An End-to-End Framework for Private DGA Detection as a Service

Ricardo J. M. Maia,[1] Dustin Ray,[2] Sikha Pentyala,[2] Rafael Dowsley,[3] Martine De Cock,[2,4] Anderson C. A. Nascimento,[5] Ricardo Jacobi[1]

***Abstract*—Domain Generation Algorithms (DGAs) are used by malware to generate pseudorandom domain names to establish communication between infected bots and Command and Control servers. While DGAs can be detected by machine learning (ML) models with great accuracy, offering DGA detection as a service raises privacy concerns when requiring network administrators to disclose their DNS traffic to the service provider. We propose the first end-to-end framework for privacy-preserving classification as a service of domain names into DGA (malicious) or non-DGA (benign) domains. We achieve this through a combination of two privacy-enhancing technologies (PETs), namely secure multi-party computation (MPC) and differential privacy (DP). Through MPC, our framework enables an enterprise network administrator to outsource the problem of classifying a DNS domain as DGA or non-DGA to an external organization without revealing any information about the domain name. Moreover, the service provider's ML model used for DGA detection is never revealed to the network administrator. Furthermore, by using DP, we also ensure that the classification result cannot be used to learn information about individual entries of the training data. Finally, we leverage the benefits of quantization of deep learning models in the context of MPC to achieve efficient, secure DGA detection. We demonstrate that we achieve a significant speed-up resulting in a 15% reduction in detection runtime without reducing accuracy.**

***Index Terms*—domain generation algorithm (DGA), machine learning, neural network, secure multi-party computation, differential privacy**

I. Introduction

Malicious software (malware) is the class of software that infects computers to perform unauthorized actions in the system or gain unauthorized access to information. Malware is a highly significant source of illicit activities with increasing impact [1], [2], [3], [4], and substantial losses have been sustained in sectors such as the government, energy, and manufacturing [5]. Examples of malware families include trojan horses, viruses, ransomware, key loggers, worms, spyware, and hidden cryptominers. Some common objectives of these types of malware are information or identity theft, espionage, and service disruption [1], [2].

Botnets, i.e. computer networks infected by malware, are commonly controlled, operated, and updated through communicating with a Command and Control (C&C) server that is under the control of an adversary or botmaster [6]. When the IP address of the C&C server is hard-coded directly into the malware, intrusion detection systems (IDS) or firewalls on Domain Name System (DNS) servers can blacklist the detected malicious domain names and block the connection to the C&C server, effectively rendering the malware useless. Cyber-attackers have therefore adopted innovative techniques for obfuscating and concealing the C&C's identity. Among the most prevalent approaches are Domain Generation Algorithms (DGA) [7].

DGAs are algorithms that periodically generate pseudorandom combinations of characters or words to form hundreds or even thousands of new domain names. The key idea is that DGAs can generate the same set of new domain names when executed by two different machines, such as by a botmaster and on an infected machine. The botmaster registers one generated domain name, while the infected machines systematically query the domains from the generated list until one of them is resolved. The domains from the list that have not been registered by the botmaster will typically result in a non-existent domain response when queried, and can be discarded by the infected machine. Once an infected machine queries the registered domain name, communication between the infected bot and the C&C center is established, and malicious activities as instructed by the C&C center can be performed by the bot. The constant changes to the domain name of the C&C server make it much more difficult for IDS and firewalls to detect and contain the attacks. The challenge of mitigating attacks that use DGA techniques lies in identifying malicious domain names. The DNS server must be able to detect and block malicious domain names while keeping a normal operation for benign domain names. In short, the ability to identify malicious domains can drastically decrease the harm caused by malware.

Using machine learning (ML) models to create classifiers that can identify and separate benign domains from DGA-generated malware domains is viable (see [8] and references therein). Such classifiers (models) can be deployed as automatic malware detection systems in enterprise networks. The state-of-the-art models use deep learning techniques that achieve high accuracy but require large amounts of training data [9]. Due to this data demand, models are usually available with third-party organizations (service providers) as a DGA detection service where the DNS traffic of an enterprise is sent to the service provider who then classifies the incoming traffic into malicious or benign domains and sends the classification result to the enterprise [10]. Such an outsourced DGA detec-

[1] University of Brasília. R. Maia ricardo.menezes@aluno.unb.br, R. Jacobi jacobi@unb.br

[2] University of Washington Tacoma D. Ray dustiv2@uw.edu, S. Pentyala sikha@uw.edu, M. De Cock mdecock@uw.edu

3 Monash University. R. Dowsley rafael.dowsley@monash.edu

4 Ghent University. M. De Cock martine.decock@ugent.be

5 Visa Research (Work done while at UW) annascim@visa.com

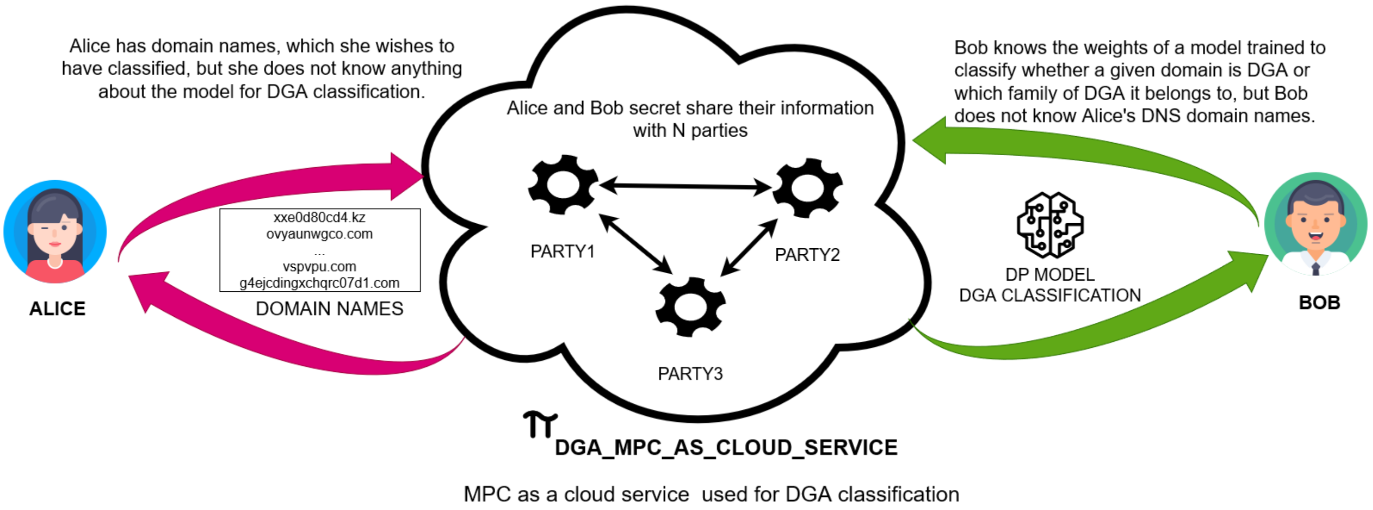
Alice has domain names, which she wishes to have classified, but she does not know anything about the model for DGA classification.
Alice and Bob secret share their information with N parties
Bob knows the weights of a model trained to classify whether a given domain is DGA or which family of DGA it belongs to, but Bob does not know Alice's DNS domain names.
xxe0d80cd4.kz
ovyaunwgco.com
...
vspvpu.com
g4ejcdingxchqrc07d1.com
DOMAIN NAMES
ALICE
PARTY1
PARTY2
PARTY3
DP MODEL
DGA CLASSIFICATION
BOB
DGA_MPC_AS_CLOUD_SERVICE
MPC as a cloud service used for DGA classification

Fig. 1. Flow diagram illustrating end-to-end privacy-preserving DGA detection as a service. The DNS domain name classifier of the service provider (Bob) is trained with DP-SGD to provide differential privacy (DP) guarantees (output privacy). New domain names coming from Alice are classified with Bob's model by secure multi-party (MPC) protocols executed by MPC servers in the cloud over encrypted data (input privacy).

tion as a service model presents numerous privacy challenges. The DNS traffic of an enterprise holds sensitive information that can impact the privacy of all the enterprise network users, which raises privacy concerns for the users in the above DGA detection as a service paradigm. A potential solution to address this concern is to make the ML model used for DGA detection available to enterprise network administrators so that it can be deployed locally. This is, again, problematic as the model is proprietary to the service provider. Moreover, the data used for training such models can be private, and releasing the model to the enterprises renders the underlying training data vulnerable to attacks [11], [12]. Simultaneously protecting the sensitive data of the enterprise and the service provider is a significant challenge. Our paper proposes an end-to-end privacy-preserving framework for outsourced DGA classification that preserves enterprise users' and service providers' privacy.

Our Contributions. We propose a novel framework that provides the benefits of automated and outsourced DGA detection while preserving the privacy of enterprise network users' and DGA detection service providers' data. In our framework, neither the DNS traffic is released in the clear to the DGA detection service providers nor is the model made available to the enterprise network administrators (also, no private data is revealed to any other party). Furthermore, we incorporate techniques that prevent attackers from gaining additional information about the training data or reconstructing the model from the classification results sent to the enterprise network administrators.

Fig. 1 illustrates our framework at a high level. *Alice* represents the enterprise and holds the DNS domains and traffic. *Bob* represents the service provider and has the weights (parameters) of the trained machine learning model. *Bob* can choose to train a multi-layer perceptron (MLP), a one-dimensional convolutional network (1D-CNN), or a recurrent neural network model (LSTM), and the chosen model shall be trained with differential privacy (DP) guarantees [13]. Furthermore, *Alice* wants her DNS traffic classified by *Bob*. We hereafter refer to the *Alice's* DNS data and *Bob's* model weights as private data.

Our proposed framework employs techniques from secure multi-party computation (MPC) [14] to preserve input privacy. To this end, *Alice* and *Bob* secret share their private data with a set of untrusted computational servers (parties). The MPC servers perform computations on the secret shares to label domain names as benign or malicious in a way such that:

(P1) No individual MPC server should obtain any information about *Alice's* domain names;
(P2) No individual MPC server should learn any information about the weights of *Bob's* machine learning model;
(P3) The result of the classification should be revealed only to *Alice*;
(P4) The result of the classification should reveal no private information about individual entries in *Bob's* training data set to *Alice*.

We note that (P4) provides *output privacy* and is achieved as *Bob* has trained the model with DP guarantees. (P1)–(P3) provide *input privacy* and are achieved through the novel MPC protocols that we propose for inference with a neural network model trained for DGA detection. MPC protocols usually lead to high communication and computation costs, thus impacting the inference runtime and overall performance. To improve the performance of our proposed MPC protocols for secure classification of DGA domains, we leverage the benefits of quantization schemes available in TensorFlow (TFLite).[1] We propose that *Bob* uses post-training quantization techniques on

[1] Quantization works by reducing the precision of the numbers used to represent a model's parameters, which by default are 32-bit floating point numbers.

the DP-trained model[2] prior to using the model for classification.

The closest related work [15] to ours on privacy-preserving detection of DGAs (see Section VI for more details) provides only input privacy through MPC, leaving the inference phase vulnerable to privacy attacks on the training data. We propose the first end-to-end privacy-preserving framework for DGA classification, ensuring input and output privacy. We summarize our contributions below:

- We propose a novel framework for private classification as a service of domains into DGA/non-DGA, with input privacy (MPC) and output privacy (DP) guarantees.
- Our proposed framework is the first that considers differentially private training of models for DGA classification.
- Our proposed solution works with classifiers based on multilayer perceptrons (MLP), 1D convolutional neural networks (1D-CNN), and long short-term memory networks (LSTM). Our MPC protocol for LSTM is novel, efficient, and the first ever implemented in the MP-SPDZ MPC framework [16].
- We evaluate our framework on real datasets – DGArchive and Alexa – for binary and multiclass domain name classification tasks. The binary classification problem distinguishes a domain into benign and malicious. The multiclass classification problem also outputs a malicious domain's corresponding DGA family.
- We empirically analyze the privacy and utility tradeoffs of our approach when using MLP, 1D-CNN, and LSTM. We observe that providing output privacy degrades the accuracy by a small amount in our experiments due to the noise introduced to provide DP guarantees. The use of our MPC protocols to provide input privacy on the other hand does not degrade the utility of the classification model.
- We demonstrate the efficiency of our proposed solution in terms of runtime with 2 or 3 computing parties. With the usage of quantization, we observe significant improvements in the performance of our MPC protocols, leading to reduced communication rounds and inference time. Our experiments show a 15% improvement in inference time without affecting accuracy, demonstrating that we can achieve near real-time secure detection of DGA domains.

II. Preliminaries

A. Domain Generation Algorithms

A Domain Generation Algorithm (DGA) is an algorithm that generates artificial malicious domain names. DGAs play a vital role in malware that relies on network communication between a botmaster and the bots (infected clients) [7], [9], [10]. As illustrated in Fig. 2, the key idea is for a botmaster and for malware on infected bots to independently run the same DGA with the same seed (e.g. based on the date) to generate the same list of artificial domains. The botmaster subsequently registers one or more of these automatically generated domains, while the malware on the bots attempts to resolve each domain with the DNS.

In Fig. 2, for example, the botmaster and the malware on the bots all use the DGA "xxhex" to generate a list of domain names (pseudorandom strings of alphanumerical characters): "xxd80cd4e0.cn", "xxe0d80cd4.kz", etc. Out of these, the botmaster registers "xxe0d80cd4.kz" as the domain related with IP "189.6.29.252". The malware on an infected client will then attempt to resolve each domain with the DNS. In the example in Fig. 2, for domain "xxd80cd4e0.cn", the DNS returns a Non-Existent Domain (NXDOMAIN or NXD) error, which indicates the DNS has no listing for "xxd80cd4e0.cn". In contrast, for domain "xxe0d80cd4.kz", the DNS returns IP "189.6.29.252". Finally, the malware connects with IP "189.6.29.252", where the command and control (C&C) server is registered. Communication then proceeds between the botmaster and the infected machine.

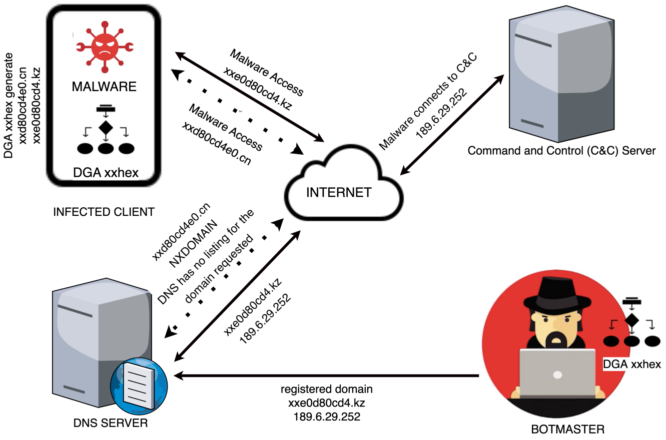

Fig. 2. Illustration of the use of a DGA. The botmaster and malware on an infected client generate the same list of domain names. The botmaster registers a domain from the list. The malware attempts to resolve each domain from the list with the DNS until it finds the registered domain and a connection between the infected client and the C&C is successfully established.

The ability of the malware to dynamically generate and use new domain names prevents ordinary blacklisting from permanently blocking access between the botmaster and infected machines. Indeed, if a firewall or intrusion detection service detects and subsequently blocks one of these domains, the botmaster will use the DGA to create and register a new domain that is not yet blocked, and the malware will then use the same DGA as the botmaster to make a new connection. In this paper we train and use neural networks that can distinguish such generated domains from real, benign domains. Such machine learning models are used to recognize DGA domains in DNS traffic and mitigate harm [8].

B. Neural Networks

1) Multilayer Perceptron (MLP): A MLP is a machine learning model representing a fully connected neural network architecture. This model has an input layer, an output layer, and one or more hidden layers. Moreover, the neurons of adjacent layers are connected to each other.

Equation 1 describes an MLP neuron where the output, y, is derived from an activation function, σ. The argument for σ is formed by taking the dot product of the neuron's input vector x with the weight vector w and then adding the bias scalar b. In subsequent sections of this paper, the symbols x and y may

[2] https://www.tensorflow.org/lite/performance/post_training_quantization

be used in different contexts. We will explicitly define their meanings each time they are referenced to prevent ambiguity.

$$y \leftarrow \sigma(x \cdot w + b) \tag{1}$$

2) Long Short-Term Memory networks (LSTM): Understanding and abstract reasoning about a given piece of information depends on previous experience. For some tasks, such as reading text, humans can extract knowledge based on the context of previous and recent parts of the text. Standard neural networks do not have memory structures analogous to the human brain. This shortcoming is addressed in recurrent neural networks (RNNs), which closely resemble these memory processes. Therefore, with an RNN, it is possible to train and learn based on a previous element of an input sequence.

With RNNs, it is also possible to model problems where the input data is time-dependent or sequential, i.e., where the next task depends on the previous one. Examples of this can be found in activities such as temperature/weather forecasting, historical air quality trends, vehicle traffic congestion patterns, etc. Furthermore, in natural language processing, the meaning of texts and language structures that humans produce depends on previous texts' content.

Long Short-Term Memory (LSTM) networks are a kind of RNNs that are designed to resolve problems with vanishing and exploding gradients that occur with long dependencies in the traditional RNN models [17]. LSTMs are well suited for DGA detection [18], [19], [20], [21], [22], [23], [24], [25], [26], hence we wish to study the effectiveness of LSTMs for the DGA detection problem when privacy is considered.

At their core, LSTM networks revolve around the concept of a cell state, a form of internal memory. LSTM cells have gates that regulate the flow of information into, within, and out of the cell. These gates can add or remove information from the cell state, acting as modification points in the memory system of the network.

We now describe such a cell in more detail and how it is used for inference, i.e. for computing outputs with a trained LSTM. We use the notation and closely follow the explanation presented in [27]. For our specific implementation, the input to the network is a sequence of characters $x_1, \ldots, x_t, \ldots, x_n$. We refer to Section II-B4 for a description of how each character is converted into a numeric representation. The inputs to the t-th cell consist of the output of the previous cell h_{t-1}, the t-th character x_t, and the state of the previous cell c_{t-1}. The cell outputs h_t and its state c_t. We now describe how these quantities are computed.

We first define how much of the previous state c_{t-1} we will "forget", denoted by the parameter f_t in equation 2. f_t is computed based on the dot product between the weights w_f and the concatenation of the output of the previous cell h_{t-1} and the cell's input x_t and adding the result to the bias b_f. Weights and biases are learned during training. This result is the input to the *sigmoid* function σ that returns values between 0 and 1. We will see that f_t equal to one means that we keep all of the previous state, while an f_t equal to zero means we forget everything about it when computing the state of the current cell.

$$f_t \leftarrow \sigma(w_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

The state of the t-th cell is computed according to equation 5. It is a weighted average of the previous cell state c_{t-1}, and a state that depends on the current input x_t, here denoted $c_t^{'}$. The weights are f_t and i_t, a coefficient that tells us how much of the "current" state ($c_t^{'}$) we want to keep. i_t and $c_t^{'}$ are computed according to equations 3 and 4. w_i, w_c are weights and b_i, b_c are biases computed during training. $tanh$ is the hyperbolic tangent.

$$i_t \leftarrow \sigma(w_i \cdot [h_{t-1}, x_t] + b_i]) \tag{3}$$

$$c_t^{'} \leftarrow tanh(w_c \cdot [h_{t-1}, x_t] + b_c]) \tag{4}$$

$$c_t \leftarrow f_t \cdot c_{t-1} \cdot i_t \cdot c_t^{'} \tag{5}$$

Finally, the output of the t-th cell h_t is computed according to equations 6 and 7. w_o and b_o are weights and biases, respectively.

$$o_t \leftarrow \sigma(w_o \cdot [h_{t-1}, x_t] + b_o]) \tag{6}$$

$$h_t \leftarrow o_t \cdot tanh(c_t) \tag{7}$$

h_t, c_t, and x_{t+1} are then fed into the next cell and the computations happen similarly for the $t+1$-th cell.

3) Convolutional Neural Network (1D-CNN): A Convolutional Neural Network (CNN) usually comprises convolution blocks followed by a fully connected network. A standard convolution block consists of a convolution layer, followed by an activation layer, and then by a pooling layer. The standard convolution layer (2D-CNN) takes a 3D input of height h, width w, and depth c and consists of f number of 3D learnable kernels each of size $k \times l \times c$. Each kernel moves along two directions of the input to generate a 3D output. In a 1-dimensional CNN layer (1D-CNN), the input is instead a matrix. The kernel moves along only one direction [28].

Let x be the input of a 1-dimensional convolution layer (1D-CNN). Let y be the output of the 1D-CNN layer, k is the total number of kernels, where the length of y is equal to $l - k + 1$. The kernel applies a sliding window operation on the input x.

The output of a 1D-CNN can be represented by equation 8, where $y[i]$ represents the output in the position i. The operation involving x and w is a dot product; b is the bias; w is the weight of the 1D-CNN trained and represents kernels; $w[j]$ is a kernel in the position j.

$$y[i] \leftarrow \sum_{j=0}^{k-1} (x[i+j] \cdot w[j]) + b, \tag{8}$$

4) Embedding layer: Embedding layers make representing a text in a vector of finite precision real numbers possible. In this work, each letter of a domain name is represented by a vector of finite precision real numbers. Instead of manually specifying the values for the embedding, they are trainable parameters.

We provide the first protocol and implementation for an embedding layer over MPC and implement it in the MP-SPDZ framework. The main idea behind our solution explained in detail in Section IV is to represent Alice's input to the protocol as a matrix x where each row of x is a one hot encoding of a character of the domain name to be classified. So, each row of x consists of a binary vector with Hamming weight equal to 1. A dot product is made between x and the private embedding matrix w, resulting in y, and the result of the embedding layer is the input to 1D-CNN, LSTM, and MLP models.

Our process to get the output of the embedding layer can be described by Equation 9:

$$y \leftarrow x \cdot w, \tag{9}$$

where the matrix x of dimension $l \times c$ represents one-hot-encoded inputs, the output of the embedding layer is the matrix y of dimension $l \times d$ representing an embedding of the input domain name to be classified, and matrix w of dimension $c \times d$ represents an embedding matrix, where l is the length of input domain, c is the length of the character set, and d is the size of the dimensional vector space.

Drichel et al. [15] proposed a solution for MPC-based private inference of DGA models. Their approach utilized several publicly available MPC frameworks for implementation. At the time, no protocol or implementation for private embedding existed, so the authors assumed that the embedding layer was publicly accessible and implemented in the clear by the party responsible for classifying the domain. However, this assumption could lead to information leakage about the model, especially if the embedding was trained on private data.

C. Privacy-Enhancing Technologies

1) Differential Privacy (DP): Informally, a differentially private (DP) algorithm produces a given output with approximately the same probability, regardless of whether a single entry is present or absent in a dataset used to compute the algorithm output. This means that the output is negligibly affected by the participation of a single user, thereby offering privacy through plausible deniability.

We recall the definition of differential privacy:

Definition 1: (ϵ, δ) - Differential Privacy [13]: A randomized algorithm $\mathcal{M}$ with domain $\mathbb{D}$ is (ϵ, δ) - differentially private if for all S $\subseteq$ Range($\mathcal{M}$) and for all $x, y \in \mathbb{D}$ such that $||x-y||_1 \leq 1$:

$$\Pr[\mathcal{M}(x) \in S] \leq \exp(\epsilon)\Pr[\mathcal{M}(y) \in S] + \delta \tag{10}$$

Definition 1 implies that queries on datasets x and y differing on a single entry ($||x-y||_1 \leq 1$) should produce different results with probability bounded by a quantity depending on the parameters ϵ and δ. The constant ϵ is the *privacy budget*. The smaller the value of ϵ, the more privacy the randomized algorithm $\mathcal{M}$ offers. The constant δ captures a small probability of violating the privacy guarantee. When δ equals zero, we say we have pure DP. When $\delta > 0$, we say we have approximate DP. δ is usually heuristically chosen to be smaller or equal to the reciprocal of the dataset size.

Deep learning models often leak information about their training dataset. Moreover, that leak is possible even when only black-box access is available, i.e. when an adversary only observes the output of the deep neural network. Differential privacy is used as a way to prevent such leaks. One can train deep learning models with DP guarantees using DP-SGD (differentially private stochastic gradient descent) [29].

The two essential steps in DP-SGD compared to traditional SGD are gradient clipping and noise addition. Gradient clipping is a technique that limits the magnitude of gradients to a predetermined threshold. Gradient clipping means the model's gradients computed for each data point are scaled down if their magnitude exceeds a certain threshold. This step prevents individual data points from disproportionately impacting the model's learning process, thus reducing the model's sensitivity to any single data point. Once the gradients have been clipped, noise is added to provide privacy. The noise is typically sampled from a Gaussian distribution, with the standard deviation or scale parameter proportional to the chosen privacy budget ϵ. Smaller values of ϵ provide higher privacy but require more significant amounts of noise to be added to the model, which reduces the model's accuracy.

To the best of our knowledge, we are the first to study the trade-off between privacy and the utility of DP-SGD in the context of DGA classification.

2) Secure Multi-Party Computation: Secure Multi-Party Computation (MPC) protocols allow mutually distrustful parties to engage in a computation so that, at the end of the protocol, all the honest parties have received the correct output of the computation. No collusion of dishonest parties can learn any information other than what can be inferred from the inputs and outputs of the dishonest parties in the computation. We use a variant of the traditional MPC scenario, where computing servers offer MPC as a service, and the inputs can come from outside parties. These parties do not engage in the MPC protocol. We refer the reader to one of the many available introductions to MPC in the literature [30], [31], [32], [33].

The main idea behind all available MPC protocols is to decompose the function to be privately computed into a circuit consisting of addition and multiplication gates. Then, we execute the underlying MPC protocol for evaluating each addition and multiplication gate sequentially till the result is computed and revealed to a designated receiver. Decomposing the functions to be computed into circuits consisting of addition and multiplication gates is a non-trivial task. Efficient representations can dramatically increase the performance of an MPC computation of a given function.

We implement our solutions using MP-SPDZ [16]. MP-SPDZ is a publicly available framework for implementing multiple MPC protocols. MP-SPDZ has a high-level interface in Python for presenting a circuit to be computed over an MPC protocol. MP-SPDZ also has several circuit representations of machine learning algorithms, including multilayer perceptrons and 2D convolutional neural networks. However, no LSTM implementation in MP-SPDZ is available in the literature. Embedding layers are also not available in MP-SPDZ. This work presents novel circuit representations for computing the inference of LSTM networks and embedding layers.

We work with MPC protocols based on secret sharing. Secret sharing is the computational process of splitting an input s into multiple secret shares x_i and giving these shares to shareholders. Only authorized families of the set of shareholders can recover the secret. Notably, no shareholder can obtain any information on the secret s. Secret sharing-based MPC protocols work by having the inputs to the computation secret shared among all the computing servers. Computations happen on shares rather than on the inputs themselves [32].

D. Quantization

The precision used to represent a machine learning (ML) model's parameters impacts the accuracy, runtime, and size of the model. The performance impact of such precision is magnified when ML models are run on top of secure multi-party protocols. So, it is desirable to quantize (reduce the precision) used in ML models to make them lightweight. We aggressively use quantization after training to optimize our models [34]. By reducing the precision of the weights of our models to 16 bits, we reduce runtimes by 15%, with a minimal impact on accuracy.

III. Proposed Framework, Privacy Requirements, and Threat Model

Overview of the Proposed Framework. We recall that *Alice* holds a DNS domain to be classified, and *Bob* holds a machine learning model that classifies DNS domains into malicious or benign. Our framework consists of set of m untrusted computing servers (MPC servers) $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$. While our proposed MPC protocols are general and work for any number of servers, we demonstrate our proposed protocols for $m = 2$ and $m = 3$. We assume pairwise authenticated and private communication channels between the servers. The communication between *Alice*, *Bob*, and any server $S_i \in \mathcal{S}$, is also authenticated and private. Our proposed secure classification works as follows:

- Initially, *Alice* and *Bob* convert their real-valued private inputs (domains in the case of *Alice* and the model parameters in the case of *Bob*) into fixed-point representations. They then secret-share their respective fixed-point inputs with the computing servers.
- The computing servers then engage in MPC-based communications and computations to execute the MPC protocols to classify *Alice's* domain names using *Bob's* model.
- The inference result is secret shared among the computing servers at the end of the MPC protocols. The computing servers send their shares to *Alice*. Finally, *Alice* aggregates the secret shares and retrieves the classification result.

Threat Model. We build our MPC protocols on existing MPC primitives [16]. Our proposed protocols can be adapted to any threat model by replacing the underlying primitives available in the literature for the given threat model. We will demonstrate the working of our protocols for the "honest-but-curious" threat model. In that model, the MPC servers follow the protocol instructions but try to obtain as much knowledge as possible about the secret data. We consider threat models (1) with two and three computing parties and (2) that can withstand one corruption, i.e., the privacy guarantees are maintained even when one of the MPC servers is corrupted.

Privacy Requirements. *Bob* should learn nothing about *Alice's* input. *Alice* should only learn the result of the classification protected by (ϵ, δ)-differential privacy and learn nothing else about *Bob's* model and training data. The MPC servers learn nothing about *Alice's* and *Bob's* private inputs.

IV. Methodology and Proposed Protocols

A. Training DGA Classifiers using Differential Privacy

In our proposed solution, *Alice* is provided with the classification output of her domains. In the binary case it is DGA/non-DGA. In the multiclass case, the output is non-DGA or one specific DGA family. This output should preserve the privacy of the individual entries of *Bob's* training data set. We employ the well-known DP-SGD (differentially private stochastic gradient descent) technique [29] to mitigate privacy concerns and provide DP guarantees for *Bob's* training data set according to Definition 1. We specifically train MLP, 1D-CNN, and LSTM models with DP guarantees.

Post-training quantization is an effective technique to improve the inference performance of a trained model. Post-training quantization transforms the model's weights and activations from floating-point precision (32-bit) to lower-precision representations. We propose to quantize the trained model parameters to float16,[3] which causes minimal loss in accuracy and allows for wider deployment of machine learning models with various hardware specifications (e.g. GPU, CPU with float32 or float16 instruction sets). The DP guarantees follow for the post-training quantized model due to the post-processing property of DP. We note that our framework can be adapted to other quantization schemes as well [35], [36], [37]. By combining DP and quantization techniques, we can preserve privacy for *Bob's* training data set and enable faster inference.

B. Secure Inference of DGA domains

During the inference stage, *Alice* has an instance (raw text input, i.e., the domain name string) that needs to be classified as a DGA or non-DGA domain. *Alice* begins by adjusting the length of the given input text to a publicly known value l by truncating the input text or padding the input text with zeros. The fixed length text is then one hot encoded based on ASCII characters resulting in a matrix x of dimension $l \times 128$.[4] *Alice* then secret shares matrix x and *Bob* secret shares the model parameters with the computing servers. The architecture of *Bob's* model is publicly known.

The computing servers execute MPC protocols that output the secret shares of the classification result to *Alice*. Our proposed framework is equipped to provide inference using models that use MLP, 1D-CNN, and LSTM architectures. All of these models have an embedding layer as the first layer. We next describe the proposed MPC protocols for these models that perform secure inference.

[3] https://www.tensorflow.org/lite/performance/post_training_quantization#float16_quantization

[4] ASCII is a set of 128 characters.

Basic Building Blocks. We build our proposed protocols upon a few building blocks that are already available in the MP-SPDZ framework [16]. We use MPC protocols π_{SIGMOID} for the sigmoid function, π_{SOFTMAX} for the softmax function, π_{MUL} for secure dot product, π_{TANH} for the hyperbolic tangent, π_{RELU} for relu function, and π_{DENSE} for dense layer. See [16] for a detailed description of these primitives.

Privacy-Preserving Embeddings. The embedding layer transforms the high-level input matrix x from *Alice* to provide a dense vector representation of the characters in the input text using *Bob's* learned embedding weights w. Many previous works for MPC-based privacy-preserving text classification, including DGA detection, require *Alice* to embed the input text, which in turn requires the trained embeddings to be made public and may leak information regarding the training data unless trained with DP guarantees [38], [15]. Moreover, these trained embeddings may be proprietary. To mitigate the above scenarios, we propose a novel MPC protocol for the embedding layer to compute the embeddings of private input text in an oblivious manner. The vectors resulting from the embedding layer of our classification models represent the lexical information of the characters in the given DGA domain (URL) in the ASCII character set. The idea behind $\pi_{\mathsf{EMBEDDING}}$ is simple: we extract the vector representation of each character in the input text (in our case, it is a domain or URL) from the trained embeddings with DP guarantees. One of the simplest ways to extract such embeddings (Protocol 1) is to represent the input text as a one hot encoded matrix x of dimension $l \times 128$ and multiply it with the weights of trained embeddings w of dimension 128×128. The product is a matrix y of dimension $l \times 128$ representing a set of vector representations of each character in the input text. We note that feature extraction done this way requires only multiplication operations for which state-of-the-art MPC primitives are available and results in an optimized performance of the MPC protocol for extracting embeddings of the input. Moreover, our protocol is general enough to work with character sets of arbitrary cardinality c.

Protocol 1: $\pi_{\mathsf{EMBEDDING}}$ for secure inference of embedding layer

```
Input : Secret shared matrices $x$ of dimension $(l \times c)$
        representing one-hot-encoded inputs and $w$ of
        dimension $(c \times d)$ representing embedding weights,
        where $l$ is the length of the input text in characters,
        $c$ is the cardinality of the character set, and $d$ is the
        dimensionality of the embedding space.
Output: A secret shared embedding matrix $y$ of dimension
        $(l \times d)$ of the input domain to be classified.
$y \leftarrow \pi_{\mathsf{MUL}}(x, w)$
return $y$
```

Privacy-Preserving MLP

In our work, we leverage an existing implementation of an MPC protocol for secure inference with an MLP, available within the MP-SPDZ framework [16]. MLPs will be used as a baseline method in our framework.

The input will be secret shared by *Alice*, while *Bob* secret-shares the model's weights. Bob's model in the architecture with MLP composes the weights of the embedding and dense layers.

Privacy-Preserving 1-D Convolution

We now present our solution based on 1D-CNN. Our architecture has an input layer with the protocol $\pi_{\mathsf{EMBEDDING}}$, a layer with the protocol $\pi_{\mathsf{1D-CNN}}$, and a layer with protocol π_{DENSE} representing the dense layer in MPC. The input will be secret shared by *Alice*, while *Bob* secret-shares the model's weights.

We leverage an existing proposal for a 1D-CNN [38] but provide our own implementation. The protocol $\pi_{\mathsf{1D-CNN}}$ for secure inference with 1D-CNN is built upon (a) the existing MPC protocols available in the literature for π_{RELU}, and π_{MUL} (b) our proposed MPC protocols for embedding $\pi_{\mathsf{EMBEDDING}}$ and 1-D convolution $\pi_{\mathsf{1D-CNN}}$. Protocol 2 for secure inference with a 1-D convolutional layer takes as input (1) the secret shared embeddings obtained as output from Protocol $\pi_{\mathsf{EMBEDDING}}$ and (2) the secret shared model parameters, i.e. weights of the kernels (k in total) each of size k for the 1-D convolution layer from *Bob*.

Protocol 2: $\pi_{\mathsf{1D-CNN}}$ for secure inference with 1-D Convolution.

```
Input : The secret shared matrices $x$ (obtained as output of
        the private embedding computed by Protocol
        $\pi_{\mathsf{EMBEDDING}}$), $b$, and $w$ represent input, bias, and
        weight. The constants $k$, $l$ represents the number of
        kernels, and rows of $x$
Output: A secret shared $y$ of dimension $l - k + 1$
for $i \leftarrow 0$ to $l - k + 1$ do
    $y[i] \leftarrow b$
    for $j \leftarrow 0$ to $k - 1$ do
        $y[i] \leftarrow y[i] + \pi_{\mathsf{MUL}}(x[i + j], w[j])$
    end
end
return $\pi_{\mathsf{RELU}}(y)$
```

Privacy-Preserving LSTM Protocol

We now present our solution based on LSTM. Our architecture has an input layer with the protocol $\pi_{\mathsf{EMBEDDING}}$, a layer with the protocol π_{LSTM}, and a layer with protocol π_{DENSE} representing the dense layer in MPC. The input will be secret shared by *Alice*, while *Bob* secret-shares the model's weights. Protocol 3 describes the LSTM layer. The input for the LSTM layer is the secret shared output of the embedding layer, while *Bob* secret shares the kernel weights (w_f, w_i, w_o, w_c) and the biases (b_f, b_i, b_o, b_c). We refer the reader to Section II-B2 for an explanation of each one of these terms. The operations involve MPC protocols π_{SIGMOID} for sigmoid, π_{MUL} for secure multiplications and π_{TANH} for the hyperbolic tangent.

V. Results

A. Dataset

The DGA dataset was obtained from DGArchive[5]. The dataset from DGArchive was the set containing all collected DGA examples up to 2019. We remove DGA families with

[5] https://dgarchive.caad.fkie.fraunhofer.de/welcome/

Protocol 3: π_{LSTM} for secure LSTM

```
Input : Secret shared vector $x$ (obtained as output of the
        private embedding computed by Protocol
        $\pi_{\mathsf{EMBEDDING}}$), and secret shared values of the
        weights ($w_f$, $w_i$, $w_o$, $w_c$), and biases ($b_f$, $b_i$, $b_o$,
        $b_c$). The input length is publicly known. Let $[a, b]$
        denote the concatenation of $a$ and $b$
. Output: A secret shared vector $y$ containing the output
        after the LSTM layer of the inference process.
$h_0 \leftarrow 0$
$c_0 \leftarrow 0$
for $t \leftarrow 1$ to $n$ do
    $f_t \leftarrow \pi_{\mathsf{SIGMOID}}(\pi_{\mathsf{MUL}}(w_f, [h_{t-1}, x[t]]) + b_f)$
    $i_t \leftarrow \pi_{\mathsf{SIGMOID}}(\pi_{\mathsf{MUL}}(w_i, [h_{t-1}, x[t]]) + b_i)$
    $c_t^{'} \leftarrow \pi_{\mathsf{TANH}}(\pi_{\mathsf{MUL}}(w_c, [h_{t-1}, x[t]]) + b_c))$
    $c_t \leftarrow \pi_{\mathsf{MUL}}(f_t, \pi_{\mathsf{MUL}}(c_{t-1}, \pi_{\mathsf{MUL}}(i_t, c_t^{'})))$
    $o_t \leftarrow \pi_{\mathsf{SIGMOID}}(\pi_{\mathsf{MUL}}(w_o, [h_{t-1}, x[t]]) + b_o)$
    $h_t \leftarrow \pi_{\mathsf{MUL}}(o_t, \pi_{\mathsf{TANH}}(c_t))$
    $y[t] \leftarrow h_t$
end
return $y$
```

low representation from the dataset (less than 30k samples). We end up with 1,000,000 examples from the DGArchive dataset for use as positive matches from the following families: bamital, banjori, bedep, beebone, blackhole, bobax, conficker, corebot, cryptolocker, darkshell, dircrypt, dnsbenchmark, dnschanger, downloader, dyre, ekforward, emotet, feodo, fobber, gameover, gameover_p2p.csv, gozi, gspy, hesperbot, locky, madmax, matsnu, modpack, murofet, murofetweekly, necurs, nymaim, oderoor, padcrypt, proslikefan, pushdo, pushdotid, pykspa, pykspa2, pykspa2s, qadars, qakbot, ramdo, ramnit, ranbyus, randomloader, redyms, rovnix, shifu, simda, sisron, suppobox, sutra, symmi, szribi, tempedreve, tinba, torpig, tsifiri, urlzone, vawtrak, virut, volatilecedar, xxhex.

For negative DGA matches, we have acquired approximately 1,000,000 domains from the last known version of the dataset "Alexa top 1 million domains"[6], which are used for training the model to recognize legitimate domains.

All data is shuffled and split into 80% for training and 20% for testing for binary and multiclass model architectures.

We convert alphanumeric characters representing domain names to lowercase for use in the model. Then, we convert each character to its corresponding ASCII code, which lies between the values of 0 to 127. Finally, the maximum length of a domain ASCII string is set to 64 characters. For domains whose size is less than 64, we prepend zero padding length to 64.

In this section, we evaluate our experimental results. First, we compare the 1D-CNN, LSTM, and MLP models using secure MPC when applied to binary and multiclass DGA detection with and without differential privacy. We also experimented with quantizing the models after the DP training (in which case, after training with DP, we applied quantization to reduce the weights to 16 bits and thus achieve performance gains in the inference phase with MPC).

B. Model Architectures

All trained models have an embedding Layer as the first layer, where the input is a vector of 64 numerical elements resulting from transforming each character into ASCII code. The result of embedding is a 128 by 128 dimension array.

The last layer in all binary models comprises one dense layer with one neuron, activation function sigmoid, loss function binary cross-entropy, and optimizer Adam.

The last layer in all multiclass models comprises one dense layer with 65 neurons representing all families, activation function softmax, loss function sparse categorical cross-entropy, and optimizer Adam with a learning rate of 0.001, batch size of 64, and 30 epochs.

We now provide further details about the other layers for each one of the architectures we used.

MLP: The binary and multiclass MLP models have a flatten layer to transform the data resulting from the embedding layer into one dimension data followed by a dense layer with 100 neurons, ReLU activation function and dropout with rate 0.1. The MLP binary model has 835,785 parameters, while the MLP multiclass model has 842,249 parameters.

1D-CNN: The binary and multiclass 1D-CNN models have: (1) a 1D-CNN with filters=32, kernels=2, ReLU activation function and dropout with rate 0.1; (2) followed by a flatten layer to transform the output of the previous layer into one-dimensional data; (3) a dense Layer with 100 neurons, ReLU activation function, dropout with rate 0.1. The 1D-CNN binary model has 226,409 parameters, while the 1D-CNN multiclass model has 232,671 parameters.

LSTM: The binary and multiclass LSTM models have: (1) a LSTM with 32 units with ReLU activation function and dropout with rate 0.1; (2) followed by a flatten layer to transform the output of the previous layer into one-dimensional data; (3) a dense layer with 100 neurons, ReLU activation function and dropout with rate 0.1. The LSTM binary model has 48,713 parameters, while the LSTM multiclass model has 55,177.

C. Experiments

Utility-Privacy Trade-Off: Table I provides a comprehensive analysis of the trade-off between utility (as measured by the accuracy of the model) and privacy (as represented by differential privacy within a specific privacy budget, denoted as ϵ) for all models. The privacy budget ϵ is a quantifiable measure of privacy. As a rule of thumb, an ϵ value less than one indicates high privacy, a value between one and two suggests moderate privacy, while an ϵ exceeding two is considered low privacy. ϵ equal to infinity represents a model without any differential privacy guarantee. As expected, the model's accuracy decreases as the privacy budget reduces, thus entering into a higher privacy regime. The decrease is more severe for the multiclass case. This observation captures the trade-off between utility and privacy in applying differential privacy. Table I also presents the accuracy levels of each model when quantization is applied. Note that the quantization step did not change the accuracy even though that quantized models are significantly faster than their non-quantized counterparts.

[6] https://en.wikipedia.org/wiki/Alexa_Internet

Model	Non-Quantized, $\epsilon =$				Quantized, $\epsilon =$			
	0.1	2	5	∞	0.1	2	5	∞
1D-CNN binary	90%	93%	93%	99%	90%	93%	93%	99%
1D-CNN multiclass	25%	47%	53%	88%	25%	47%	53%	88%
LSTM binary	88%	91%	92%	97%	88%	91%	92%	97%
LSTM multiclass	23%	46%	51%	88%	23%	46%	51%	88%
MLP binary	90%	93%	93%	96%	90%	93%	93%	96%
MLP multiclass	24%	46%	51%	87%	24%	46%	51%	87%

TABLE I
RESULTS ON THE DGA INFERENCE ACCURACY FOR DIFFERENT NOISE LEVELS WITH AND WITHOUT QUANTIZATION

Runtimes. We implemented the MPC-based inference both in the scenario of two computing servers (2PC) as well as three computing servers (3PC) connected over a Gigabit Ethernet local area network. In the inference experiments, we used three Azure instances with 32 Cores of Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz and 64GB of RAM. We used the following underlying MPC protocols available on MP-SPDZ [16] in our experiments: semi2k for 2PC and replicated secret sharing for 3PC. The runtimes are available in Table II and include communication and computation delays. The MLP model had the best runtimes, but 1D-CNN had the better accuracy, especially for higher values of ϵ (see Table I).

We note that runtimes are unaffected if the model is protected by differential privacy. Note also that runtimes strongly depend on the corruption threshold: the three party protocols (i.e., honest majority) are faster than the two-party protocols, in which there is no honest majority.

Model	Setting	Inference Time (sec)	Rounds	Data Sent (MB)
MLP Binary	3PC	0.0778787	2773	17.217
1D-CNN Binary	3PC	0.319441	8551	21.6062
LSTM Binary	3PC	10.4153	195131	1485.53
MLP Multiclass	3PC	0.133239	3615	24.8676
1D-CNN Multiclass	3PC	0.359278	9382	29.0157
LSTM Multiclass	3PC	10.5449	197123	1489.92
MLP Binary	2PC	14.2051	42923	3951.16
1D-CNN Binary	2PC	14.2954	54151	3983.37
LSTM Binary	2PC	103.472	441023	26752.6
MLP Multiclass	2PC	14.577	44599	4054.73
1D-CNN Multiclass	2PC	14.6503	55831	4089.34
LSTM Multiclass	2PC	104.077	443895	26820.1

TABLE II
INFERENCE USING MP-SPDZ PROTOCOL

We also performed the same experiments regarding MPC-based inference but using the models that we quantized after the DP training. The runtimes are presented in Table III. Note that in the 3PC setting (using replicated secret sharing) the quantization step reduced inference runtime by 15%, as opposed to a 2% reduction in runtime for the 2PC setting (using semi2k).

D. Security and Privacy

The security of our solution follows from the security of the underlying MPC protocols (3-party replicated and 2-party semi2k in MP-SPDZ [16]) and from the fact that we only perform operations over secret shares. Therefore, the computing servers responsible for the MPC operations never learn anything about *Alice's* or *Bob's* inputs. That provides input privacy to our solution. Output privacy - the guarantee that *Alice* does not learn information about individual entries used in the training of *Bob's* model - is provided by the DP guarantees of DP-SGD [29] as utilized by *Bob*.

Model	Setting	Inference Time (sec)	Rounds	Data Sent (MB)
MLP Binary	3PC	0.0593449	2469	12.3028
1D-CNN Binary	3PC	0.223521	7371	12.3982
LSTM Binary	3PC	5.9959	137044	545.121
MLP Multiclass	3PC	0.076885	3038	15.0529
1D-CNN Multiclass	3PC	0.260958	7940	15.1483
LSTM Multiclass	3PC	6.15767	138573	546.706
MLP Binary	2PC	13.813	42723	3845.97
1D-CNN Binary	2PC	14.1394	51795	3937.96
LSTM Binary	2PC	60.3208	310659	15352.2
MLP Multiclass	2PC	13.9138	44599	3983.37
1D-CNN Multiclass	2PC	14.4177	54151	4054.73
LSTM Multiclass	2PC	60.247	312951	15388.2

TABLE III
INFERENCE USING MP-SPDZ PROTOCOL WITH QUANTIZATION AFTER DP TRAINING

VI. RELATED WORKS

We now present the related literature and compare our solution to previous ones.

A. DGA detection with deep learning

DGA detection methods did not rely on machine learning in the early stages. For instance, Sharifnya et al. [39] developed a technique that identifies hosts with a high volume of failed DNS queries, subsequently adding these hosts to a "suspicious failure matrix." This section lists relevant works regarding DGA detection using deep or machine learning without any privacy guarantees.

Li et al. [40] propose several real-time detection models and frameworks which utilize meta-data generated from domains and combine the advantages of a deep neural network model and a lexical features-based model using the ensemble technique. Another work in this line is [41], which uses Helix's architecture that represents DGA as embeddings. The utilization of Multilayer Perceptron (MLP) for DGA output detection was also investigated [42], [43].

Huang et al. [44] propose a Helios architecture that uses CNN to detect DGA. Zhou et al. [45] also uses 1D-CNN to detect DGA and enables binary and multiclass analysis of DGA. Berman [46] uses 1D-CNN with a convolutional layer of one-dimensional data to detect DGA. Chen et al. [47] apply 1D-CNN and BiGRU to detect DGA.

Shahzad et al. [48] uses a recurrent neural network to detect DGA outputs on a per-domain basis using the domain name only, with no additional information, which the authors compare to the performance of a DGA classifier based on the following RNN architectures: Unidirectional LSTM network, bidirectional LSTM (Bi-LSTM), and Gated Recurrent Unit (GRU).

Zhang et al. [49] design and implement several DGA classifiers based on machine learning (SVM and RF) and deep learning (CNN, LSTM, and Bi-LSTM) methods. Yang et al. [50] exploits the character-level characteristics of the DGA domain names and proposes a heterogeneous deep neural network framework that includes 1D-CNN and LSTM. LSTM has been used to detect binary DGA and multiclass DGA by the alphanumeric domain name [18], [19]. Tran et al. [21] present a new LSTM algorithm to address the multiclass imbalance problem in DGA-related botnet detection. Strategies with unbalanced datasets are vital for multiclass DGA detection. Balakrishna et al. [51] use an LSTM which is adapted to predict better if the dataset is unbalanced. Josan et al. [22] use a bidirectional LSTM network for binary DGA and multiclass DGA detection. Other applications that use LSTM to detect DGA are discussed in works [23], [24], [26], [52].

Liu et al. [53] combines a convolutional neural network and a bidirectional long short-term memory network to detect DGA. Yun et al. [54] show a method based on natural language processing and Wasserstein Generative Adversarial Networks and a new way to prevent attackers' evasion of neural network's DGA detection.

Malware detection is relevant for IoT applications (Internet of Things), and DGA detection for this IoT scenario has been investigated [55].

Li et al. [56] make the inference using the Hidden Markov Model (HMM). Koh et al. [57] uses a pre-trained context-sensitive word embedding to classify DGA. Cucchiarelli et al. [58] use the Kullback-Leibner divergence and the Jaccard Index to estimate similarities to detect DGA. Finally, Yilmaz et al. [59] use a method to detect DGA using LSTM and add a GAN (generative adversarial network) to infer previously unknown malicious domains.

Yu et al. [9] comment that simpler architectures are faster in training and inference and are less prone to overfitting. This conclusion is fundamental and was the focus of this work, as we seek lighter architecture toward achieving greater performance in protocols with MPC. Another work from Yu et al. [8] proposes heuristics for automatically labeling domain names monitored in real traffic. This labeled data is essential for improving the accuracy of deep learning models in detecting DGA. Finally, Yu et al. [10] propose a new way to label a large volume of data collected from real traffic as DGA-related and non-DGA-related, allowing models to be trained with large amounts of real traffic.

Sivaguru et al. [60] strengthen DGA detectors against adversarial attacks and evaluate deep learning models and random forests (RFs) to detect DGA using information beyond the domain name.

B. Secure Multi-Party Computation for DGA Detection

The work of Drichel et al. [15] is the one that is mostly related to our proposal. That work proposes using MPC protocols to classify domains into DGA and non-DGA. They implement their proposals using several different MPC frameworks: PySyft, TF-Encrypted, MP2ML, and SecureQ8 applied in the classifiers Inline [10], NYU [9], ResNet [6], and FANCI. However, despite its pioneering aspect, the work of Drichel et al. [15] still has several deficiencies:

- It uses CNN2D for performing private inference of DNS domains. This adds unnecessary complexity to the classifiers since 1D-CNN is best suited to text classification problems.
- It does not use privacy-preserving embedded layers. In practice, this assumes that *Bob* has to leak the embedding layer to *Alice* for her to perform the embedding operation in the clear. Thus, this solution is not end-to-end private.
- It does not consider LSTM models over MPC. Our work develops the first LSTM implementation available in any existing MPC framework.
- It considers only binary classification. We show how to carry out binary and multiclass DGA predictions.

C. Secure Multi-Party Computation for Natural Language Processing

Hao et al. [61] present private inference on transformer BERT based models in a client-server setting. Clients have private inputs, and servers hold proprietary models. One contribution is a customized homomorphic encryption-based method for matrix multiplication.

Adams et al. [38] present the first application of MPC protocols for CNN-based text classification. Their method adapts a CNN2D from the Crypten framework into a 1D-CNN by utilizing two 2D convolutional layers to emulate the behavior of a 1D-CNN. In contrast, our work directly implements a one-dimensional, private embedding layer, resulting in a more efficient solution. Furthermore, [38] focuses on word-level classifications, while our research emphasizes secure character-level text classification. Lastly, Adams et al. [38] does not address private embeddings. Their approach assumes that *Alice* first converts her text into an embedded vector using a publicly available BERT model before secret-sharing it. This method is infeasible for a model where the embedding is private and part of *Bob's* confidential information. Our work overcomes this limitation by providing protocols and implementation for secure embeddings.

The model SecureNLP [62] has two security protocols for LSTM and RNN in the honest-but-curious model. The differences between SecureNLP and our solution are that we use LSTM for inference with characters, and SecureNLP conducts inference on words. Additionally, SecureNLP does not work with private embedding layers.

Knott et al. [63] offers a comprehensive overview of the Crypten framework, demonstrating its application in text classification, speech recognition, and image classification. In their work, text classification is conducted through a sentiment analysis experiment using a linear layer operating on word embeddings. Our approach differs from theirs in focusing on character-level rather than word-level classification. Moreover, their work does not involve private embeddings or provide an LSTM protocol and implementation.

VII. Conclusions and Future Work

This work presents the first framework for performing outsourced privacy-preserving DGA detection with input and output privacy. Input privacy means that no information about the domain being classified leaks to the computing servers or the model owner *Bob*, and no information about the model leaks to the domain owner (*Alice*) or the computing servers. Output privacy means that the output of the computation does not allow the extraction of individual entries in the training dataset of the ML model used in the framework.

We additionally contribute MPC protocols for LSTM and embedding layer in the MP-SPDZ framework, furthering the practical application of private and secure machine learning.

We compared the performance of CNN, LSTM, and MLP trained with DP and secure inference with various MPC protocols. The 1D-CNN presented better cost-benefit accuracy and performance on MPC for two parties, and in the three computing servers case there were trade-offs between CNN and MLP.

We demonstrated that quantizing the weights to 16 bits after training with DP improved our runtimes by 15% without a significant drop in accuracy when we are in the setting of three parties with honest majority.

In future work, we propose to apply the techniques of this paper to general tasks of natural language classification.

Acknowledgements. The authors would like to thank Marcel Keller for making the MP-SPDZ framework available and for his assistance in the use of the framework. The authors thank Microsoft for the generous donation of cloud computing credits through the UW Azure Cloud Computing Credits for Research program.

References

[1] "Enisa threat landscape - the year in review — enisa," https://www.enisa.europa.eu/publications/year-in-review, 2020, (Accessed on 09/09/2021). [Online]. Available: https://www.enisa.europa.eu/publications/year-in-review/view/++widget++form.widgets.fullReport/@@download/ETL2020+-+A+year+in+review+A4.pdf

[2] ENISA, "Enisa etl2020 - malware," https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/etl-review-folder/etl-2020-malware, 2020, (Accessed on 09/09/2021).

[3] McAfee, "Mcafee labs threats report, april 2021," https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-apr-2021.pdf, april 2021, (Accessed on 09/09/2021).

[4] "Dbir - data breach investigations report," https://enterprise.verizon.com/resources/reports/2021-data-breach-investigations-report.pdf, 2021, (Accessed on 09/09/2021).

[5] C. Patsakis and F. Casino, "Exploiting statistical and structural features for the detection of domain generation algorithms," *Journal of Information Security and Applications*, vol. 58, p. 102725, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212620308632

[6] A. Drichel, U. Meyer, S. Schüppen, and D. Teubert, "Analyzing the real-world applicability of dga classifiers," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ser. ARES '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3407023.3407030

[7] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A Comprehensive Measurement Study of Domain Generating Malware," in *25th USENIX Security Symposium*, 2016, pp. 263–278. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_plohmann.pdf

[8] B. Yu, J. Pan, D. Gray, J. Hu, C. Choudhary, A. C. A. Nascimento, and M. De Cock, "Weakly supervised deep learning for the detection of domain generation algorithms," *IEEE Access*, vol. 7, pp. 51 542–51 556, 2019.

[9] B. Yu, J. Pan, J. Hu, A. Nascimento, and M. De Cock, "Character level based detection of dga domain names," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, pp. 1–8.

[10] B. Yu, D. L. Gray, J. Pan, M. D. Cock, and A. C. A. Nascimento, "Inline dga detection with deep networks," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 683–692.

[11] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *USENIX 2019*, 2019, pp. 267–284.

[12] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 587–601. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3133956.3134077

[13] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3-4, p. 211–407, 2013.

[14] R. Cramer, I. Damgard, and J. Nielsen, *Secure Multiparty Computation and Secret Sharing*. New York: Cambridge University Press Print, 2015.

[15] A. Drichel, M. A. Gurabi, T. Amelung, and U. Meyer, "Towards privacy-preserving classification-as-a-service for dga detection," in *2021 18th International Conference on Privacy, Security and Trust (PST)*, 2021, pp. 1–10.

[16] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," Cryptology ePrint Archive, Report 2020/521, 2020, https://eprint.iacr.org/2020/521.

[17] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks," 2015.

[18] P. Vij, S. Nikam, and A. Bhatia, "Detection of algorithmically generated domain names using lstm," in *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)*, 2020, pp. 1–6.

[19] S. Akarsh, S. Sriram, P. Poornachandran, V. K. Menon, and K. P. Soman, "Deep learning framework for domain generation algorithms prediction using long short-term memory," in *2019 5th International Conference on Advanced Computing Communication Systems (ICACCS)*, 2019, pp. 666–671.

[20] H. Mac, D. Tran, V. Tong, L. G. Nguyen, and H. A. Tran, "Dga botnet detection using supervised learning methods," in *Proceedings of the Eighth International Symposium on Information and Communication Technology*, ser. SoICT 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 211–218. [Online]. Available: https://doi.org/10.1145/3155133.3155166

[21] D. Tran, H. Mac, V. Tong, H. A. Tran, and L. G. Nguyen, "A lstm based framework for handling multiclass imbalance in dga botnet detection," *Neurocomputing*, vol. 275, pp. 2401–2413, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217317320

[22] G. Josan and J. Kaur, "Lstm network based malicious domain name detection," 08 2019.

[23] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," 2016.

[24] Y. Qiao, B. Zhang, W. Zhang, A. K. Sangaiah, and H. Wu, "Dga domain name classification method based on long short-term memory with attention mechanism," *Applied Sciences*, vol. 9, no. 20, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/20/4205

[25] P. Lison and V. Mavroeidis, "Automatic detection of malware-generated domains with recurrent neural models," 2017.

[26] R. R. Curtin, A. B. Gardner, S. Grzonkowski, A. Kleymenov, and A. Mosquera, "Detecting dga domains with recurrent neural networks and side information," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3339252.3339258

[27] C. Olah, "Understanding lstm networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed: 2023-07-04.

[28] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, 2015.

[29] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[30] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Found. Trends Priv. Secur.*, vol. 2, no. 2–3, p. 70–246, Dec. 2018. [Online]. Available: https://doi.org/10.1561/3300000019

[31] Y. Lindell, "Secure multiparty computation," *Commun. ACM*, vol. 64, no. 1, p. 86–96, Dec. 2020. [Online]. Available: https://doi.org/10.1145/3387108

[32] R. Cramer, I. B. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[33] R. Cramer, I. Damgård, D. Catalano, G. Crescenzo, I. Darmgård, D. Pointcheval, and T. Takagi, *Multiparty Computation, an Introduction*, 03 2006, pp. 41–87.

[34] M. Keller and K. Sun, "Secure quantized training for deep learning," Cryptology ePrint Archive, Paper 2022/933, 2022, https://eprint.iacr.org/2022/933. [Online]. Available: https://eprint.iacr.org/2022/933

[35] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[36] A. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, no. 4, p. 355–375, Aug 2020. [Online]. Available: http://dx.doi.org/10.2478/popets-2020-0077

[37] M. Keller and K. Sun, "Secure quantized training for deep learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 10 912–10 938.

[38] S. Adams, D. Melanson, and M. De Cock, "Private text classification with convolutional neural networks," in *Proceedings of the Third Workshop on Privacy in Natural Language Processing*, 2021, pp. 53–58.

[39] R. Sharifnya and M. Abadi, "A novel reputation system to detect dga-based botnets," in *ICCKE 2013*, 2013, pp. 417–423.

[40] S. Li, T. Huang, Z. Qin, F. Zhang, and Y. Chang, "Domain generation algorithms detection through deep neural network and ensemble," in *Companion Proceedings of The 2019 World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 189–196. [Online]. Available: https://doi.org/10.1145/3308558.3316498

[41] L. Sidi, Y. Mirsky, A. Nadler, Y. Elovici, and A. Shabtai, "Helix: Dga domain embeddings for tracking and exploring botnets," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2741–2748. [Online]. Available: https://doi.org/10.1145/3340531.3416022

[42] M. I. Ashiq, P. Bhowmick, M. S. Hossain, and H. S. Narman, "Domain flux-based dga botnet detection using feedforward neural network," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019, pp. 1–6.

[43] J. Mao, J. Zhang, Z. Tang, and Z. Gu, "Dns anti-attack machine learning model for dga domain name detection," *Physical Communication*, vol. 40, p. 101069, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874490719309036

[44] J. Huang, P. Wang, T. Zang, Q. Qiang, Y. Wang, and M. Yu, "Detecting domain generation algorithms with convolutional neural language models," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018, pp. 1360–1367.

[45] S. Zhou, L. Lin, J. Yuan, F. Wang, Z. Ling, and J. Cui, "Cnn-based dga detection with high coverage," in *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2019, pp. 62–67.

[46] D. S. Berman, "Dga capsnet: 1d application of capsule networks to dga detection," *Information*, vol. 10, no. 5, 2019. [Online]. Available: https://www.mdpi.com/2078-2489/10/5/157

[47] C. Chen, L. Pan, and X. Xie, "Dga domain name detection based on bigru-mcnn," in *Proceedings of the 2019 4th International Conference on Intelligent Information Processing*, ser. ICIIP 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 315–319. [Online]. Available: https://doi.org/10.1145/3378065.3378126

[48] H. Shahzad, A. R. Sattar, and J. Skandaraniyam, "Dga domain detection using deep learning," in *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*, 2021, pp. 139–143.

[49] Y. Zhang, "Automatic algorithmically generated domain detection with deep learning methods," in *2020 IEEE 3rd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, 2020, pp. 463–469.

[50] L. Yang, G. Liu, Y. Dai, J. Wang, and J. Zhai, "Detecting stealthy domain generation algorithms using heterogeneous deep neural network framework," *IEEE Access*, vol. 8, pp. 82 876–82 889, 2020.

[51] S. K, P. Balakrishna, V. Ravi, and S. KP, "Deep learning based frameworks for handling imbalance in dga, email, and url data analysis," 2020.

[52] S. Kumar and A. Bhatia, "Detecting domain generation algorithms to prevent ddos attacks using deep learning," in *2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2019, pp. 1–4.

[53] Z. Liu, Y. Zhang, Y. Chen, X. Fan, and C. Dong, "Detection of algorithmically generated domain names using the recurrent convolutional neural network with spatial pyramid pooling," *Entropy*, vol. 22, no. 9, 2020. [Online]. Available: https://www.mdpi.com/1099-4300/22/9/1058

[54] X. Yun, J. Huang, Y. Wang, T. Zang, Y. Zhou, and Y. Zhang, "Khaos: An adversarial neural network dga with high anti-detection ability," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2225–2240, 2020.

[55] R. Vinayakumar, M. Alazab, S. Srinivasan, Q. Pham, S. K. Padannayil, and K. Simran, "A visualized botnet detection system based deep learning for the internet of things networks of smart cities," *IEEE Transactions on Industry Applications*, vol. 56, no. 4, pp. 4436–4456, 2020.

[56] Y. Li, K. Xiong, T. Chin, and C. Hu, "A machine learning framework for domain generation algorithm-based malware detection," *IEEE Access*, vol. 7, pp. 32 765–32 782, 2019.

[57] J. J. Koh and B. Rhodes, "Inline detection of domain generation algorithms with context-sensitive word embeddings," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 2966–2971.

[58] A. Cucchiarelli, C. Morbidoni, L. Spalazzi, and M. Baldi, "Algorithmically generated malicious domain names detection based on n-grams features," *Expert Systems with Applications*, vol. 170, p. 114551, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420311957

[59] I. Yilmaz, A. Siraj, and D. Ulybyshev, "Improving dga-based malicious domain classifiers for malware defense with adversarial machine learning," in *2020 IEEE 4th Conference on Information Communication Technology (CICT)*, 2020, pp. 1–6.

[60] R. Sivaguru, J. Peck, F. Olumofin, A. Nascimento, and M. De Cock, "Inline detection of dga domains using side information," *IEEE Access*, vol. 8, pp. 141 910–141 922, 2020.

[61] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private inference on transformers," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=deyqjpcTfsG

[62] Q. Feng, D. He, Z. Liu, H. Wang, and K.-K. R. Choo, "Securenlp: A system for multi-party privacy-preserving natural language processing," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3709–3721, 2020.

[63] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.