plaintext ones of SPFlow's `log_likelihood`. This gives insignificant RMSEs of 4.2×10^{-9} for 32-bit and 2.3×10^{-17} for 64-bit models. This difference is due to the log2 space evaluation and does not affect the probabilistic inference in a noticeable way.

7 CONCLUSION

While recent advancements push PPML towards practice, more trustworthy probabilistic models have been severely underrepresented in the literature. Therefore, to expand the PPML toolkit and its reach, we provide CryptoSPN for PPML of sum-product networks (SPNs), a trending deep probabilistic model. To ensure usability for ML experts in practice, we implement CryptoSPN and seamlessly integrate it with the SPFlow library [33]. A broad practical evaluation demonstrates that private inference is possible in seconds even for medium-sized SPNs.

With CryptoSPN laying the foundation, future work can further improve performance and study private SPN learning. Furthermore, an important question opened by our work is how vulnerable SPNs are to model extraction attacks.

ACKNOWLEDGMENTS

KK acknowledges the support of the Federal Ministry of Education and Research (BMBF), grant number 01IS18043B "MADESI". This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) — SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within ATHENE.

REFERENCES

[1] Masaud Y. Alhassan, Daniel Günther, Ágnes Kiss, and Thomas Schneider. 2020. Efficient and Scalable Universal Circuits. *Journal of Cryptology (JoC).*
[2] Mohamed R. Amer and Sinisa Todorovic. 2016. Sum product networks for activity recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI).*
[3] Galen Andrew, Steve Chien, and Nicolas Papernot. 2019. TensorFlow Privacy. https://github.com/tensorflow/privacy.
[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2017. More efficient oblivious transfer extensions. *Journal of Cryptology.*
[5] Assi Barak, Daniel Escudero, Anders Dalskov, and Marcel Keller. 2019. *Secure Evaluation of Quantized Neural Networks.*
[6] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. 2011. Privacy-Preserving ECG Classification with Branching Programs and Neural Networks. *IEEE Transactions on Information Forensics and Security (TIFS).*
[7] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. 2020. MP2ML: A Mixed-Protocol Machine Learning Framework for Private Inference. In *International Conference on Availability, Reliability and Security (ARES).*
[8] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC).*
[9] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data. In *Network and Distributed System Security Symposium (NDSS).*
[10] Arthur Choi and Adnan Darwiche. 2017. On Relaxing Determinism in Arithmetic Circuits. In *International Conference on Machine Learning (ICML).*
[11] Scott Cyphers, Arjun K. Bansal, Anahita Bhiwandiwalla, Jayaram Bobba, Matthew Brookhart, Avijit Chakraborty, William Constable, Christian Convey, Leona Cook, Omar Kanawi, Robert Kimball, Jason Knight, Nikolay Korovaiko, Varun Kumar Vijay, Yixing Lao, Christopher R. Lishka, Jaikrishnan Menon, Jennifer Myers, Sandeep Aswath Narayana, Adam Procter, and Tristan J. Webb. 2018. Intel nGraph: An Intermediate Representation, Compiler, and Executor for Deep Learning. *arXiv preprint arXiv:1801.08058.*
[12] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. 2018. Private Machine Learning in TensorFlow using Secure Computation. *arXiv preprint arXiv:1810.08130.*
[13] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni. 2015. Automated synthesis of optimized circuits for secure computation. In *ACM Conference on Computer and Communications Security (CCS).*
[14] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY-A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Network and Distributed System Security Symposium (NDSS).*
[15] Robert Gens and Pedro M Domingos. 2013. Learning the Structure of Sum-Product Networks. In *International Conference on Machine Learning (ICML).*
[16] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature.*
[17] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML.*
[18] Bryce Goodman and Seth Flaxman. 2017. European Union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine.*
[19] David Gunning, Awni Hannun, Mark Ibrahim, Brian Knott, Laurens van der Maaten, Vinicius Reis, Shubho Sengupta, Shobha Venkataraman, and Xing Zhou. 2019. CrypTen: A new research tool for secure machine learning with PyTorch. https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch/.
[20] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries!. In *Proceedings of the VLDB Endowment (PVLDB).*
[21] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security.*
[22] Ágnes Kiss, Masoud Naderpour, Jian Liu, N Asokan, and Thomas Schneider. 2019. SoK: Modular and efficient private decision tree evaluation. *Privacy Enhancing Technologies (PETs).*
[23] Vladimir Kolesnikov and Thomas Schneider. 2008. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security (FC).*
[24] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press.
[25] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow: Secure tensorflow inference. In *IEEE Symposium on Security and Privacy (S&P).*

[26] Yehuda Lindell and Benny Pinkas. 2012. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology (JoC)*.

[27] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. 2017. Oblivious neural network predictions via MiniONN transformations. In *ACM Conference on Computer & Communications Security (CCS)*.

[28] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. 2004. Fairplay - A Secure Two-Party Computation System.. In *USENIX Security*.

[29] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. DELPHI: A Cryptographic Inference Service for Neural Networks. In *USENIX Security*.

[30] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[31] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (S&P)*.

[32] Alejandro Molina, Sriraam Natarajan, and Kristian Kersting. 2017. Poisson Sum-Product Networks: A Deep Architecture for Tractable Multivariate Poisson Distributions. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[33] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. 2019. SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks. *arXiv preprint arXiv:1901.03704*.

[34] Claudio Orlandi, Alessandro Piva, and Mauro Barni. 2007. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*.

[35] Robert Peharz, Georg Kapeller, Pejman Mowlaee, and Franz Pernkopf. 2014. Modeling speech with sum-product networks: Application to bandwidth extension. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[36] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. 2019. Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

[37] Hoifung Poon and Pedro M Domingos. 2011. Sum-Product Networks: A New Deep Architecture. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

[38] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. In *USENIX Security*.

[39] 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 *ACM ASIA Conference on Computer and Communications Security (ASIACCS)*.

[40] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*.

[41] Ahmad-Reza Sadeghi and Thomas Schneider. 2008. Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification. In *International Conference on Information Security and Cryptology (ICISC)*.

[42] Amos Treiber, Alejandro Molina, Christian Weinert, Thomas Schneider, and Kristian Kersting. 2020. CryptoSPN: Privacy-preserving Sum-Product Network Inference. In *European Conference on Artificial Intelligence (ECAI)*.

[43] Leslie G Valiant. 1976. Universal circuits (preliminary report). In *STOC*.

[44] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law. 2019. SEALion: A Framework for Neural Network Inference on Encrypted Data. *arXiv preprint arXiv:1904.12840*.

[45] Andrew Yao. 1986. How to generate and exchange secrets. In *FOCS*.

[46] Zehuan Yuan, Hao Wang, Limin Wang, Tong Lu, Shivakumara Palaiahnakote, and Chew Lim Tan. 2016. Modeling spatial layout for scene image understanding via a novel multiscale sum-product network. *Expert Systems with Applications*.