ECAI 2020
G.D. Giacomo et al. (Eds.)

doi:10.3233/FAIA200313

CryptoSPN: Privacy-Preserving Sum-Product Network Inference

Amos Treiber[1] and **Alejandro Molina**[2] and **Christian Weinert**[1] and **Thomas Schneider**[1] and **Kristian Kersting**[2]

Abstract. AI algorithms, and machine learning (ML) techniques in particular, are increasingly important to individuals' lives, but have caused a range of privacy concerns addressed by, e.g., the European GDPR. Using cryptographic techniques, it is possible to perform inference tasks remotely on sensitive client data in a privacy-preserving way: the server learns nothing about the input data and the model predictions, while the client learns nothing about the ML model (which is often considered intellectual property and might contain traces of sensitive data). While such privacy-preserving solutions are relatively efficient, they are mostly targeted at neural networks, can degrade the predictive accuracy, and usually reveal the network's topology. Furthermore, existing solutions are not readily accessible to ML experts, as prototype implementations are not well-integrated into ML frameworks and require extensive cryptographic knowledge.

In this paper, we present CryptoSPN, a framework for privacy-preserving inference of sum-product networks (SPNs). SPNs are a tractable probabilistic graphical model that allows a range of exact inference queries in linear time. Specifically, we show how to efficiently perform SPN inference via secure multi-party computation (SMPC) without accuracy degradation while hiding sensitive client and training information with provable security guarantees. Next to foundations, CryptoSPN encompasses tools to easily transform existing SPNs into privacy-preserving executables. Our empirical results demonstrate that CryptoSPN achieves highly efficient and accurate inference in the order of seconds for medium-sized SPNs.

1 INTRODUCTION

In our increasingly connected world, the abundance of user information and availability of data analysis techniques originating from artificial intelligence (AI) research has brought machine learning (ML) techniques into daily life. While these techniques are already deployed in many applications like credit scoring, medical diagnosis, biometric verification, recommender systems, fraud detection, and language processing, emerging technologies such as self-driving cars will further increase their popularity.

Privacy Concerns for ML Applications. These examples show that progress in AI research certainly has improved user experience, potentially even saving lives when employed for medical or safety purposes. However, as the prevalent usage in modern applications often requires the processing of massive amounts of sensitive information, the impact on user privacy has come into the public spotlight.

This culminated in privacy regulations such as the European General Data Protection Regulation (GDPR), which came into effect in 2018. Not only does the GDPR provide certain requirements to protect user data, it also includes restrictions on decisions based on user data, which may be interpreted as a "right to explanation" [17]. Luckily, new deep probabilistic models that encode the joint distribution, such as sum-product networks (SPNs) [36], can indicate whether the model is fit to predict the data at hand, or raise a warning otherwise. This increases trust as they "know when they do not know". Moreover, SPNs can also perform inference with missing information [35], an important aspect for real-life applications.

Generally, probabilistic graphical models [24] provide a framework for understanding what inference and learning are, and have therefore emerged as one of the principal theoretical and practical approaches to ML and AI [14]. However, one of the main challenges in probabilistic modeling is the trade-off between the expressivity of the models and the complexity of performing various types of inference, as well as learning them from data. This inherent trade-off is clearly visible in powerful – but intractable – models like Markov random fields, (restricted) Boltzmann machines, (hierarchical) Dirichlet processes, and variational autoencoders. Despite these models' successes, performing inference on them resorts to approximate routines. Moreover, learning such models from data is generally harder as inference is a sub-routine of learning, requiring simplified assumptions or further approximations. Having guarantees on tractability at inference and learning time is therefore a highly desired property in many real-world scenarios.

Tractable graphical models such as SPNs guarantee exactly this: performing exact inference for a range of queries. They compile probabilistic inference routines into efficient computational graphs similar to deep neural networks, but encode a joint probability distribution. As a result, they can not only be used for one ML task, but support many different tasks by design, ranging from outlier detection (joint distribution) to classification or regression (conditional inference). They have been successfully used in numerous real-world applications such as image classification, completion and generation, scene understanding, activity recognition, and language and speech modeling. Despite these successes, it is unclear how one can develop an SPN framework that is GDPR-friendly.

As a naive solution, SPN tasks can be performed only on client devices to ensure that no sensitive information is handed out, but this requires the service provider to ship a trained model to clients, thereby giving up valuable intellectual property and potentially leaking sensitive data as such models often contain traces of sensitive

[1] Cryptography and Privacy Engineering Group, TU Darmstadt, Germany, email addresses: {treiber,weinert,schneider}@encrypto.cs.tu-darmstadt.de
[2] Artificial Intelligence and Machine Learning Lab, TU Darmstadt, Germany, email addresses: {molina,kersting}@cs.tu-darmstadt.de

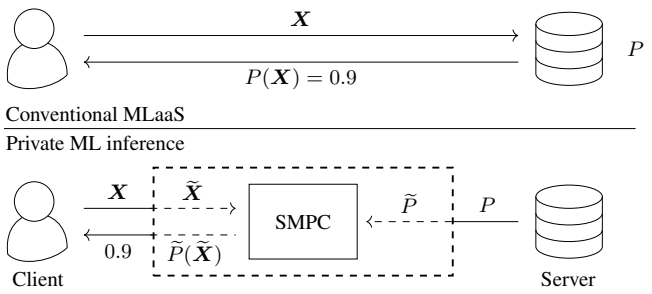

Figure 1. *Conventional* MLaaS (top) vs. *private* ML inference (bottom) on client feature vector $\boldsymbol{X}$ using the server's model P. In private ML inference, the input of both client and server is protected with a cryptographic SMPC protocol: the parties learn only encrypted values $\tilde{\boldsymbol{X}}$ and $\tilde{P}$, respectively.

training data, e.g., due to unintended memorization [7]. Therefore, current "ML as a Service" (MLaaS) applications usually send and hence leak client data to a remote server operated by the service provider to perform inference (cf. top of Figure 1).

Even if the service provider and the remote server are considered trustworthy, the privacy of clients can still be compromised by breaches, hacks, and negligent or malicious insiders. Such incidents occur frequently even at high-profile companies: recently, Microsoft Outlook was hacked [45] and AT&T's customer support was bribed [29]. Thus, it is not enough to protect client data just from outsiders, it must also be hidden from the server to ensure privacy.

Previously, protecting the identity of individuals via anonymization techniques was seen as sufficient when learning on or inferring from data of a collection of users. Such techniques reduce raw data to still enable extraction of knowledge without individuals being identifiable. However, recent works conclude that current de-identification measures are insufficient and unlikely satisfy GDPR standards [41].

Cryptography for ML Applications. We believe this indicates that cryptographic measures should be employed to satisfy today's privacy demands. The cryptographic literature has actively developed protocols and frameworks for efficient and privacy-preserving ML in the past years. So far, efforts were focused on deep/convolutional neural networks, see [38] for a recent systematization of knowledge. There, usually a scenario is considered where the server holds a model and performs *private* ML inference on a client's data, with *no information* except for the inference result being revealed to the client (cf. bottom of Figure 1).

Existing frameworks mostly rely on homomorphic encryption (HE), secure multi-party computation (SMPC), or a combination of both, to enable private inference with various security, resource, and usage properties. As many ML tasks today already require intense computational resources, the overhead incurred by introducing cryptographic privacy mechanisms is substantial. Though a line of prominent frameworks from CryptoNets [15] to XONN [39] has established increased efficiency and relatively low execution times for private inference, research has *mainly focused on NNs* by looking for efficient ways to securely compute common activation functions, sometimes degrading accuracy by using more efficient approximations. Existing frameworks only possess a low degree of automation and often require very low-level model descriptions, making it hard for non-experts to run private inference using their own models. Additionally, for approaches using SMPC, it is very common that the topology of the NN is leaked, which might reveal some model information to the client.

Our Contributions. In this work, we present foundations and tools for privacy-preserving ML in the unexplored domain of sum-product networks (SPNs). Our framework, which we call *CryptoSPN*, demonstrates that SPNs can very well be protected with cryptographic measures. Specifically, after presenting the necessary background for private ML and SPNs (Section 2), we show how to efficiently perform private SPN inference using SMPC (Section 3). We combine techniques from both AI and applied cryptography to achieve this. Contrary to popular SMPC-based approaches for protecting NNs, ours leaks no information from the network topology by using Random Tensorized SPNs (RAT-SPNs) [35]. We implement CryptoSPN using the state-of-the-art SMPC framework ABY [12] and provide an open-source tool that can transform SPN instances from the SPFlow framework [33] into privacy-preserving executables (Section 4). CryptoSPN is easily usable by non-experts and intended to make private ML available to the broader AI community working on a wide range of sophisticated models. In an experimental evaluation (Section 5), we show that CryptoSPN performs private inference in reasonable time while preserving accuracy. With our work, we push private ML beyond NNs and bring attention to the crucial, emerging task of making a *variety* of ML applications private.

2 BACKGROUND

We start with the necessary background on secure computation, existing privacy-preserving ML solutions, and SPNs.

2.1 Secure Computation (SC) of ML Tasks

First described by [46], the concept of secure computation (SC) lets computational parties (e.g., a client and a server) evaluate arbitrary functions on secret inputs without leaking any information but the results. For example, a server can calculate statistics on client data without learning the raw data, or a group of clients can jointly schedule meetings without revealing their availability. The SC research community has put forth efficient schemes with practical implementations for applications that rely on homomorphic encryption (HE) or secure multi-party computation (SMPC). The former allows computations directly on encrypted data, whereas in SMPC an interactive protocol is executed between parties that, in the end, reveals only the desired output. A general rule of thumb is that SMPC requires more communication, whereas computation is the bottleneck for HE. In this work, we rely on secure *two-party* computation, i.e., SMPC with two parties: client and server.

2.1.1 *Privacy-Preserving Machine Learning*

We shortly recapitulate the most influential works for preserving privacy when performing machine learning tasks using SC techniques.

Privacy-preserving neural network inference was first proposed in [34, 42, 3]. Secure classification via hyper-plane decision, naive Bayes, and decision trees was presented in [5]. SecureML [31] provides SMPC-friendly linear regression, logistic regression, and neural network training using SGD as well as secure inference. With CryptoNets [15], the race for the fastest NN-based privacy-preserving image classification began: MiniONN [28], Chameleon [40], Gazelle [20], XONN [39], and DELPHI [30] are only some of the proposed frameworks.

These frameworks mostly offer privacy-preserving deep/convolutional neural network inference based on HE or SMPC

protocols, or even combinations of both techniques in different computational and security models. However, they are not readily accessible to ML experts, as prototype implementations are not well-integrated into ML frameworks and require extensive cryptographic knowledge to secure applications. Moreover, these frameworks are often engineered towards delivering outstanding performance for benchmarks with certain standard data sets (e.g., MNIST [26]), but fail to generalize in terms of accuracy and performance. There are some but very few attempts to directly integrate privacy technology into ML frameworks: for TensorFlow there exists rudimentary support for differential privacy [1], HE [44], and SMPC [10], and for Intel's nGraph compiler there exists an HE backend [4]. Very recenty, Facebook's AI researchers released CrypTen [18], which provides an SMPC integration with PyTorch. However, currently not much is known about the underlying cryptographic techniques and, therefore, its security guarantees.

Trusted execution environments (TEEs) are an intriguing alternative to cryptographic protocols. They use hardware features to shield sensitive data processing tasks. TEEs are widely available, e.g., via Intel Software Guard Extensions (SGX), and therefore are explored for efficiently performing ML tasks [25]. Unfortunately, Intel SGX provides no provable security guarantees and requires software developers to manually incorporate defenses against software side-channel attacks, which is extremely difficult. Moreover, severe attacks on Intel SGX allowed to extract private data from the TEE, making SGX less secure than cryptographic protocols [8].

2.1.2 SMPC

In SMPC, the function f to be computed securely is represented as a Boolean circuit consisting of XOR and AND gates: each gate is securely computed based on the encrypted outputs of preceding gates, and only the values of the output wires of the entire circuit are decrypted to obtain the overall output. The intermediate results leak no information and only the outputs are decrypted by running a corresponding sub-protocol.

The literature considers two security settings: *semi-honest*, where the involved parties are assumed to honestly follow the protocol but want to learn additional information about other parties' inputs, and *malicious*, which even covers active deviations from the protocol. SMPC protocols are usually divided into two phases: a *setup* phase that can be performed independently of the inputs (e.g., during off-peak hours or at night), and an *online* phase that can only be executed once the inputs are known. Most of the "expensive" operations can be performed in the setup phase in advance such that the online phase is very efficient. Two prominent SMPC protocols are Yao's garbled circuit (GC) [46] and the GMW protocol [16]. As we heavily rely on floating-point operations with high-depth circuits, we use Yao's GC protocol, which has a constant round complexity (the round complexity of the GMW protocol depends on the circuit depth and, hence, is not suited for our case).

2.1.3 Yao's GC

We present a schematic overview for the secure evaluation of a single gate in Figure 2 and refer to [27] for further technical details.

The central idea of this protocol is to *encode* the function (more precisely, its representation as a Boolean circuit) and the inputs such that the encoding reveals no information about the inputs but can still be used to evaluate the circuit. This encoding is called "garbling" and individual garbled values can be seen as encryptions. We will use the common notation of $\tilde{g}$ and $\tilde{x}, \tilde{y}$ to refer to a garbled gate or garbled inputs, respectively. The evaluation of the garbled circuit $\tilde{C}$ (consisting of many garbled gates) using the garbled inputs, in turn, results in an encoding $\tilde{z} = \tilde{C}(\tilde{x}, \tilde{y})$ of the output. The encoded output can only be decoded *jointly* to the plain value $z = C(x, y)$, i.e., both parties have to agree to do so. In the protocol, one of the parties, the "garbler", is in charge of creating the garbled circuit. The other party, the "evaluator", obtains the garbled circuit, evaluates it, and then both parties jointly reveal the output.

Circuit Garbling. The wires in the Boolean circuit C of a function f are assigned two *randomly* chosen labels / keys: k_0 and k_1, indicating that the wire's plain value is 0 or 1, respectively. Though there is a label for each possible value $\{0, 1\}$, only the garbled value $\tilde{x} = k_x^w$ for the actual plain value x of wire w is used to evaluate the garbled circuit. The garbler creates both labels and therefore is the only one who knows the mapping to plaintext values – for the evaluator, a single randomly-looking label reveals no information.

The garbler creates a randomly permuted "garbled" gate $\tilde{g}$ in the form of an encrypted truth table for each gate g in the circuit C of f, and sends all garbled gates to the evaluator. The key idea is to use an encryption scheme Enc that has *two* encryption keys. For each truth table entry, the label associated with the plaintext value of the outgoing wire is then encrypted using the labels associated with the plain values of the two incoming wires as encryption keys (cf. Figure 2).

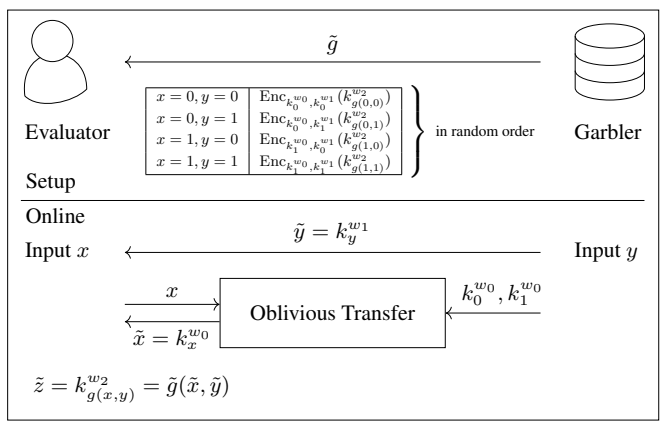

Figure 2. Overview of Yao's GC protocol for securely evaluating a binary gate g (output wire w_2) on binary inputs x (input wire w_0) and y (input wire w_1), where $\tilde{g}$ is the garbled truth table. All labels k_a^w (corresponding to wire w having value a) are generated uniformly at random by the garbler. The protocol is composable and can be used to securely compute any circuit based on the secure computation of one gate.

Garbled Circuit Evaluation. Now, if the evaluator is in possession of the labels $\tilde{x}$ and $\tilde{y}$ corresponding to the incoming wires' values x and y, then exactly one entry of $\tilde{g}$ can be successfully decrypted using $\tilde{x}$ and $\tilde{y}$ as decryption keys[3]. This will result in $\tilde{z} = \tilde{g}(\tilde{x}, \tilde{y})$, the label of the outgoing wire of g associated with the desired plaintext value $z = g(x, y)$. Since only the desired entry can be decrypted and, given that the labels are chosen randomly and independently of the wire values, the evaluator can perform this computation without learning any plaintext information.

[3] A special type of encryption scheme is used to detect if the decryption was successful or not.

The remaining challenge is that the evaluator needs to obtain the correct garbled inputs (i.e., the labels corresponding to its inputs) without revealing the inputs to the garbler. This is solved by a cryptographic protocol called oblivious transfer (OT), which enables one party with input bit b to obliviously obtain a string s_b from another party holding two input strings s_0, s_1 without revealing b and learning anything about s_{1-b}. With this building block, Yao's GC protocol is composed as follows (cf. Figure 2): In the setup phase, the garbler creates all wire labels for the garbled circuit $\tilde{C}$ and sends $\tilde{C}$ to the evaluator. During the online phase, the garbler sends the labels corresponding to its input to the evaluator. The evaluator's garbled inputs are obtained via OT. Then, the evaluator decrypts $\tilde{C}$. The output can be jointly decrypted if the parties reveal the output label associations.

Protocol Costs. Improvements on OT [19, 2] and the garbling scheme [22, 47] have significantly reduced the overhead of Yao's GC protocol, making it viable to be used in applications. Specifically, the GC created and sent in the setup phase requires 2κ bits per binary AND gate, where κ is the symmetric security parameter (e.g., $\kappa = 128$ for the currently recommended security level). For obliviously transferring the labels corresponding to the evaluator's input x, $|x|\kappa$ bits must be sent in the setup phase, as well as $|x|(2\kappa + 1)$ bits in the online phase. Additionally, the labels for the garbler's input y must be sent in the online phase ($|y|\kappa$ bits). The protocol only requires a constant number of rounds of interaction.

2.2 Sum-Product Networks (SPNs)

Recent years have seen a significant interest in tractable probabilistic representations such as Arithmetic Circuits (ACs) [9], Cutset Networks [37], and SPNs [36]. In particular, SPNs, an instance of ACs, are deep probabilistic models that can represent high-treewidth models [48] and facilitate *exact* inference for a range of queries in time *linear* in the network size.

2.2.1 Definition of SPNs

Formally, an SPN is a rooted directed acyclic graph, consisting of *sum*, *product*, and *leaf* nodes. The scope of an SPN is the set of random variables (RVs) appearing on the network. An SPN can be defined recursively as follows: (1) a tractable univariate distribution is an SPN; (2) a product of SPNs defined over different scopes is an SPN; and (3) a convex combination of SPNs over the same scope is an SPN. Thus, a product node in an SPN represents a factorization over independent distributions defined over different RVs $P(\boldsymbol{X}, \boldsymbol{Y}) = P(\boldsymbol{X})P(\boldsymbol{Y})$, while a sum node stands for a mixture of distributions defined over the same variables $P(\boldsymbol{X}, \boldsymbol{Y}) = wP_1(\boldsymbol{X}, \boldsymbol{Y}) + (1 - w)P_2(\boldsymbol{X}, \boldsymbol{Y})$. From this definition, it follows that the joint distribution modeled by such an SPN is a valid normalized probability distribution [36].

2.2.2 Tractable Inference in SPNs

To answer probabilistic queries in an SPN, we evaluate the nodes starting at the leaves. Given some evidence, the probability output of querying leaf distributions is propagated bottom up. For product nodes, the values of the child nodes are multiplied and propagated to their parents. For sum nodes, we sum the weighted values of the child nodes. The value at the root indicates the probability of the asked query. To compute marginals, i.e., the probability of partial configurations, we set the probability at the leaves for those variables to 1 and then proceed as before. This allows us to also compute conditional queries such as $P(\boldsymbol{Y}|\boldsymbol{X}) = P(\boldsymbol{X}, \boldsymbol{Y})/P(\boldsymbol{X})$. Finally, using a bottom-up and top-down pass, we can compute approximate MPE states [36]. All these operations traverse the tree at most twice and therefore can be achieved in linear time w.r.t. the size of the SPN.

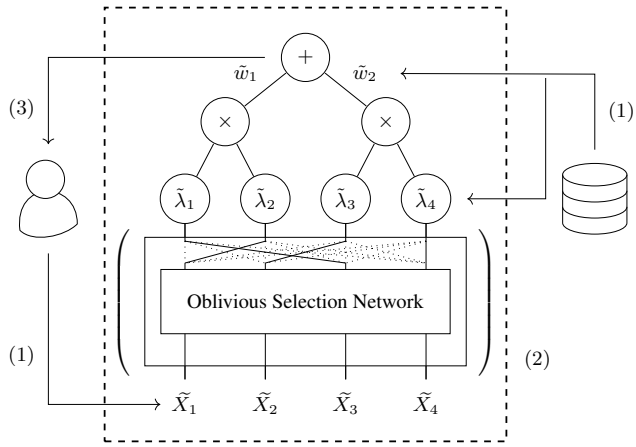

Figure 3. CryptoSPN protocol flow for an exemplary miniature SPN with Poisson leaves: (1) client and server have private inputs $X_{1,\ldots,4}$ and $w_{1,2}, \lambda_{1,\ldots,4}$, respectively; (2) private evaluation of leaf-, sum- and product nodes using SMPC; (3) client receives SPN inference result.

3 CryptoSPN INFERENCE

Given the AC structure of SPNs, SMPC is a fitting mechanism to preserve privacy in SPN inference as it relies on securely evaluating a circuit. Compared to, e.g., NNs, SPNs do not have alternating linear and non-linear layers, which would complicate the application of SMPC protocols. Here, we are concerned with *private* SPN inference in a setting where the client has a private input and the server is in possession of a model; in the end, the server learns nothing, and the client only learns the inference result (cf. bottom of Figure 1). Unfortunately, we cannot use the arithmetic version of the GMW protocol, as it only provides integer or fixed-point operations, which is insufficient for tractable and normalized probabilistic inference such as the case of SPNs. Instead, CryptoSPN uses Yao's GC protocol that evaluates Boolean circuits, which allows us to use floating point operations by including Boolean sub-circuits corresponding to IEEE 754-compliant 32- or 64-bit floating point operations [11] in the circuit representation of the to-be-evaluated SPN.

3.1 Secure Evaluation of SPNs with SMPC

Our approach (cf. Figure 3) is to transform the SPN into a Boolean circuit and to then evaluate it via SMPC. The server input consists of all the model parameters of the SPN (i.e., weights for the weighted sums and parameters for the leaf distribution), the client input consists of the evidence, and the output is the root node value. We perform all computations in the log-domain using the well-known *log-sum-exp* trick, which also provides a runtime advantage for our SMPC approach as it replaces products with more efficient additions. Contrary to the convention, we use the log2 domain in CryptoSPN since the circuits for log2 and exp2 operations are significantly smaller than the natural log and exp operations.

Due to the SMPC security properties, all the model parameters are hidden from the client and the input values or evidence from the

server. However, this naive approach alone does not provide our desired privacy guarantees since the circuit evaluated in SMPC is public. Therefore, the topology of the SPN is leaked to the client, including which RVs (the scope) are used in which leaves. Depending on how the SPN was learned, this might reveal information about the server's model, such as correlations among RVs, number of mixtures, etc. To hide this information, one could make use of generic *private function evaluation* techniques such as incorporating *universal circuits* (UCs) [43]. UCs allow one party to choose a function as the private input, which is then obliviously evaluated on the other party's input such that nothing about the function or the input is revealed. Employing these generic techniques, however, would drastically increase the overhead we introduce via SMPC. For this reason, the related work on SMPC for private NN inference usually assumes that the NN topology is public, with the impact on model privacy being unclear in this situation. To mitigate these concerns in CryptoSPN, we tailor efficient techniques stemming from both AI as well as applied cryptography research specifically to SPNs. The first method hides specifics of the training data by using Random Tensorized SPNs (RAT-SPNs) [35], while the second method allows to hide the scope of any existing SPN without the need to re-learn a RAT-SPN.

3.1.1 Hiding the Training Data

It is possible that the structure of a general SPN leaks information about the training data. To hide any information that could be revealed from the SPN structure, we propose to use RAT-SPNs [35]. The RAT-SPN structure is built randomly via region graphs. Given a set of RVs $\boldsymbol{X}$, a *region* $\mathbf{R}$ is defined as any non-empty subset of $\boldsymbol{X}$. Given any region $\mathbf{R}$, a K-*partition* $\mathcal{P}$ of $\mathbf{R}$ is a collection of K non-overlapping sub-regions $\mathbf{R}_1, \dots, \mathbf{R}_K$, whose union is $\mathbf{R}$, i.e., $\mathcal{P} = \{\mathbf{R}_1, \dots, \mathbf{R}_K\}$, $\forall k\colon \mathbf{R}_k \neq \emptyset$, $\forall k \neq l\colon \mathbf{R}_k \cap \mathbf{R}_l = \emptyset$, $\bigcup_k \mathbf{R}_k = \mathbf{R}$. This partitioning algorithm randomly splits the RVs. Furthermore, we recursively split the regions until we reach a desired partitioning depth. Here, we consider only 2-partitions. From these region graphs, we can construct an SPN specifying the number of uniform leaves per RV. Since the structure-building algorithm is data-agnostic (it only knows the number of RVs in the dataset), there is no information leakage. This also means that any initial random structure for $|\boldsymbol{X}|$, the number of random variables, is a valid initial structure for any other dataset with the same number of dimensions. After obtaining the structure, we use a standard optimization algorithm for parameter estimation. The structure produced by the RAT-SPN algorithm is regular, and the values of the parameters after the optimization encode the knowledge needed to build the joint distribution. In our scheme, the parameters are only visible to the service provider. Using a random structure also enables us to choose the size of the SPNs, which allows service providers to trade off model complexity, efficiency, and accuracy.

3.1.2 Hiding the Scope

Since the scope of a node is defined by the scope of its children, it suffices to hide the leaves' scopes. Concretely, for each leaf, we have to hide which X_j for $j \in \{1, \dots, n\}$ from the client's RVs $\boldsymbol{X} = X_1, \dots, X_n$ is selected. This corresponds to an *oblivious array access* in each leaf, where an array can be accessed without revealing the accessed location j. There exist efficient methods to do this based on homomorphic encryption [6] or secure evaluation of selection networks [23] via SMPC. A recent study of private decision tree evaluation [21] shows that selection networks outperform selection based on HE in both total and online runtime. Hence, we obliviously select RVs via securely evaluating a selection network in CryptoSPN.

Similar to the usage in decision trees, we add just one selection network below the SPN instead of selecting one variable per leaf. That is, the variable input of the secure leaf computation (see below) is the outcome of the selection network, which selects the variables $X_{\phi(1)}, \dots, X_{\phi(m)}$ for the m leaves in the SPN from the n client inputs according to a server input $\phi\colon [m] \mapsto [n]$ denoting which leaf $i \leq m$ uses $X_{\phi(i)}$. If $m \geq n$ (which we assume is true since RVs are usually used more than once), the complexity of such a selection network is [23]:

$$C^{\text{sel}}_{n,m} = \frac{1}{2}(n+m)\log_2(n) + m\log_2(m) - n + 1,$$

beating the trivial solution of $O(nm)$. This requires $|X|\kappa(n + 2C^{\text{sel}}_{n,m})$ bits of setup communication and $|X|n(2\kappa+1)$ bits online [21]. Hereby, one can hide the scope of any SPN, including ones learned through other methods [13] (although the topology is still leaked). We propose to use this approach to increase privacy in cases where leaking the topology is deemed acceptable, or where relearning the structure of an already existing SPN is infeasible.

3.1.3 Hiding the RVs and Leaf Parameters

Because the secure computation of each floating point operation introduces overhead, our approach at the leaf level is to let the respective parties locally pre-compute as many terms as possible before inputting them into the secure SPN evaluation. For Gaussians in the log2 domain, the result can be evaluated in SMPC with just two multiplications and two additions based on the client's RV input X_j and server inputs μ, $s_1 = -\log_2(2\pi\sigma^2)$, and $s_2 = \frac{\log_2(e)}{2\sigma^2}$ based on parameters mean μ and variance σ^2:

$$\widetilde{P}_{\text{Gauss}}\left(X_j; \mu, \sigma^2\right) = \tilde{s}_1 - \left(\widetilde{X}_j - \tilde{\mu}\right)^2 \tilde{s}_2.$$

Thus, for each leaf, the SPN circuit requires $C^{\text{Gauss}}_b = 2(C^{\text{ADD}}_b + C^{\text{MUL}}_b)$ AND gates, where C^{OP}_b denotes the number of AND gates for a b-bit floating point operation OP, cf. [11][4]. Additionally, for the entire SPN, $IC^{\text{Gauss}}_b = nb$ bits of client input and $IS^{\text{Gauss}}_b = 3mb$ bits of server input are added, where m is the amount of leaves and n is the number of RVs.

Similarly, we can securely compute Poissons with just one multiplication and two additions based on the client's RV inputs X_j and $c_1 = -\log_2(X_j!)$, and server inputs $s_1 = \log_2(\lambda)$ and $s_2 = -\lambda\log_2(e)$ based on mean λ:

$$\widetilde{P}_{\text{Pois}}\left(X_j; \lambda\right) = \widetilde{X}_j \cdot \tilde{s}_1 + \tilde{c}_1 + \tilde{s}_2.$$

This results in the leaf size $C^{\text{Pois}}_b = 2C^{\text{ADD}}_b + C^{\text{MUL}}_b$ with input sizes $IC^{\text{Pois}}_b = 2nb$ and $IS^{\text{Pois}}_b = 2mb$ for the entire SPN.

Bernoullis consist of just one MUX gate, selecting from two server inputs p and $q = 1 - p$ based on the binary client RV input X_j:

$$\widetilde{P}_{\text{Bern}}\left(X_j; p\right) = \text{MUX}\left(\widetilde{X}_j, \tilde{p}, \tilde{q}\right) = \begin{cases} \tilde{p}, & \text{if } X_j = 1 \\ \tilde{q}, & \text{if } X_j = 0 \end{cases}.$$

Hence, they have a complexity of $|p|$ AND gates, yielding the costs $C^{\text{Bern}}_b = b$, $IC^{\text{Bern}}_b = n$, and $IS^{\text{Bern}}_b = 2mb$.

Due to the log2 domain, computations of a product node just introduce a complexity of $(ch(s) - 1)C^{\text{ADD}}_b$, where $ch(s)$ denotes the

[4] For instance, $C^{\text{ADD}}_{32} = 1820, C^{\text{MUL}}_{32} = 3016, C^{\text{EXP2}}_{32} = 9740$, and $C^{\text{LOG2}}_{32} = 10568$.

amount of children of a node s. For the same reason, the complexity of a sum node is:

$$C_b^{\text{LOG2}} + (ch(s)-1)C_b^{\text{ADD}} + ch(s)(C_b^{\text{ADD}} + C_b^{\text{EXP2}}).$$

3.2 Efficiency

Putting all of the presented building blocks together, we get the following amount of AND gates (the only relevant cost metric for Yao's GC protocol) for an SPN with n RVs and m leaves of distribution $D \in \{\text{Gauss}, \text{Pois}, \text{Bern}\}$ that operates with b-bit precision and consists of a set of sum nodes $\boldsymbol{S}$ and product nodes $\boldsymbol{P}$, where $ch(s)$ for $s \in \boldsymbol{S} \cup \boldsymbol{P}$ denotes the amount of children of node s:

$$\begin{aligned} C_b^{\text{SPN}} =& mC_b^D + \sum_{s\in\boldsymbol{S}} \Big(C_b^{\text{LOG2}} + (ch(s)-1)C_b^{\text{ADD}} \\ &+ ch(s)\left(C_b^{\text{ADD}} + C_b^{\text{EXP2}}\right)\Big) + \sum_{p\in\boldsymbol{P}} (ch(p)-1)\, C_b^{\text{ADD}}. \end{aligned}$$

In addition, we also have IC_b^D client input bits stemming from the RVs and $IS_b^D + b\sum_{s\in\boldsymbol{S}} ch(s)$ server input bits stemming from the leaf parameters as well as the sum weights. Therefore, using Yao's GC protocol, CryptoSPN has the following communication costs in bits in the *setup phase*:

$$\kappa\left(IC_b^D + 2C_b^{\text{SPN}}\right)$$

and in the *online phase*:

$$\kappa\left(2IC_b^D + IS_b^D + b\sum_{s\in\boldsymbol{S}} ch(s)\right) + IC_b^D,$$

where κ is the symmetric security parameter (e.g., $\kappa = 128$).

If one does not use RAT-SPNs and instead our scope-hiding private SPN evaluation, obliviously selecting the leaves' RVs for Gaussian, Poisson, and Bernoulli leaves has the following online communication, respectively: $nb(2\kappa+1)$, $2nb(2\kappa+1)$, and $n(\kappa+1)$ bits. The setup communication is $\kappa b\left(n + 2C_{n,m}^{\text{sel}}\right)$, $\kappa b\left(2n + 2C_{2n,m}^{\text{sel}}\right)$, and $\kappa\left(n + 2C_{n,m}^{\text{sel}}\right)$ bits, respectively.

3.3 Security Guarantees

As we use RAT-SPNs, the underlying structure, the size and depth of the SPN are known to the client. However, this structure is randomly generated and comes only from hyper-parameters. Therefore, the structure is independent of training data and leaks no private information. The number of random variables is known by both parties, as usual (e.g., [15, 20, 28, 30, 39, 40]). And while the value of the input variables is hidden, the output is not and might reveal some information but is data that *inherently* has to be revealed.

The protocols we use in our implementation of CryptoSPN are provably secure in the semi-honest model [27]. In the studied setting, it is reasonable to assume the server is semi-honest, as reputable service providers are confined by regulations and potential audits. Furthermore, detected malicious behaviour would hurt their reputation, providing an economic incentive to behave honestly. However, these regulations and incentives do not exist for the client's device, which can be arbitrarily modified by the client or harmful software.

Fortunately, CryptoSPN can easily be extended to provide security against malicious clients as it relies on Yao's GC protocol. There, the only messages sent by the (potentially malicious) client are in the oblivious transfer. Thus, one just needs to instantiate a maliciously secure OT protocol to achieve security against malicious clients, which incurs only a negligible performance overhead [2].

4 IMPLEMENTATION & SPFlow INTEGRATION

We implemented CryptoSPN using the state-of-the-art SMPC framework ABY [12] with the floating point operation sub-circuits of [11] and the selection network circuit of [21]. ABY implements various SMPC protocols in C++ and provides APIs for the secure evaluation of supplied circuits within these protocols. It also supports *single instruction, multiple data* (SIMD) instructions, which allows CryptoSPN to batch-process multiple queries at the same time. Notably, like most other SMPC frameworks, ABY requires a very low-level circuit description of the function that is computed securely, making it hard for AI researchers and others without a background in cryptography to actually perform private ML inference. Motivated by this gap, we integrate CryptoSPN with SPFlow [33], an open-source Python library that provides an interface for SPN learning, manipulation, and inference. For users, CryptoSPN appears as another SPFlow export that enables private SPN inference. Specifically, CryptoSPN allows ML experts to easily transform an SPN in SPFlow into a privacy-preserving ABY program with just the SPN as input. The resulting ABY program can be compiled into an executable for simple deployment on the client and server side. CryptoSPN is available at `https://encrypto.de/code/CryptoSPN`.

5 EXPERIMENTAL EVALUATION

We evaluate CryptoSPN on random SPNs trained with SPFlow for the standard datasets provided in [13], and on regular SPNs for `nips`, a count dataset from [32]. We evaluate models with both 32- and 64-bit precision to study the trade-off between accuracy and efficiency. 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.

Our benchmarks are given in Table 1. Compared to previous works focused on NNs, we evaluate a variety of datasets, which shows that CryptoSPN can easily transform any SPN into a privacy-preserving version. In addition to the theoretical analysis of Section 3.2, we also investigate RAT-SPNs of various sizes for the `nltcs` dataset of [13] to gain a practical sense of how different SPN parameters affect our runtime. Moreover, we use two *regular* SPNs trained for `nips` to see how hiding the scope (cf. Section 3.1.2) increases the runtime.

Generally, our results shown in Table 1 demonstrate that we achieve tractable setup and highly efficient online performance for medium-sized SPNs. Specifically, the setup phase requires costs in the order of minutes and gigabytes, while the online phase takes only a few seconds and megabytes. Though multiple seconds might seem like a significant slow-down in some cases, this is certainly justified in many scenarios where privacy demands outweigh the costs of privacy protection (such as legal requirements for medical diagnostics).

While no single parameter appears to be decisive for the runtimes, we observe that some parameters are much more significant:

1. The number of sums has a significantly larger effect than products or leaves, which is expected given the log2 and exp2 operations. But, since the absolute amount of sums is still relatively small, the additional input weights do not affect online communication.
2. Though differences in the number of RVs, product nodes, leaves, and edges do influence the runtimes, deviations have to be very large to take an effect. For instance, when examining the SPNs

Table 1. Benchmarks of private SPN inference with CryptoSPN in a WAN network. The SPN has $|\boldsymbol{X}|$ RVs, $|\boldsymbol{S}|$ sum nodes, and $|\boldsymbol{P}|$ product nodes. Setup and online runtime as well as communication are measured for both 32- and 64-bit precision. All SPNs are RAT-SPNs with Bernoulli leaves except the ones for `nips`, which are regular SPNs with Poisson leaves. † indicates usage of a selection network for hiding RV assignments.

dataset	$\lvert\boldsymbol{X}\rvert$	$\lvert\boldsymbol{S}\rvert$	$\lvert\boldsymbol{P}\rvert$	#leaves	#edges	#layers	setup (s)		setup (GB)		online (s)		online (MB)	
							32b	64b	32b	64b	32b	64b	32b	64b
accidents	111	22	4420	11 100	27 161	7	365	825	4.34	9.83	22.5	55.5	15.5	30.9
baudio	100	22	4420	10 000	26 061	7	359	812	4.28	9.67	22.1	53.6	14.4	28.7
bbc	1058	2	880	42 320	44 721	5	248	577	2.95	6.87	18.6	41.9	43.8	87.5
bnetflix	100	2	4400	20 000	32 001	5	264	604	3.15	7.20	17.1	40.8	22.5	45.1
book	500	2	880	20 000	22 401	5	134	312	1.60	3.71	9.8	22.3	20.9	41.8
c20ng	910	2	880	36 400	38 801	5	218	524	2.59	6.03	16.4	37.6	37.7	75.4
cr52	889	10	1768	35 560	41 985	7	304	701	3.62	8.34	21.1	49.1	38.1	76.1
cwebkb	839	10	1768	33 560	39 985	7	294	677	3.50	8.06	20.6	46.8	36.0	72.0
dna	180	22	4420	18 000	34 061	7	400	907	4.76	10.81	24.4	59.4	22.5	45.1
jester	100	2	4400	20 000	32 001	5	264	604	3.15	7.20	17.1	40.9	22.5	45.1
kdd	64	10	1768	2560	8985	7	137	308	1.62	3.67	8.5	20.8	4.3	8.5
kosarek	190	2	2200	19 000	25 001	5	178	410	2.12	4.87	12.3	28.7	20.5	41.0
msnbc	17	10	1768	680	7105	7	127	285	1.51	3.40	8.0	19.1	2.3	4.7
msweb	294	22	4420	29 400	45 461	7	458	1042	5.45	12.42	29.1	69.0	34.2	68.4
plants	69	2	4400	13 800	25 801	5	233	531	2.77	6.32	14.7	35.3	16.2	32.4
pumsb_star	163	2	4400	32 600	44 601	5	328	754	3.91	8.98	22.1	52.0	35.4	70.9
tmovie	500	10	1768	20 000	26 425	7	225	515	2.68	6.14	14.9	35.2	22.1	44.3
tretail	135	2	4400	27 000	39 001	5	300	688	3.57	8.19	19.8	47.2	29.7	59.4
nltcs	16	2	880	640	3041	5	36	81	0.43	0.96	2.7	5.8	1.1	2.1
	16	2	2200	1600	7601	5	90	202	1.07	2.41	5.9	13.8	2.7	5.4
	16	2	4400	3200	15 201	5	179	404	2.13	4.82	11.3	27.4	5.3	10.7
	16	10	1768	640	7065	7	127	285	1.51	3.39	8.0	19.6	2.3	4.6
	16	22	4420	1600	17 661	7	316	712	3.77	8.48	19.3	47.7	5.7	11.5
nips	100	7	17	1061	1084	11	26	71	0.30	0.83	2.1	5.0	1.3	2.6
							$29^{\dagger}$	$76^{\dagger}$	$0.33^{\dagger}$	$0.89^{\dagger}$	$2.2^{\dagger}$	$5.3^{\dagger}$	$1.8^{\dagger}$	$3.1^{\dagger}$
	100	15	43	2750	2807	15	66	182	0.77	2.16	4.6	12.4	3.0	6.1
							$73^{\dagger}$	$196^{\dagger}$	$0.85^{\dagger}$	$2.33^{\dagger}$	$4.8^{\dagger}$	$12.9^{\dagger}$	$4.4^{\dagger}$	$7.4^{\dagger}$

for `accidents`, `baudio`, and `msweb`, it takes roughly twice the amount of RVs and edges (the SPN for `msweb`) compared to the others to reach a significant runtime deviation.

3. When looking at the SPNs for `nltcs`, the first three SPNs have roughly the same density and the runtime seems to scale according to their size. The last two SPNs, however, have a noticeably higher density but comparable size and result in much higher runtimes. Thus, density (especially the amount of edges) is a much more significant parameter than plain network size.

Yet, depending on the SPN, the costs of other, less important parameters can outweigh the costs of individual parameters. This is in line with our theoretical analysis in Section 3.2: the circuit's size depends on the number of children (with different costs for sums and products) as well as the number of RVs and leaves. The amount of layers has no direct effect because the round complexity of Yao's GC protocol is independent of the depth. As for the regular SPNs for `nips`, one can observe that the effects of hiding RV assignments are insignificant compared to the overall performance.

Using 64-bit precision roughly doubles the costs of 32-bit precision, which is expected as the sub-circuits are about twice the size [11]. Comparing the difference of the resulting log-probabilities when evaluating the SPNs in CryptoSPN to the plain evaluation with SPFlow, we get an RMSE of 4.2×10^{-9} for 32-bit and 2.3×10^{-17} for 64-bit models. We stress that this insignificant loss in accuracy is not due to the cryptographic measures, but rather due to the more SMPC-friendly computation in the log2 domain.

6 CONCLUSION

Resolving privacy issues in ML applications is becoming a challenging duty for researchers, not least due to recent legal regulations such as the GDPR. By combining efforts from both AI and applied cryptography research, we presented CryptoSPN, which successfully addresses this challenge for the evaluation of sum-product networks (SPNs) that support a wide variety of desired ML tasks. The protocols of CryptoSPN together with the tools developed for ML experts deliver efficient yet extremely accurate SPN inference while providing unprecedented protection guarantees that even cover the network scope and structure. With our work serving as a foundation, future research can investigate further efficiency improvements (e.g., via quantization techniques appropriate for SPNs), hiding the structure of SPNs that cannot be re-trained, and private SPN learning.

ACKNOWLEDGEMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (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 BMBF and HMWK within ATHENE. KK also acknowledges the support of the Federal Ministry of Education and Research (BMBF), grant number 01IS18043B "MADESI".

REFERENCES

[1] Galen Andrew, Steve Chien, and Nicolas Papernot. TensorFlow privacy. https://github.com/tensorflow/privacy, 2019.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner, 'More efficient oblivious transfer extensions', *Journal of Cryptology*, (2017).

[3] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider, 'Privacy-preserving ECG classification with branching programs and neural networks', *IEEE Transactions on Information Forensics and Security (TIFS)*, (2011).

[4] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski, 'nGraph-HE2: A high-throughput framework for neural network inference on encrypted data', in *Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, (2019).

[5] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser, 'Machine learning classification over encrypted data', in *Network and Distributed System Security Symposium (NDSS)*, (2015).

[6] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel, 'Privacy-preserving remote diagnostics', in *ACM Conference on Computer and Communications Security (CCS)*, (2007).

[7] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song, 'The secret sharer: Evaluating and testing unintended memorization in neural networks', in *USENIX Security*, (2019).

[8] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai, 'SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution', in *IEEE European Symposium on Security and Privacy (EuroS&P)*, (2019).

[9] Arthur Choi and Adnan Darwiche, 'On relaxing determinism in arithmetic circuits', in *ICML*, (2017).

[10] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma, 'Private machine learning in TensorFlow using secure computation', *arXiv preprint arXiv:1810.08130*, (2018).

[11] Daniel Demmler, Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, and Shaza Zeitouni, 'Automated synthesis of optimized circuits for secure computation', in *ACM Conference on Computer and Communications Security (CCS)*, (2015).

[12] Daniel Demmler, Thomas Schneider, and Michael Zohner, 'ABY-A framework for efficient mixed-protocol secure two-party computation', in *Network and Distributed System Security Symposium (NDSS)*, (2015).

[13] Robert Gens and Pedro M Domingos, 'Learning the structure of sum-product networks', in *ICML*, (2013).

[14] Zoubin Ghahramani, 'Probabilistic machine learning and artificial intelligence', *Nature*, (2015).

[15] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E Lauter, Michael Naehrig, and John Wernsing, 'Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy', in *ICML*, (2016).

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson, 'How to play any mental game', in *STOC*, (1987).

[17] Bryce Goodman and Seth Flaxman, 'European Union regulations on algorithmic decision-making and a "right to explanation"', *AI Magazine*, (2017).

[18] David Gunning, Awni Hannun, Mark Ibrahim, Brian Knott, Laurens van der Maaten, Vinicius Reis, Shubho Sengupta, Shobha Venkataraman, and Xing Zhou. 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/, 2019.

[19] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank, 'Extending oblivious transfers efficiently', in *CRYPTO*, (2003).

[20] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan, 'GAZELLE: A low latency framework for secure neural network inference', in *USENIX Security*, (2018).

[21] Ágnes Kiss, Masoud Naderpour, Jian Liu, N Asokan, and Thomas Schneider, 'SoK: Modular and efficient private decision tree evaluation', *Privacy Enhancing Technologies (PETs)*, (2019).

[22] Vladimir Kolesnikov and Thomas Schneider, 'Improved garbled circuit: Free XOR gates and applications', in *International Colloquium on Automata, Languages, and Programming (ICALP)*, (2008).

[23] Vladimir Kolesnikov and Thomas Schneider, 'A practical universal circuit construction and secure evaluation of private functions', in *Financial Cryptography and Data Security (FC)*, (2008).

[24] Daphne Koller and Nir Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, 2009.

[25] Roland Kunkel, Do Le Quoc, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer, 'TensorSCONE: A secure TensorFlow framework using Intel SGX', *arXiv preprint arXiv:1902.04413*, (2019).

[26] Yann LeCun, Corinna Cortes, and Christopher Burges. MNIST Dataset. http://yann.lecun.com/exdb/mnist/, 1998.

[27] Yehuda Lindell and Benny Pinkas, 'A proof of security of Yao's protocol for two-party computation', *Journal of Cryptology*, (2009).

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

[29] Louise Matsakis. How AT&T insiders were bribed to 'unlock' millions of phones. https://www.wired.com/story/att-insiders-bribed-unlock-phones/, 2019.

[30] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa, 'DELPHI: A cryptographic inference service for neural networks', in *USENIX Security*, (2020).

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

[32] Alejandro Molina, Sriraam Natarajan, and Kristian Kersting, 'Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions', in *AAAI*, (2017).

[33] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting, 'SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks', *arXiv preprint arXiv:1901.03704*, (2019).

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

[35] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani, 'Random sum-product networks: A simple and effective approach to probabilistic deep learning', in *UAI*, (2019).

[36] Hoifung Poon and Pedro M Domingos, 'Sum-product networks: A new deep architecture', in *UAI*, (2011).

[37] Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate, 'Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of Chow-Liu trees', in *ECML PKDD*, (2014).

[38] M Sadegh Riazi, Bita Darvish Rouhani, and Farinaz Koushanfar, 'Deep learning on private data', *IEEE Security and Privacy Magazine*, (2019).

[39] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar, 'XONN: XNOR-based oblivious deep neural network inference', in *USENIX Security*, (2019).

[40] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar, 'Chameleon: A hybrid secure computation framework for machine learning applications', in *ACM ASIA Conference on Computer and Communications Security (ASIACCS)*, (2018).

[41] Luc Rocher, Julien M Hendrickx, and Yves-Alexandre de Montjoye, 'Estimating the success of re-identifications in incomplete datasets using generative models', *Nature Communications*, (2019).

[42] Ahmad-Reza Sadeghi and Thomas Schneider, 'Generalized universal circuits for secure evaluation of private functions with application to data classification', in *International Conference on Information Security and Cryptology (ICISC)*, (2008).

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

[44] Tim van Elsloo, Giorgio Patrini, and Hamish Ivey-Law, 'SEALion: A framework for neural network inference on encrypted data', *arXiv preprint arXiv:1904.12840*, (2019).

[45] Tom Warren. Microsoft admits Outlook.com hackers were able to access emails. https://theverge.com/2019/4/15/18311112/microsoft-outlook-web-email-hack-response-comment, 2019.

[46] Andrew Yao, 'How to generate and exchange secrets', in *FOCS*, (1986).

[47] Samee Zahur, Mike Rosulek, and David Evans, 'Two halves make a whole', in *EUROCRYPT*, (2015).

[48] Han Zhao, Mazen Melibari, and Pascal Poupart, 'On the relationship between sum-product networks and Bayesian networks', in *ICML*, (2015).