Secure Featurization and Applications to Secure Phishing Detection

Akash Shah*
UCLA
akashsha08@ucla.edu

Nishanth Chandran
Microsoft Research
India
nichandr@microsoft.com

Mesfin Dema
Microsoft Corporation
USA
medema@microsoft.com

Divya Gupta
Microsoft Research
India
divya.gupta@microsoft.com

Arun Gururajan
Microsoft Corporation
USA
argurura@microsoft.com

Huan Yu
Microsoft Corporation
USA
huanyu@microsoft.com

ABSTRACT

Secure inference allows a server holding a machine learning (ML) inference algorithm with private weights, and a client with a private input, to obtain the output of the inference algorithm, without revealing their respective private inputs to one another. While this problem has received plenty of attention, existing systems are not applicable to a large class of ML algorithms (such as in the domain of Natural Language Processing) that perform featurization as their first step. In this work, we address this gap and make the following contributions:

- We initiate the formal study of secure featurization and its use in conjunction with secure inference protocols.
- We build secure featurization protocols in the one/two/three-server settings that provide a tradeoff between security and efficiency.
- Finally, we apply our algorithms in the context of secure phishing detection and evaluate our end-to-end protocol on models that are commonly used for phishing detection.

KEYWORDS

Secure multi-party computation; Phishing Detection; Private Set Intersection

1 INTRODUCTION

Secure Inference. The availability of vast volumes of data, as well as advances in machine learning (ML) algorithms, have dramatically improved the performance of ML models in a variety of domains. With the ever growing demand to protect private information, perhaps the most important problem that has emerged in this space, is that of *secure inference*. In this, a server $\mathcal{S}$ holds a pre-trained ML model M (with weights w) and a client C holds an input x. The goal is for one (or both) of the parties to learn the output of the inference algorithm on input x - i.e., $M(w, x)$ - without revealing any other information to either participant. Specifically, C must learn nothing more about w (other than what can be inferred from the output it receives) and $\mathcal{S}$ must learn nothing about x. Secure inference is a special case of the cryptographically well-known notion of secure 2-party computation [16, 52] and a rich body of work [3, 8, 15, 19, 25, 29, 30, 33, 43–45, 51] has studied this problem in detail.

*Work done at Microsoft Research.

Featurization. Unfortunately, existing secure inference protocols are applicable only for use-cases such as image-recognition [19, 29, 33, 44], or for time-series forecasting [43]. In these applications, M can directly operate on the raw input data of C, call this x', as x' is a vector of numerical values. However, in a large class of ML algorithms, such as in the domain of Natural Language Processing (NLP) [11], the raw input data x' must be processed into a vector of numerical values x before it can be input to M [54]. This process is called *featurization*. A featurization algorithm, F, typically utilizes a list of tokens, known as the *dictionary* D, to convert x' to x. D is a private input of $\mathcal{S}$ (model owner) since it is obtained at model training time. Hence, to protect the privacy of inputs during secure inference, the featurization algorithm must also be executed in a privacy-preserving manner, a problem that has remained unexplored. This work focuses on the problem of *secure featurization* and its use in secure inference protocols. The goal in secure featurization is for $\mathcal{S}$, with dictionary D and C, with raw input x' to securely compute $x = \mathsf{F}(D, x')$. Further, in order to protect the sensitive inputs of both the participants, they must not learn this output (x) in the clear and only learn secret shares of it. These can then be fed into secure inference protocols in order for the relevant participants to learn $M(w, x)$.

Bag of Words Approach [54]. We focus on a ubiquitous featurization algorithm in the domain of NLP called *Bag-of-Words* approach. This technique makes use of *tokenization*, the process of parsing textual data record (raw input) into a list of tokens. This can be done either through simple n-gram character tokens or by splitting the text by basic delimiters (".", ",", ":", "/", etc.) to generate tokens.

To build the dictionary D, $\mathcal{S}$ first processes each textual record in the training dataset into tokens. $\mathcal{S}$ then performs further different kinds of processing on these obtained tokens, e.g., removes rarely occurring tokens, groups all numerical tokens as a single token called "numeric" token, groups all tokens that comprise of whitespace characters only (space character, tab character, new-line character) into a single token called "whitespace token", and so on. This list of processed tokens is called a *dictionary* D. Let n_0 denote the size of this dictionary.

Featurization of client's raw input x' into a vector of numerical values x (of same size n_0) is done as follows: First, tokenization is performed on x' to obtain a collection of tokens. These tokens are processed further to obtain *token list* T along with their corresponding associated attributes. Example of an associated attribute is the *frequency* with which the token occurs in the input text. Now, the featurization algorithm, F, operating on x', the associated attributes, and D is as follows: for an index $i \in [n_0]$, if $D[i] \in T$, it sets $x[i]$ to be the associated attribute, else it records 0.

Phishing Detection. With an aim to obtain sensitive information from end-users, phishing attacks are typically carried out by presenting a web-page that looks nearly identical to an authentic web-page that the user intended to access [13]. Machine learning techniques are a popular method to build phishing detection algorithms in order to offer *phishing detection as a service* to end-users [10, 18, 26, 28, 53]. In this, the server $\mathcal{S}$ trains an appropriate ML model, M, to obtain model weights w and a dictionary D. The input of the end-user C is the information of the accessed web-page, x' = (URL, Title, Body). The goal of a secure phishing detection system is to then allow C to learn output $M(w, \mathsf{F}(D, x'))$, without revealing any other information to either participant. Prior works [10] have attempted to solve this problem by weakening the security requirements, and in particular allow C to learn D in the clear. This weakening of security can have disastrous consequences - malicious C entities can use D to build phishing websites that can go undetected by the phishing detection system.

1.1 Our Contributions

In this work, we initiate the formal study of secure featurization. Motivated by efficiency in real-time machine learning systems, apart from the standard one-server threat model, we also consider and formalize other threat models as well. In these threat models, client privacy is provided in multi-server setting with the assumption that the servers do not collude among themselves. Protocols that satisfy our definitions, can be used in conjunction with existing efficient secure inference protocols in order to obtain an end-to-end secure inference protocol for use-cases that involve featurization.

We make the following main contributions.

- We build secure featurization protocols in the one/two/three-server settings that provide various security vs. efficiency trade-offs. The one-server protocol provides the strongest security for C, whereas the two/three-server protocols provide a weaker security for C as the security relies on non-collusion assumption of the servers. However, on the flip side, two/three-server protocols perform more than an order of magnitude better than the one-server protocol both in end-to-end latency as well as communication. For instance, for a dictionary of size $\approx 21,000$ and token list of size ≈ 70, our one-server, two-server and three-server secure featurization protocols take 3.66s, 0.26s and 0.05s respectively. The communication of C in these settings is 146590 KB, 180KB and 2.8 KB respectively. (See Section 5 for communication of servers and other details.)
- We evaluate our protocols on models that are commonly used for phishing detection in realistic scenarios such as web browser protection. More specifically, we build an end-to-end secure phishing detection system using our secure featurization protocols and existing secure inference protocols. The one-server, two-server and three-server protocols took 10.48s, 0.45s, and 0.08s respectively and require the client to communicate 1301 MB, 0.18 MB, and 2.8 KB, respectively. The end-to-end latency and performance of two-server and three-server protocols meet the performance requirement for production level environment in practice. Additionally, the three-server protocol has the property of being extremely lightweight from the client's perspective, an important criteria in production systems.

1.2 Techniques

We first give a very high level description of our protocols for secure featurization and then briefly discuss how we use it to build a secure phishing detection system.

One server setting. Our first observation is that the process of featurization is very similar to that of computing the private set intersection [17, 23, 39, 42] between the client C's token list T and the server $\mathcal{S}$'s dictionary D. Now, since the featurized input must not be revealed in the clear to any of the parties (and must be used as input to a secure inference algorithm), one must obtain secret shares of the intersection and not the intersection itself. Such a problem has been studied under the name of circuit private set intersection (circuit PSI) [40, 41]. Hence, one can potentially make use of a 2-party circuit PSI protocol followed by a 2-party secure inference protocol [44] to construct the secure phishing detection protocol. However, this is not completely true. In circuit PSI protocols [9, 40, 41], while the 2 parties do learn secret shares of the set intersection (i.e., for every element in the larger set, they learn shares of 1 if the element is also present in the smaller set and shares of 0 otherwise), these shares could be permuted in a random order known only to one of the parties. Hence, to allow for the use of such circuit PSI protocols as a preface to secure inference protocols, we do 2 things: first, we define a functionality that takes this random order as a parameter and show how existing protocols realize this functionality. Next, we modify the 2-party secure inference protocol [44] to compute the inverse permutation as the first step before proceeding to execute the secure inference protocol.

While this protocol provides the client with a high level of security, we observe that the performance of circuit PSI protocols do not satisfy the latency requirements to be used in real-world phishing detection systems. Hence, we consider both two and three server settings, where client privacy is only provided against non-colluding servers. Such a non-collusion assumption has been widely used in a variety of secure computation protocols where performance is of paramount importance [2, 30, 32, 51]. More recently, this assumption has seen adoption to provide real world privacy guarantees to users of web browsers [1, 12, 14], a setting relevant to our problem of secure phishing detection.

Two server setting. In the two-server setting, we use techniques from the recent line of work on Function Secret-Sharing (FSS) [4], to build a secure featurization protocol. An FSS scheme [4, 5] for a function f, splits f into f_0 and f_1 s.t. for all inputs x, $f_0(x)+f_1(x) = f(x)$ and each f_b computationally hides f. For each token t_i in token list T, C considers the point function f_{t_i,a_i}: $f_{t_i,a_i}(t_i) = a_i$ and $f_{t_i,a_i}(x) = 0$ for any $x \neq t_i$. (Here, a_i is the attribute associated with token t_i.) Now, using the FSS scheme, C obtains shares of function f_{t_i,a_i}, call this $g_{0,i}$ and $g_{1,i}$. It sends $g_{0,i}$ and $g_{1,i}$ to servers $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively. For each keyword w_j in dictionary D, $\mathcal{S}_b$ computes $\sum_i g_{b,i}(w_j)$, where $b \in \{0,1\}$. Correctness of FSS scheme ensures that if w_j is in T, then the servers obtain shares of the associated attribute, else they obtain shares of 0. The security property of FSS ensures that servers don't learn any information about C's inputs. However, this naïve technique results in a computation overhead of $O(n_0 n_1)$ at server's end, where n_0 is the size of the dictionary and n_1 is the size of tokens list. To reduce this, we deploy cuckoo hashing techniques[38], to bring the server's computation overhead down to $O(n_0)$. The protocol obtained involves only light-weight AES computations and hence, it is an order of magnitude faster than the one-server protocol. Unlike our one-server protocol, this two-server protocol outputs shares of associated attribute in order of elements in dictionary D, thus, no inverse permutation is needed and hence a 2-party secure inference protocol can be directly computed on outputs of the two-server secure featurization protocol. An additional benefit of this protocol is that the C performs no computation in the inference computation.

Three server setting. In certain real-time applications, end-to-end performance of secure inference is critical and a response time of less than 100 ms is desirable. For such cases, we propose a secure featurization protocol in the three-server setting. Here, there are 3 servers: $\mathcal{S}_0$, $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_2$ is a server with *no inputs*. $\mathcal{S}_0$ and C deterministically encrypt and then permute the dictionary D and token list T respectively and send them to $\mathcal{S}_2$. $\mathcal{S}_2$ then computes the intersection of D and T[1]. In this process, $\mathcal{S}_2$ only learns the cardinality of intersection and does not get any other information about the dictionary or the token list. Building on this technique, we construct a secure protocol using light-weight symmetric primitives only that outputs shares of featurized input vector to servers $\mathcal{S}_0$ and $\mathcal{S}_1$. Combining the constructed secure featurization protocol with 3-party secure inference protocols [25] gives us our end-to-end secure inference protocol.

2 PRELIMINARIES

Notation. The computational and statistical security parameters are denoted by λ and σ respectively. The function $\mathsf{neg}(\gamma)$ denotes a negligible function in γ. Operator $\|$ denotes concatenation and log (resp. ln) denotes logarithms with base 2 (resp. e). For a positive interger ℓ, $[\ell]$ denotes the set of integers $\{1,\ldots,\ell\}$. For a finite set X, $x \xleftarrow{\$} X$ denotes that x is uniformly sampled from X and $|X|$ denotes the cardinality of set X. We consider ordered sets or lists, and use $X[i]$ to denote i^{th} element of X. For a permutation $\pi : [n] \to [n]$ and a list X of size n, $Y \leftarrow \pi(X)$ is a permuted list of element in X where $Y[i] = X[\pi(i)]$, for all $i \in [n]$ and π^{-1} denotes the inverse permutation. For two lists X and Y of same size, $-X$ and $X \cdot Y$ denote element-wise negation of list X and element-wise product of lists X and Y, respectively. Similarly, for a function f, and a set X, we write $f(X)$ to denote $\{f(X[i])\}_{i\in|X|}$.

[1]The work of [20] explore PSI protocols in the server aided setting. However, their work does not obtain circuit PSI protocols where sets have associated payloads list, a necessary feature needed by our protocol.

2.1 Security Model

We consider protocols in the multi-party setting with $m \geqslant 2$ parties and security against static probabilistic polynomial time (PPT) semi-honest adversaries that corrupts one of the parties. We argue security in the standard the real/ideal simulation paradigm [16, 27]. That is, a protocol Π is said to securely realize a functionality $\mathcal{F}$ if for every real-word adversary $\mathcal{A}$ there exists an ideal-world adversary Sim such that the following two distributions are computationally indistinguishable:

- $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$: The parties execute protocol Π in the presence of $\mathcal{A}$ and the environment $\mathcal{Z}$. Output the binary distribution ensemble describing $\mathcal{Z}'s$ output in this interaction.
- $\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim},\mathcal{Z}}$: The parties send their inputs to a trusted functionality $\mathcal{F}$ that performs the computation faithfully. Let Sim be the adversary in this interaction. Output the binary distribution ensemble describing $\mathcal{Z}'s$ output in this interaction.

Moreover, we consider semi-honest security in a hybrid model [6] in which the parties additionally have access to an ideal functionality. By the universal composition theorem [6], if there exists a semi-honest secure protocol Π realizing functionality $\mathcal{F}$ in the $\mathcal{F}_1$-hybrid model, then protocol Π' that realizes functionality $\mathcal{F}$ can be constructed in the standard model by replacing call to $\mathcal{F}_1$ with the protocol Φ that securely realizes functionality $\mathcal{F}_1$.

2.2 Cryptographic Primitives

2.2.1 Pseudorandom Permutations and Symmetric Key Encryption. A pseudorandom permutation (PRP) $G : \{0,1\}^\lambda \times \{0,1\}^\mu \to \{0,1\}^\mu$ is a polynomial-time (in λ) computable function and is computationally indistinguishable from a truly random permutation for any PPT adversary [21].

A symmetric key encryption scheme $\mathcal{E}$ [21] for message space $\mathcal{M}$ and ciphertext space C comprises of a) $\mathsf{Gen}(1^\lambda)$, that outputs a secret-key k; b) $\mathsf{Enc}(k,m)$, that, encrypts the message $m \in \mathcal{M}$ using the secret key k to produce ciphertext $c \in C$; and c) $\mathsf{Dec}(k,c)$ that using k and a ciphertext c, outputs a message $m \in \mathcal{M}$ or $\perp$. Correctness holds if for all $k \leftarrow \mathsf{Gen}(1^\lambda)$ and for all messages $m \in \mathcal{M}$, $\mathsf{Dec}(k, \mathsf{Enc}(k,m)) = m$. We require $\mathcal{E}$ to satisfy the standard notion of RCPA security [7] (refer Appendix A.1 for details).

2.2.2 Secret Sharing. We use 2-out-of-2 additive secret sharing scheme [49] over rings $\mathbb{Z}_2$ and $\mathbb{Z}_L$, where $L = 2^\ell$, for some integer $\ell > 1$. For $(W,\mathcal{R}) \in \{(B,\mathbb{Z}_2),(A,\mathbb{Z}_L)\}$, a secret sharing scheme comprises of two algorithms: (1) Share^W takes as input an element $x \in \mathcal{R}$ and outputs random shares over $\mathcal{R}$ denoted by $\langle x\rangle_0^W$ and $\langle x\rangle_1^W$, with the only constraint that $\langle x\rangle_0^W + \langle x\rangle_1^W = x$. (2) $\mathsf{Reconstruct}^W$ takes shares $\langle x\rangle_0^W, \langle x\rangle_1^W \in \mathcal{R}$ as input and outputs the reconstructed value $x = \langle x\rangle_0^W + \langle x\rangle_1^W$. We refer to shares over $\mathbb{Z}_2$ and $\mathbb{Z}_L$ as boolean shares and arithmetic shares respectively.

2.2.3 Hashing. Simple hashing uses d many universal hash functions, $h_1,\ldots,h_d : \{0,1\}^* \mapsto [\beta]$, to map elements to bins in HT, where HT is a hash table comprising of β bins. An element x is inserted into bins $h_1(x),\ldots,h_d(x)$ in HT. Notice that a bin in hash table HT built using simple hashing can have more than one element. In contrast, cuckoo hashing [38] uses $d > 1$ many universal hash functions, $h_1,\ldots,h_d : \{0,1\}^* \mapsto [\beta]$ to map n_h elements to β

bins in hash table HT, where $\beta = O(n_h)$. The procedure to insert an element x to a bin in HT is as follows: If there exists an empty bin among $\mathsf{HT}[h_1(x)], \ldots, \mathsf{HT}[h_d(x)]$ then insert x in the lexicographically first empty bin. Otherwise, pick a random $i \in [d]$, evict the element present in $\mathsf{HT}[h_i(x)]$, insert x in bin $\mathsf{HT}[h_i(x)]$, and recursively try to insert the evicted element. If the number of evictions reach a certain threshold, the last evicted element is placed in a special bin called the *stash* that can hold multiple elements. Observe that, after cuckoo hashing, an element x can be found in one of the following bins: $h_1(x), \ldots h_d(x)$ or the stash, and each bin except the stash accommodates at most one element. For a discussion on the probability of stash overflow (i.e. insertion of $s + 1$ elements into a stash of size s), refer Appendix A.2.

2.3 Two-party Functionalities

2.3.1 Multiplexer. The functionality $\mathcal{F}_{\mathsf{MUX}}$ takes as input arithmetic shares of x over $\mathbb{Z}_L$, where $L = 2^\ell$, and boolean shares of choice bit b from P_0 and P_1. $\mathcal{F}_{\mathsf{MUX}}$ returns shares of x if $b = 1$, else it returns shares of 0 over $\mathbb{Z}_L$. We use the protocol of [44] that has a communication of $2(\lambda + 2L)$. We abuse notation and overload this functionality to also operate element wise over input vectors of equal size (over $\mathbb{Z}_2$ and $\mathbb{Z}_L$ respectively).

Parameters. A function g (possibly randomized) that takes as input a list of size n and outputs a map $f : [n] \rightarrow [cn]$, for a some constant $c \geqslant 1$. If g is randomized then it also takes random tape R as input. Let $\mathsf{SS}_A = (\mathsf{Share}^A, \mathsf{Reconstruct}^A)$ be a secret-sharing scheme over ring $\mathbb{Z}_L$ and $\mathsf{SS}_B = (\mathsf{Share}^B, \mathsf{Reconstruct}^B)$ be a boolean secret-sharing scheme.
Inputs of P_0. Input set $X = \{x_1, \ldots, x_{n_0}\}$, where $x_i \in \{0,1\}^*$, for all $i \in [n_0]$ and random tape R_{P_0}.
Inputs of P_1. Input set $Y = \{y_1, \ldots, y_{n_1}\}$, where $y_i \in \{0,1\}^*$, for all $i \in [n_1]$ and list of associated payloads V, where $V[i] \in \mathbb{Z}_L$ is the payload associated with element $y_i \in Y$.

The functionality does the following:

```
1. $f \leftarrow g(X, R_S)$.
2. Create lists $\mathcal{I}$, $Q$ of size $cn_0$. Set $\mathcal{I}[i] \leftarrow 0$ and
   $Q[i] \xleftarrow{\$} \mathbb{Z}_L, \forall i \in [cn_0]$.
3. for $i \in n_0$ do
4.     if $\exists y_j \in Y$ s.t. $x_i = y_j$ then
5.         $\mathcal{I}[f(i)] \leftarrow 1$ and $Q[f(i)] \leftarrow V[j]$.
6.     end
7. end
8. Compute $(\langle \mathcal{I} \rangle_0^B, \langle \mathcal{I} \rangle_1^B) \leftarrow \mathsf{SS}_B.\mathsf{Share}^B(\mathcal{I})$ and
   $(\langle Q \rangle_0^A, \langle Q \rangle_1^A) \leftarrow \mathsf{SS}_A.\mathsf{Share}^A(Q)$.
9. Send $\langle \mathcal{I} \rangle_b^B$ and $\langle Q \rangle_b^A$ to $P_b$, for $b \in \{0,1\}$.
```

Fig. 1: Circuit-PSI Functionality $\mathcal{F}_{\mathsf{CPSI}}$

2.3.2 Circuit-PSI. $\mathcal{F}_{\mathsf{CPSI}}$ [9, 17, 39–41, 47] is a two-party functionality that at a high level outputs secret shares of intersection of private sets of two parties. More formally, the functionality takes as input set X of size n_0 from P_0, and set Y of size n_1 and its associated payloads V from P_1. Existing protocols of circuit-PSI [9, 39–41, 47] that use Cuckoo hashing, consider a permutation, represented as an injective function f, that is known to P_0 and is function of X and P_0's randomness. P_0 uses cuckoo hashing scheme that maps elements in X to hash table HT of size $(1+\varepsilon)n_0$, where $\varepsilon \in (0,1)$. The injective function $f : [n_0] \rightarrow [cn_0]$, where $c = (1+\varepsilon)$, is defined as follows: $i \overset{f}{\mapsto} j$ s.t. element $\mathsf{HT}[j] = x_i$, where $i \in [n_0]$ and $j \in [cn_0]$. Now the $\mathcal{F}_{\mathsf{CPSI}}$ gives two secret shared vectors I, Q of length cn_0 as output to two parties, where I is a boolean vector (that stores 1 corresponding to elements in intersection) and Q is a arithmetic vector (that stores the shares of correponding payload). Formally, if $X[i] = Y[j]$, it sets $I[f(i)] = 1$ and $Q[f(i)] = V[j]$. Other indices in I and Q that do not correspond to elements in intersection, are set to 0 and uniformly random, respectively. We define the functionality formally in Fig. 1. Here, g outputs the injective map using X and randomness of P_0.

2.4 Function Secret Sharing

Function Secret Sharing (FSS) [4, 5] scheme for a function $f : I \rightarrow \mathbb{G}$, where I is the input domain and $(\mathbb{G}, +)$ is an abelian group which is the output domain, splits f into f_0 and f_1 s.t. $\forall x \in I, f_0(x) + f_1(x) = f(x)$ and each f_b computationally hides f. FSS is formally described in Definition 1.

Definition 1. *Function secret sharing (FSS) scheme F for functions in $\mathcal{F} = \{f : I \rightarrow \mathbb{G}\}$ is a pair of* PPT *algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *described below:*

- $\mathsf{Gen}(1^\lambda, f)$ *takes as input the description of function f and outputs two keys k_0, k_1.*
- $\mathsf{Eval}(b, k_b, x)$ *takes as input the party index $b \in \{0,1\}$, key k_b and input $x \in I$ and it outputs $y_b \in \mathbb{G}$.*

Let Leak : $\{0,1\}^* \rightarrow \{0,1\}^*$ *denote allowable leakage function that takes as input the description of function f and outputs the description of allowed leakage.*
Correctness: *For all $f \in \mathcal{F}$ and $x \in I$, if $(k_0, k_1) \leftarrow F.\mathsf{Gen}(1^\lambda, f)$ then $Pr[F.\mathsf{Eval}(0, k_0, x) + F.\mathsf{Eval}(1, k_1, x) = f(x)] = 1$.*
Security: *For any $b \in \{0,1\}$, there exists a* PPT *simulator* Sim *s.t. for every polynomial-size function sequence $(f_\lambda)_{\lambda \in \mathbb{N}}$ from $\mathcal{F}$, the two distributions given below are computationally indistinguishable:*

- $\{k_0, k_1 \leftarrow F.\mathsf{Gen}(1^\lambda, f_\lambda)$. *Output* $k_b\}$.
- $\{$*Output* $\mathsf{Sim}(1^\lambda, \mathsf{Leak}(f_\lambda))\}$.

In this work, we make use of Distributed Point Functions (DPF) which is an FSS scheme for point functions $f_{a,b} : \{0,1\}^\ell \rightarrow \mathbb{G}$ s.t. $f_{a,b}(a) = b$ and $f_{a,b}(x) = 0$ for any $x \neq a$. We use the DPF scheme from [5]. This DPF scheme leaks only the description of input domain and output group of the underlying point functions. $\mathsf{Gen}(\cdot)$ algorithm of this scheme requires 4ℓ AES evaluations and the keys k_0 and k_1 are each of size $\ell(\lambda+2)$. $\mathsf{Eval}(\cdot)$ algorithm makes 2ℓ AES evaluations.

3 SECURE FEATURIZATION

In this section, we first give a formal description of secure featurization functionality $\mathcal{F}_{\mathsf{SF}}$. Next, we present our secure featurization protocols in the one, two and three-server settings.

3.1 Functionality $\mathcal{F}_{\mathsf{SF}}$

We assume that both the server as well as the client have performed the tokenization process on their respective inputs (see Section 1). Thus, in the problem of secure featurization, the server ($\mathcal{S}_0$) has dictionary, i.e., a list of keywords, D of size n_0 and the client (C) has a list of tokens T of size n_1 with associated attributes E. Server and client must learn shares of a vector Q of size n_0 such that $Q[i] = E[j]$ if $D[i] = T[j]$, and 0 otherwise. Similar to the functionality of circuit-PSI (Section 2.3.2), we allow parties to learn these shares in a permuted order and allow length of Q to be larger than n_0 due to the use of Cuckoo Hashing. We define our functionality formally in Fig. 2.

Extending this to the multi-server setting, we consider a modified version of the $\mathcal{F}_{\mathsf{SF}}$ functionality. The multi-server setting comprises of m-servers $\mathcal{S}_0, \ldots, \mathcal{S}_{m-1}$ and a client C. In this modified version, the inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$ are the same as $\mathcal{S}_0$ in Fig. 2, the input of C remains the same and the remaining servers have *no inputs*. $\mathcal{F}_{\mathsf{SF}}$ outputs the shares of the associated attributes, i.e., $\langle Q\rangle_0^A$ and $\langle Q\rangle_1^A$ to servers $\mathcal{S}_0$ and $\mathcal{S}_1$ respectively. The other parties (including C) do not receive any output.

Remark. The nuances of permutation and larger size Q is only needed for the one-server protocol that builds on circuit-PSI protocols. For our protocols in two/three-server setting, $c = 1$ and there is no permutation f. That is, parties learn the shares of (simplified) vector Q described above in intuitive description.

Remark. In our protocols for two/three-server setting, parties additionally learn size of input of other parties, apart from just the final shares. We spell out these details in respective protocol descriptions and theorems. For ease of exposition, we avoid cluttering the functionality description with these details, as the main functionality is to learn shares of featurized input, and it is standard for protocols to reveal the size of honest party inputs (for efficiency).

Parameters. A function g (possibly randomized) that takes as input a list of size n and outputs a map $f : [n] \to [cn]$, for a some constant $c \geqslant 1$. If g is randomized then it also takes random tape R as input. Let SS = (Share^A, $\mathsf{Reconstruct}^A$) be a secret-sharing scheme over ring $\mathbb{Z}_L$.
Inputs of $\mathcal{S}_0$. A dictionary D of size n_0, where $D[i] \in \{0,1\}^*$, for all $i \in [n_0]$ and an optional random tape $R_{\mathcal{S}}$.
Inputs of C. A tokens list T of size n_1, where $T[i] \in \{0,1\}^*$, for all $i \in [n_1]$ and a list E of corresponding associated attributes in ring $\mathbb{Z}_L$.

The functionality does the following:

```
1. $f \leftarrow g(D, R_{\mathcal{S}})$.
2. Create list $Q$ of size $cn_0$. Set $Q[i] \leftarrow 0, \forall i \in [cn_0]$.
3. for $i \in n_0$ do
4.     if $\exists j \in [n_1]$ s.t. $D[i] = T[j]$ then
5.         $Q[f(i)] \leftarrow \mathsf{E}[j]$.
6.     end
7. end
8. Compute $(\langle Q\rangle_0^A, \langle Q\rangle_1^A) \leftarrow \mathsf{SS.Share}^A(Q)$. Send $\langle Q\rangle_0^A$ to
   $\mathcal{S}_0$ and $\langle Q\rangle_1^A$ to $C$.
```

Fig. 2: Secure Featurization Functionality $\mathcal{F}_{\mathsf{SF}}$

Parameters. Circuit-PSI Functionality $\mathcal{F}_{\mathsf{CPSI}}$ and multiplexer functionality $\mathcal{F}_{\mathsf{MUX}}$.
Inputs of $\mathcal{S}_0$. A dictionary D of size n_0, where $D[i] \in \{0,1\}^*$, for all $i \in [n_0]$ and a random tape $R_{\mathcal{S}}$.
Inputs of C. A tokens list T of size n_1, where $T[i] \in \{0,1\}^*$, for all $i \in [n_1]$ and a list E of corresponding associated attributes in ring $\mathbb{Z}_L$.

1. $\mathcal{S}_0$ and C invoke $\mathcal{F}_{\mathsf{CPSI}}$ functionality where $\mathcal{S}_0$ plays the role of P_0 with inputs D and $R_{\mathcal{S}}$, and C plays the role of P_1 with inputs T and E. $\mathcal{S}$ receives $\mathcal{I}_0$, Q_0 and C receives $\mathcal{I}_1$, Q_1 from $\mathcal{F}_{\mathsf{CPSI}}$ functionality.
2. $\mathcal{S}_0$ and C invoke the functionality $\mathcal{F}_{\mathsf{MUX}}$ where the input of $\mathcal{S}_0$ is $\mathcal{I}_0$ and Q_0 and the input of C is $\mathcal{I}_1$ and Q_1. Let U_0 and U_1 denote the output received by $\mathcal{S}_0$ and C from $\mathcal{F}_{\mathsf{MUX}}$ functionality .
3. $\mathcal{S}_0$ and C output the lists U_0 and U_1 respectively.

Fig. 3: One-Server Secure Featurization Protocol

3.2 One-Server Protocol

We present our secure featurization protocol in the one-server setting. Our protocol makes use of functionalities $\mathcal{F}_{\mathsf{CPSI}}$ and $\mathcal{F}_{\mathsf{MUX}}$. This protocol provides the strongest privacy guarantee for the client but has the highest computational/communication overhead for the client compared to the protocols in two/three-server settings.

Protocol Overview. Recall that $\mathcal{F}_{\mathsf{CPSI}}$ outputs boolean shares of intersection and arithmetic shares of the associated payloads in some permuted order. Note that the only difference in functionalities of secure featurization and circuit-PSI is that for elements not in intersection, secure featurization requires the value to be 0 whereas it is a uniformly random value in circuit-PSI. However, the boolean vector in circuit-PSI captures exactly the information whether the index is in intersection or not. Hence, a protocol for $\mathcal{F}_{\mathsf{SF}}$ can be constructed using $\mathcal{F}_{\mathsf{CPSI}}$ and $\mathcal{F}_{\mathsf{MUX}}$ functionalities in a straight-forward manner. The boolean shares of intersection and the arithmetic shares of associated payloads returned by $\mathcal{F}_{\mathsf{CPSI}}$ act as the boolean shares of the choice bits and the input arithmetic shares in the call to $\mathcal{F}_{\mathsf{MUX}}$. The protocol is described in Fig. 3.

Complexity. The complexity of the proposed protocol is the sum of the complexities of Circuit-PSI protocol [9] and Multiplexer protocol [44] for appropriate size inputs.

We give the correctness and security proof of our protocol in Fig. 3 in Appendix B that formally proves the theorem below.

THEOREM 1. *The protocol in Fig. 3 securely realizes $\mathcal{F}_{\mathsf{SF}}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}_{\mathsf{CPSI}}, \mathcal{F}_{\mathsf{MUX}})$, against semi-honest corruption of one party $P \in \{\mathcal{S}_0, C\}$.*

3.3 Two-Server Protocol

While one-server featurization protocol provides the strongest security guarantee, it may not satisfy the efficiency requirements of many real-time applications. Moreover, the computation time of C in our one-server protocol is $O(n_0)$. Typically, the size of tokens list (n_1) is orders of magnitude less than the size of the dictionary (n_0). In many applications, the client machine is of low configuration, e.g., settings where the inference protocol is run on hand-held devices by C. For such applications, it is ideal to have C's computation time to be dependent on its input size only and should be independent of the dictionary size. In this section, we propose a secure featurization protocol in two-server setting which is orders of magnitude more efficient than our protocol in one-server setting. Further, the computation time of C in our two-server protocol is independent of the dictionary size. However, the two-server setting provides weaker client privacy guarantees than the one-server setting as client's input privacy is guaranteed only when the two servers $\mathcal{S}_0$ and $\mathcal{S}_1$ are non-colluding. Our two-server protocol makes use of FSS for point functions, i.e., DPF scheme, and cuckoo hashing.

Protocol Overview. In the two-server setting, both servers $\mathcal{S}_0$ and $\mathcal{S}_1$ hold the same dictionary D. Recall that, DPF scheme $F = (\mathsf{Gen}, \mathsf{Eval})$ is an FSS scheme for point functions in family $\mathcal{F} = \{f_{a,b}(\cdot) | f_{a,b} : \{0,1\}^\mu \to \mathbb{Z}_L$ with $f_{a,b}(a) = b$ and $f_{a,b}(x) = 0$ for $x \neq a\}$. We first describe a basic scheme that requires each server to perform $n_0 n_1$ DPF evaluations and then show how to reduce this to be independent of n_1. For each token-attribute pair $(T[j], \mathsf{E}[j])$, where $j \in n_1$, C generates DPF keys $K_0[j]$ and $K_1[j]$ for point function $f_{T[j],\mathsf{E}[j]}$ and then sends list of keys K_b to $\mathcal{S}_b$, where $b \in \{0,1\}$. For each keyword $D[i]$, where $i \in n_0$, $\mathcal{S}_b$ computes the sum of evaluations of DPF scheme on $D[i]$ using each key $K_b[j]$. For keyword $D[i]$ in tokens list T such that $D[i] = T[j]$, the DPF evaluations using key-pair $K_0[j]$ and $K_1[j]$ output shares of attribute $\mathsf{E}[j]$, whereas for the remaining key-pairs, the DPF evaluations output shares of 0. Hence, the sum of shares of all DPF evaluations on keyword $D[i]$ is shares of attribute $\mathsf{E}[j]$. Analogously, for keyword $D[i]$ not in T, the DPF evaluations on all key-pairs output shares of 0. As a result, the sum of shares of all DPF evaluations on keyword $D[i]$ in this case is shares of 0. The limitation of using this simple approach is that each server has to perform $n_0 n_1$ DPF evaluations. We now show how to use cuckoo hashing to reduce the number of DPF evaluations to dn_0, where d denotes the number of hash functions in cuckoo hashing scheme[2]. The complete two-server secure featurization protocol is described in Fig. 4.

Observe that in this protocol, servers additionally learn the size of tokens list, i.e., n_1 (from the size of lists K_b and output length μ of hash function h). Similarly, the client learns the dictionary size n_0 again from the output length of hash function[3].

Complexity. We make use of the single-point DPF scheme proposed in [5] to instantiate our protocol in Fig. 4. The only communication that takes place in our two-server secure featurization protocol is in step 14. Hence, the communication complexity of C is $2(1+\varepsilon)(\lambda+2)n_1\mu$. Whereas, the communication cost incurred at $\mathcal{S}_0$'s and $\mathcal{S}_1$'s end is $(1+\varepsilon)(\lambda+2)n_1\mu$. C does $4(1+\varepsilon)n_1\mu$ AES evaluations whereas each server does $6n_0\mu$ AES evaluations.

Parameters. Let h: $\{0,1\}^* \to \{0,1\}^\mu$ be a universal hash function, where $\mu = \sigma + \log(n_0) + \log(n_1) + 1$; let h_1, h_2 and h_3: $\{0,1\}^\mu \mapsto [\beta]$ be three universal hash functions, where $\beta = (1+\varepsilon)n_1$ and $\varepsilon \in (0,1)$; let $F = (\mathsf{Gen}, \mathsf{Eval})$ be a DPF scheme for point functions whose inputs are of length μ and its output domain is $\mathbb{Z}_L$.
Inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$. A dictionary D of size n_0, where $D[i] \in \{0,1\}^*$, for all $i \in [n_0]$.
Inputs of C. A tokens list T of size n_1, where $T[i] \in \{0,1\}^*$, for all $i \in [n_1]$ and a list E of corresponding associated attributes in ring $\mathbb{Z}_L$.

```
1. Servers $\mathcal{S}_0$ and $\mathcal{S}_1$ create list $A$ of length $n_0$ such that
   $A[i] = h(D[i])$, for all $i \in [n_0]$.
2. $C$ creates list $B$ of length $n_1$ such that $B[i] = h(T[i])$, for
   all $i \in [n_1]$.
3. $C$ builds hash table $\mathsf{HT}_C$ with $\beta$ bins by mapping
   elements in $B$ to $\mathsf{HT}_C$ using Cuckoo hashing with hash
   functions $h_1, h_2, h_3$. Since $\beta > n_1$, $C$ fills the empty bins in
   $\mathsf{HT}_C$ with a dummy value.
4. $C$ creates list of DPF keys $K_0$ and $K_1$ as follows:
5. for $j \in [\beta]$ do
6.     if $\mathsf{HT}_C[j] \in T$ then
7.         Let $i \in [n_1]$ be the index s.t. $\mathsf{HT}_C[j] = h(T[i])$.
8.         $(k_0, k_1) \leftarrow F.\mathsf{Gen}(1^\lambda, \mathsf{HT}_C[j], \mathsf{E}[i], \mathbb{Z}_L)$.
9.     else
10.        $(k_0, k_1) \leftarrow F.\mathsf{Gen}(1^\lambda, \mathsf{HT}_C[j], 0, \mathbb{Z}_L)$.
11.    end
12.    Set $K_0[j] \leftarrow k_0$ and $K_1[j] \leftarrow k_1$.
13. end
14. $C$ sends $K_b$ to $\mathcal{S}_b$, for all $b \in \{0,1\}$.
15. For $b \in \{0,1\}$, $\mathcal{S}_b$ computes list $\mathsf{FS}_b$ of size $n_0$ as follows:
16. Create list $\mathsf{FS}_b$ of size $n_0$. For all $i \in [n_0]$, set $\mathsf{FS}_b[i]$ to 0.
17. for $t \in [n_0]$ do
18.     $\mathsf{PSet} = \{h_1(A[t]), h_2(A[t]), h_3(A[t])\}$.
19.     for pos $\in \mathsf{PSet}$ do
20.         $\mathsf{FS}_b[t] \leftarrow \mathsf{FS}_b[t] + F.\mathsf{Eval}(b, K_b[\mathrm{pos}], A[t])$.
21.     end
22. end
23. For $b \in \{0,1\}$, $\mathcal{S}_b$ outputs $\mathsf{FS}_b$.
```

Fig. 4: Two-Server Secure Featurization Protocol

We give complete correctness and security proof of our protocol in Fig. 4 in Appendix C. Formally, we prove the theorem below.

THEOREM 2. *Given a secure DPF scheme, the protocol in Fig. 4 securely realizes $\mathcal{F}_{\mathsf{SF}}$ in the two-server setting, where $\mathcal{F}_{\mathsf{SF}}$ parameterized with a constant function g that outputs identity map $f : [n_0] \to [n_0]$. Moreover, along with secret shares of featurized input, the servers learn the size of client's token list and client learns the size of the dictionary as additional outputs.*

[2]Our use of single point DPFs with hashing techniques to obtain shares of associated attributes for elements in intersection of the two lists is similar to the approach used in construction of multi-point DPFs in [48].
[3]The output length of the hash function, h, is chosen such that the probability of collision for two elements is negligible in statistical security parameter, and hence, depends on the total number of elements in dictionary and token list.

3.4 Three-Server Protocol

Although our two-server protocol provides drastic performance improvement over our one-server protocol, the overhead for the client in this protocol is still higher than that desired by many applications. This is because the client generates and communicates DPF keys whose number is proportional to the size of its token list. We address this bottleneck by considering the three-server setting. This three-server protocol has very low latency and C is super-light weight, albeit the protocol provides a weaker privacy guarantee for the client. In particular, the privacy of the client requires that no two servers collude.

Protocol Overview. In the three-server setting, there are 3 servers, $\mathcal{S}_0$, $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_0$ and $\mathcal{S}_1$ holds the dictionary D and $\mathcal{S}_2$ has no input. $\mathcal{S}_0$ applies pseudorandom permutation G to all keywords in the dictionary D and permutes the obtained list using a random permutation. $\mathcal{S}_0$ then sends this processed dictionary to $\mathcal{S}_2$. C processes its token list T in a similar way using G and same key, and sends the processed tokens list to $\mathcal{S}_2$. As a result, $\mathcal{S}_2$ can easily identify indices of entries in the processed dictionary that belong in the intersection of dictionary D and tokens list T. Observe that, only the cardinality of intersection is leaked to $\mathcal{S}_2$ in this computation as $\mathcal{S}_2$ learns no information about individual entries in either of the processed list (from PRP security) or the order of entries in the processed list (due to the application of random permutation).

Now, client C handles the attributes of the tokens as follows: First, C generates secret-shares of the attributes E associated with tokens in T, E_0 and E_1. It encrypts E_0. C sends a permuted list of E_1 and a permuted list of encryption of E_0, using the same random permutation used to permute entries in T, to $\mathcal{S}_2$. C sends the encryption key to $\mathcal{S}_0$ and $\mathcal{S}_1$. $\mathcal{S}_2$ first computes the share of featurized input vector for $\mathcal{S}_1$ by using the values in E_1 for elements in intersection and a random value for indices not in intersection. It sends these to $\mathcal{S}_1$. Next, $\mathcal{S}_1$ encrypts the additive inverse of shares obtained from $\mathcal{S}_2$ and sends it back to $\mathcal{S}_2$. Finally, $\mathcal{S}_2$ creates share of featurized input for $\mathcal{S}_0$ as follows: For elements in intersection, it uses the encryption of E_0 obtained from C. For elements not in intersection, it uses the encyption obtained from $\mathcal{S}_1$ (this gives additive shares of 0). $\mathcal{S}_0$ decrypts its share using the key obtained from the client. The protocol is formally described in Fig. 5.

In this protocol, setting the optimal output length of the hash function results in $\mathcal{S}_0$ learning the size of token list and C learning the size of the dictionary. Moreover, the server $\mathcal{S}_2$ also learns the size of token list from the message received from C and also the size of the intersection.

Complexity. The communication complexity of C is $n_1(\lambda+\mu+\ell)+2\lambda$, where $\ell = \log(L)$ and it performs $2n_1$ many AES evaluations. $\mathcal{S}_0$ has a communication complexity of $n_0(\lambda+\mu)+2\lambda$ and does $2n_0$ many AES Evaluations. $\mathcal{S}_1$ incurs a communication cost of $n_0(\lambda+\ell)+\lambda$ and makes n_0 many AES Evaluations. The communication cost incurred by $\mathcal{S}_2$ is $n_0(2\lambda+\mu+\ell)+n_1(\lambda+\mu+\ell)$ and doesn't perform any AES evaluations.

We give correctness and security proof of our protocol in Fig. 5 in Appendix D. Formally, we prove the theorem below.

Parameters. Let h: $\{0,1\}^* \to \{0,1\}^\mu$ be a universal hash function, where $\mu = \sigma + \log(n_0) + \log(n_1)$; let $G : \{0,1\}^\lambda \times \{0,1\}^\mu \to \{0,1\}^\mu$ be a PRP; let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an RCPA secure symmetric key encryption scheme; and finally, let secret-sharing scheme SS $= (\mathsf{Share}^A, \mathsf{Reconstruct}^A)$ over ring $\mathbb{Z}_L$.
Inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$. A dictionary D of size n_0, which is a keywords list where $D[i] \in \{0,1\}^*$, for all $i \in [n_0]$.
Inputs of C. A tokens list T of size n_1, where $T[i] \in \{0,1\}^*$, for all $i \in [n_1]$ and a list E of corresponding associated attributes in ring $\mathbb{Z}_L$.

```
1.  Servers $\mathcal{S}_0$ create a list $A$ of length $n_0$ such that
    $A[i] = h(D[i])$, for all $i \in [n_0]$.
2.  $C$ creates list $B$ of length $n_1$ such that $B[i] = h(T[i])$, for
    all $i \in [n_1]$.
3.  $C$ generates $k_G \xleftarrow{\$} \{0,1\}^\lambda$ and $k_\mathcal{E} \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$. $C$
    sends $k_G$ to $\mathcal{S}_0$ and sends $k_\mathcal{E}$ to both $\mathcal{S}_0$ and $\mathcal{S}_1$.
4.  $\mathcal{S}_0$ selects a random-permutation $\pi_A$ on $[n_0]$ and sends
    $\pi_A$ to $\mathcal{S}_1$. $\mathcal{S}_0$ computes list $A' = \pi_A(G(k_G, A))$ and sends
    $A'$ to $\mathcal{S}_2$.
5.  $C$ selects a random-permutation $\pi_B$ on $[n_1]$ and creates
    list $B' = \pi_B(G(k_G, B))$ and sends $B'$ to $\mathcal{S}_2$.; $C$ computes
    $(E_0, E_1) \leftarrow \mathsf{SS.Share}^A(E)$.
6.  $C$ creates two other lists $EF_0$ and $F_1$ of size $n_1$ as follows:
    $EF_0 = \pi_B(\mathcal{E}.\mathsf{Enc}(k_\mathcal{E}, E_0))$ and $F_1 = \pi_B(E_1)$. Finally, $C$
    sends list $EF_0$ and $F_1$ to $\mathcal{S}_2$.
7.  $\mathcal{S}_2$ computes an indicator list $\mathcal{I}$ of size $n_0$, where in,
    $\mathcal{I}[i] = 1$ if $A'[i] \in B'$ and $\mathcal{I}[i] = 0$ otherwise.
8.  $\mathcal{S}_2$ creates list $\mathbf{F}_1$ of size $n_0$ as described below and sends
    it to $\mathcal{S}_1$.
9.  for $i \in [n_0]$ do
10.     if $\mathcal{I}[i] = 1$ then
11.         Let $j \in [n_1]$ be an index s.t. $A'[i] = B'[j]$. Set
            $\mathbf{F}_1[i] \leftarrow F_1[j]$.
12.     else
13.         Set $\mathbf{F}_1[i] \xleftarrow{\$} \mathbb{Z}_L$.
14.     end
15. end
16. $\mathcal{S}_1$ computes $\mathbf{EF}_1 \leftarrow \mathcal{E}.\mathsf{Enc}(k_\mathcal{E}, -\mathbf{F}_1)$. $\mathcal{S}_1$ sends $\mathbf{EF}_1$ to $\mathcal{S}_2$.

17. $\mathcal{S}_2$ creates list $\mathbf{EF}_0$ of size $n_0$ as described below and sends
    it to $\mathcal{S}_0$.
18. for $i \in [n_0]$ do
19.     if $\mathcal{I}[i] = 1$ then
20.         Let $j \in [n_1]$ be an index s.t. $A'[i] = B'[j]$. Set
            $\mathbf{EF}_0[i] \leftarrow EF_0[j]$.
21.     else
22.         Set $\mathbf{EF}_0[i] \leftarrow \mathbf{EF}_1[i]$.
23.     end
24. end
25. $\mathcal{S}_0$ computes $\mathbf{F}_0 \leftarrow \mathcal{E}.\mathsf{Dec}(k_\mathcal{E}, \mathbf{EF}_0)$.
26. For $b \in \{0,1\}$, $\mathcal{S}_b$ outputs $\mathbf{F}'_b \leftarrow \pi_A^{-1}(\mathbf{F}_b)$.
```

Fig. 5: Three-Server Secure Featurization Protocol

THEOREM 3. *Given a pseudorandom permutation, an RCPA-secure secret-key encryption scheme and a secret-sharing scheme over ring $\mathbb{Z}_L$, the protocol in Fig. 5 securely realizes $\mathcal{F}_{\mathsf{SF}}$ in the three-server setting where $\mathcal{F}_{\mathsf{SF}}$ parameterized with a constant function g that outputs identity map $f : [n_0] \rightarrow [n_0]$. Moreover, additionally, $\mathcal{S}_0$ and $\mathcal{S}_2$ learn the size of token list (n_1) $\mathcal{S}_2$ learns the size of intersection, i.e., $|D \cap T|$, and C learns size of the dictionary (n_0).*

4 END-TO-END SECURE INFERENCE

In this section, we discuss how to obtain end-to-end secure inference protocols using our secure featurization protocols and existing secure inference protocols [25, 44, 51] that work on featurized inputs. Denote the end-to-end secure inference functionality by $\mathcal{F}_{\mathsf{GSI}}$, for general secure inference.

Problem Statement. In one-server setting, $\mathcal{S}_0$ sends model M (with weights w) and dictionary D to $\mathcal{F}_{\mathsf{GSI}}$, and C sends tokens list T and the list of associated attributes E to $\mathcal{F}_{\mathsf{GSI}}$. $\mathcal{F}_{\mathsf{GSI}}$ outputs the inference output $M(w, x)$ to C, where x is the featurized input (and is the output of $\mathsf{F}(D, T, \mathsf{E})$).

Similar to secure featurization, we consider a modified version of end-to-end secure inference for multi-server settings. In the modified version, the inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$ are the same as that of $\mathcal{S}_0$ in one-server $\mathcal{F}_{\mathsf{GSI}}$ functionality, the input of C remains the same and the remaining servers have no inputs. The output of modified $\mathcal{F}_{\mathsf{GSI}}$ is same as that of one-server $\mathcal{F}_{\mathsf{GSI}}$, i.e., only C receives the inference output. As we will discuss below, our protocol in the multi-server setting provides stronger security for the server, namely, hides the model description M as well from the client.

4.1 One-server

We first formalize the secure inference functionality $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$ that operates on featurized inputs. $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$ takes as input Model M (with model weights w) from P_0 and shares of featurized input vector x, i.e., $\langle x \rangle_0^A$ and $\langle x \rangle_1^A$ from P_0 and P_1 respectively and outputs $M(w, x)$ to P_1. There is a long line of work that focuses on the problem of secure inference, given featurized inputs at the client's end [19, 29, 33, 44] (and references therein). These protocols can be easily modified to work in settings where the client featurized input is secret shared between 2-parties, as is the case for secure featurization. Moreover, all these works assume that the model description, i.e., is public or known to both parties running secure inference. In our setting, this can be viewed as additional output of the client. This additional information may or may not be acceptable depending on the application, and our protocols for multi-server settings, described in next subsections, avoid this leakage to the client. Finally, in the one-server setting, our implementation makes use of the state-of-the-art work for 2-party secure inference, CrypTFlow2 [44], to realize $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$ functionality as described below.

In end-to-end secure inference in one-server setting, $\mathcal{S}_0$ has model M, model-weights w, and dictionary D and C has tokens list T and the list of associated attributes E. $\mathcal{S}_0$ samples randomness $R_{\mathcal{S}}$. First, $\mathcal{S}_0$ and C invoke the one-server secure featurization protocol (Fig. 3) with $\mathcal{S}$'s input as D and $R_{\mathcal{S}}$ and C's input as T and E. The secure featurization protocol outputs the shares of the associated attribute, U_0 and U_1, in a permuted order defined by function f of $\mathcal{F}_{\mathsf{CPSI}}$ functionality (see line 1 of Fig. 1). However, the obtained shares of featurized inputs cannot be directly fed to $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$ as they are permuted.

Using the description of function f, $\mathcal{S}_0$ and C can re-order the lists U_0 and U_1 to obtain lists of shares in the order of the dictionary D. As f is computed using $\mathcal{S}_0$'s input and random tape $R_{\mathcal{S}}$, $\mathcal{S}_0$ can obtain function f locally and it can then send f to C. However, f depends on the input of $\mathcal{S}_0$. Thus, the above approach is not secure. Instead, $\mathcal{S}_0$ does the following: It prepends the model M with a linear layer of dimension $n_0 \times cn_0$ that un-permutes the feature vector and removes the entries for dummy elements. The weights of this layer are also kept secret with $\mathcal{S}_0$. In cases where the first layer of M is itself a fully connected linear layer (of dimension, say, $m \times n_0$) we can perform the following optimization: server $\mathcal{S}_0$ multiplies the above additional layer with the first layer in M to obtain a fully connected layer of dimension $m \times cn_0$. Denote the modified weights vector by w'.

$\mathcal{S}_0$ plays the role of P_0 and C plays the role of P_1 in call to $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$ functionality. The inputs of $\mathcal{S}$ are M, w' and U_0 from $\mathcal{S}$ and that of C is U_1. C then obtains the inference output from $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$.

The protocol provides strongest privacy guarantee from client's perspective. However, the privacy guarantee is not the strongest from $\mathcal{S}_0$'s perspective. This is because apart from C learning the dictionary size n_0 on executing the one-server secure featurization protocol, C also learns the description of model M in execution of $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$ functionality.

4.2 Two-server

At the end of two-server secure featurization, servers $\mathcal{S}_0$ and $\mathcal{S}_1$ learn the secret shares of the featurized input and will run the subsequent secure inference protocol. In this setting, we consider an alternate secure inference functionality $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}'}$ where the input of P_0 (resp. P_1) is model M, model weights w and P_0's (resp. P_1's) share of featurized input vector $\langle x \rangle_0^A$ (resp. $\langle x \rangle_1^A$). P_0 and P_1 receive shares of inference result $M(w, x)$ from $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}'}$ functionality. We make use of CrypTFlow2 secure inference framework [44] to realize $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}'}$ functionality. Compared to $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}}$, $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}'}$ can be realized more efficiently because now the weights of linear layers are known to both the parties that allow the linear layers to be computed without communication and at the same cost of computing the linear layer on cleartext.

In end-to-end secure inference in two-server setting, the inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$ are the model M, model weights w and dictionary D and the inputs of C are tokens list T and the list of associated attributes E. $\mathcal{S}_0$, $\mathcal{S}_1$ and C first invoke the two-server secure featurization protocol (Fig. 4) with input of $\mathcal{S}_0$ and $\mathcal{S}_1$ as D, and input of C as T and E. $\mathcal{S}_0$ and $\mathcal{S}_1$ obtain shares of the featurized input FS_0 and FS_1 respectively. Unlike the one-server secure featurization protocol, our two-server secure featurization protocol outputs the shares of the associated attribute in the same order as that of the dictionary D. $\mathcal{S}_0$ (resp. $\mathcal{S}_1$) with inputs model M with weights w and FS_0 (resp. FS_1) play the role of P_0 (resp. P_1) in $\mathcal{F}_{\mathsf{SI}}^{\mathsf{2PC}'}$ functionality and obtain secret shares of the inference output. $\mathcal{S}_0$ and $\mathcal{S}_1$ then relay these shares to C, who can then reconstruct to learn the inference output.

The protocol provides weaker security for C as the security relies on non-collusion assumption of the servers. However, it provides a stronger security for the model owner as the description of model M is not revealed to C. The two-server protocol is orders of magnitude faster than the one-server protocol due to faster secure featurization as well as faster secure inference (as the model is known to both parties performing the computation). Additionally, as C does not take part in *inference phase*, its computation/communication overhead is low and independent of machine learning model being used.

4.3 Three-server

At the end of three-server secure featurization, servers $\mathcal{S}_0$ and $\mathcal{S}_1$ learn the secret shares of the featurized input, and $\mathcal{S}_2$ has no output and all three servers will run the subsequent secure inference protocol. We consider a 3-party secure inference functionality $\mathcal{F}_{\mathsf{SI}}^{\mathsf{3PC}'}$, where P_0's input are model M, model weights w and P_0's share of featurized input vector $\langle x\rangle_0^A$, P_1's inputs are model M, model weights w and P_1's share of featurized input vector $\langle x\rangle_1^A$, and P_2 has no input. The functionality outputs shares of $M(x,w)$ to P_0 and P_1 and P_2 has no output. To realize this, we optimize the 3-party protocol Porthos [25, 51] for the setting when weights of the model are known to P_0, P_1. Overall, due to the use of helper party, i.e., P_2, this secure inference is much more efficient than the two-server setting considered above.

In end-to-end secure inference in three-server setting, the inputs of $\mathcal{S}_0$ and $\mathcal{S}_1$ are the model M, model weights w and dictionary D, the inputs of C are tokens list T and the list of associated attributes E, and $\mathcal{S}_2$ has no inputs. $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$ and C first invoke the three-server secure featurization protocol (Fig. 5) with input of $\mathcal{S}_0$ and $\mathcal{S}_1$ as D, and input of C as T and E. $\mathcal{S}_0$ and $\mathcal{S}_1$ obtain shares of the featurized input $\mathbf{F}'_0$ and $\mathbf{F}'_1$ respectively. Similar to our two-server featurization protocol, our three-server secure featurization protocol outputs the shares of the associated attribute in the same order as that of the dictionary D. $\mathcal{S}_0$ (resp. $\mathcal{S}_1$) with inputs model M with weights w and $\mathbf{F}'_0$ (resp. $\mathbf{F}'_1$) play the role of P_0 (resp. P_1) and $\mathcal{S}_2$ plays the role of P_2 in $\mathcal{F}_{\mathsf{SI}}^{\mathsf{3PC}'}$ functionality. $\mathcal{S}_0$ and $\mathcal{S}_1$ obtain secret shares of the inference output. $\mathcal{S}_0$ and $\mathcal{S}_1$ then relay these shares to C, who can then reconstruct the inference output.

Though the protocol in this setting provides the weakest security guarantee for the client among the settings we consider, it is the most performant and C is very light-weight in this protocol. Similar to two-server setting, the client does not participate in secure inference protocol and its complexity is independent of the complexity of the machine learning model used. Furthermore, client learns nothing about model structure as well.

5 IMPLEMENTATION AND EVALUATION

In this section, we discuss the performance of our secure featurization protocols and end-to-end secure inference protocols obtained using our secure featurization protocols. We consider the application of secure phishing detection to demonstrate the performance results. We set statistical security parameter $\sigma = 40$ and computational security parameter $\lambda = 128$.

Implementation Details. We implement our secure featurization and end-to-end secure inference protocols in C++. We use the implementation of circuit-PSI protocol [9] available at [35]. We make use of the implementation of $\mathcal{F}_{\mathsf{MUX}}$ functionality [44] available at [34]. For PRP G and RCPA secure-SKE scheme $\mathcal{E}$, we use the implementation of AES function available in OpenSSL library [50]. We implement DPF scheme [5] using libOTe library [37]. We use the implementation of Cuckoo Hashing [22, 38] available at [36]. Finally, we use the implementation of CrypTFlow2 framework [44] and Porthos Protocols [25] available at [34].

Protocol Parameters. We instantiate circuit-PSI protocol of [9] with appropriate parameters to achieve statistical security of 40 bits in our one-server secure featurization protocol. For two-server secure featurization scheme, the parameters in cuckoo hashing are set as discussed in Section 2.2 to obtain failure probability in no stash setting to be at most 2^{-41} and μ is set to $\sigma+\log(n_0)+\log(n_1)+1$. This ensures that our two-server secure featurization protocol achieves statistical security of 40 bits. Finally, we set $\mu = \sigma+\log(n_0)+\log(n_1)$ to obtain statistical security of 40-bits in our three-server protocol.

Evaluation Setup. We use machines with commodity class hardware: Intel Xeon E5 2.4GHz CPU with 16GBs of RAM for each party: $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2$ and C. We ran our experiments in a network setting with observed network bandwidth of 233 MBps and latency of 0.6 ms.

5.1 Description of Dataset and ML Models

To assess the performance of our proposed protocols, we build two phishing detection ML models. The dataset we have access to comprises of 10,000 manually labeled records of which 1,000 records are malicious (identified to be phishing attempts). The dataset is split into 70% training and 30% test dataset. The split is done in such a way that, the training dataset and test dataset contain the same proportion of malicious records. We consider the following raw texts from a dataset record: Webpage URL, Title and Body. We apply *bag-of-words* featurization algorithm on the aforementioned raw texts of the dataset and obtain a dictionary D of size 21,413.

On the training dataset, we build two different classification models with significant model complexity differences: 1) Logistic Regression (LR) (21,414 parameters). 2) A 4-layer deep Neural Network (4NN) that comprises of 21,413 input nodes, 4 fully-connected layers with 100, 50, 10 and 5 units respectively, each fully-connected layer is followed with ReLU activation function, and a final softmax output layer (2,147,021 parameters). These two ML Models represent two-ends of the class of production grade ML Models. While LR Model is the simplest, 4NN Model is a complex multi-layered neural network. Without performing any hyper-parameter tuning and extensive feature selection, the LR Model has a positive recall of 80% on the training dataset; and it has a positive recall and precision of 76% and 97% respectively on the test dataset. Similarly, the 4NN Model has a positive recall of 99% on the training dataset; and it has a positive recall and precision of 91% and 95% respectively on the test dataset[4].

[4]Positive recall denotes the fraction of URLs that were classified as phish by the algorithm out of all true phish URLs; while precision denotes the fraction of true phish URLs out of all URLs that were classified as phish by the algorithm.

Protocol	CPU Execution Time (ms)				Runtime (ms)	Communication (MB)				Total Communication (MB)
	$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	C		$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	C	
1-server	3660	–	–	3662	3662	146.59	–	–	146.59	146.59
2-server	264	264	–	3	264	0.09	0.09	–	0.18	0.18
3-server	31	12	40	0.3	50	0.77	0.60	1.20	2.8×10^{-3}	1.28

Table 1: Execution time (in ms) and Communication (in MB) of Secure Featurization Protocols.

Protocol	CPU Execution Time (ms)				Runtime (ms)	Communication (MB)				Total Communication (MB)
	$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	C		$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	C	
1-server	3842	–	–	3836	3842	167.24	–	–	167.24	167.24
2-server	264	264	–	3	264	0.13	0.13	–	0.18	0.22
3-server	34	17	40	0.4	57	0.77	0.60	1.20	2.8×10^{-3}	1.29

Table 2: Execution time (in ms) and Communication (in MB) of end-to-end secure inference with LR Model.

Protocol	CPU Execution Time (ms)				Runtime (ms)	Communication (MB)				Total Communication (MB)
	$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	C		$\mathcal{S}_0$	$\mathcal{S}_1$	$\mathcal{S}_2$	C	
1-server	10477	–	–	10472	10477	1301.36	–	–	1301.36	1301.36
2-server	450	450	–	3	450	0.78	0.78	–	0.18	0.87
3-server	55	47	40	0.4	82	0.82	0.65	1.28	2.8×10^{-3}	1.38

Table 3: Execution time (in ms) and Communication (in MB) of end-to-end secure inference with 4NN Model.

5.2 Secure Featurization

In this section, we report the performance of our secure featurization protocols on dictionary D of size $21,413$ and token list[5] T of size 73. We report performance for only one dictionary size (n_0) and token list size (n_1). As the performance of our one-server protocol is linear in n_0 and that of the two-server and three-server is linear in $n_0 + n_1$, one can estimate their performance for varying dictionary and input sizes.

Along with end-to-end runtime and total communication, we summarize the CPU execution time and communication of individual participating parties in our protocols in Table 1. The observed runtime of one-server, two-server and three-server secure featurization protocol is 3662 ms, 264 ms and 50 ms respectively. Runtime of one-server protocol is orders of magnitude higher than the other two protocols due to the use of relatively expensive $\mathcal{F}_{\text{CPSI}}$ functionality. Whereas, our two-server and three-server protocols involve light-weight AES computations. Though the dominant cost in both the protocols is AES invocations, the runtime of two-server protocol is more than three-server protocol because the number of AES computations in two-server protocol is $O(\mu(n_0 + n_1))$ and that in three-server protocol is $O(n_0 + n_1)$. However, the total communication of two-server protocol (0.18 MB) is lower than the total communication of three-server protocol (1.28 MB) as the communication incurred in two-server protocol is independent of dictionary size (n_0) unlike in the three-server protocol.

The majority of communication in the three-server protocol is in *server-to-server* interaction as is clearly seen in Table 1. C in three-server protocol is extremely light-weight as its CPU execution time is a mere 0.3 ms and incurs just 2.8 KB of communication. Thus, even if we were to run C on a machine with poor system configuration and poor network connectivity to servers (as is the case of common browser environments), the performance of the protocol will almost be unaffected. While C in the two-server protocol is also light-weight but it has 10× more computation time and it incurs 64× more communication than C in three-server protocol.

[5]Token list size of 73 is the median of top 67% of input token lists in dataset, sorted by size and is representational of typical token lists arising in practice.

5.3 End-to-end Secure Inference Performance

Table 2 and Table 3 illustrate the performance of end-to-end secure inference protocols in all the three settings for LR and 4NN model respectively. The runtime of one-server, two-server and three-server is 3,842 ms, 264 ms and 57 ms for LR model and 10,477 ms, 450 ms, 82 ms for 4NN model respectively. The cost of secure computation of ML Models is significant for one-server protocol as it requires interactive secure two-party computations for evaluation of linear as well as non-linear layers. This is unlike the two-server protocol that requires interaction only for the non-linear layers (see Section 4.2). Moreover, the cost of secure computation of ML Models in the three-server setting is even lower compared to that in the two-server due to the presence of the additional server ($\mathcal{S}_2$) and once again the need for interaction only for non-linear layers (see Section 4.3).

As C is involved in the *secure featurization phase* only in two-server and three-server protocols, the computation and performance of C is independent of the ML Model. This can be observed from performance numbers corresponding to C in Table 2 and Table 3. These solutions also provide the flexibility in updating ML Models when required without worrying about C's system configuration; an important practical consideration.

To summarize, if C's input privacy is critical, then one-server protocols can be used. However, the runtime of one-server protocol is $3.8 - 10.5$ s which is unreasonable for real-time systems such as secure phishing detection. The two-server protocol provides acceptable security-efficiency trade-off with runtimes of only $264 - 450$ ms and security guarantee in non-colluding servers setting. Nevertheless, for applications of secure phishing detection that require critical response time, an execution time > 100 ms is unreasonable. In these cases, our three-server secure featurization protocol can be used which has an execution time of only 82 ms even for complex models like 4NN and still provides acceptable security guarantees to the client.

REFERENCES

[1] Apple and Google. Exposure notification privacy-preserving analytics (enpa) white paper. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *CCS*, 2016.

[3] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. In *NDSS 2015*.

[4] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *EUROCRYPT*. Springer, 2015.

[5] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, 2016.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*. IEEE Computer Society, 2001.

[7] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In *NDSS*. The Internet Society, 2014.

[8] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning. In *IEEE EuroS&P 2019*.

[9] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPPRF. *PoPETs*, 2022(1), 2022.

[10] Edward J. Chou, Arun Gururajan, Kim Laine, Nitin Kumar Goel, Anna Bertiger, and Jack W. Stokes. Privacy-preserving phishing web page classification via fully homomorphic encryption. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, 2020.

[11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 2011.

[12] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282. USENIX Association, 2017.

[13] Sanchari Das, Andrew Kim, Zachary Tingle, and Christena Nippert-Eng. All about phishing: Exploring user research through a systematic literature review. *CoRR*, abs/1908.05897, 2019.

[14] Steven Englehardt. Next steps in privacy-preserving telemetry with prio. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/.

[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 or A Completeness Theorem for Protocols with Honest Majority. In *STOC 1987*.

[17] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[18] B. Issac, R. Chiong, and S. M. Jacob. Analysis of phishing attacks and countermeasures, 2014.

[19] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security 2018*.

[20] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In *FC*. Springer, 2014.

[21] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[22] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 2009.

[23] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*. Springer, 2005.

[24] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS*, 2017.

[25] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow Inference. In *IEEE S&P 2020*.

[26] Hung Le, Quang Pham, Doyen Sahoo, and Steven C. H. Hoi. Urlnet: Learning a URL representation with deep learning for malicious URL detection. *CoRR*, abs/1802.03162, 2018.

[27] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://eprint.iacr.org/2016/046.

[28] Pranav Maneriker, Jack W. Stokes, Edir Garcia Lazo, Diana Carutasu, Farid Tajaddodianfar, and Arun Gururajan. Urltran: Improving phishing URL detection using transformers. *CoRR*, abs/2106.05256, 2021.

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

[30] Payman Mohassel and Peter Rindal. ABY^3: A Mixed Protocol Framework for Machine Learning. In *CCS 2018*.

[31] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast database joins and PSI for secret shared data. In *CCS*, 2020.

[32] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *CCS*, 2015.

[33] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE S&P 2017*.

[34] mpc-msri. EzPC. https://github.com/mpc-msri/EzPC.

[35] mpc msri. 2pc-circuit-psi, 2021.

[36] Oleksandr-Tkachenko. HashingTables. https://github.com/Oleksandr-Tkachenko/HashingTables.

[37] osu-crypto. libOTe. https://github.com/osu-crypto/libOTe. Accessed: 2020-10-07.

[38] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*. Springer, 2001.

[39] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX*, 2015.

[40] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*. Springer, 2019.

[41] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT*. Springer, 2018.

[42] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 2018.

[43] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. SIRNN: A math library for secure inference of RNNs. In *IEEE S&P 2020*, 2020.

[44] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In *CCS*, 2020.

[45] 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 *AsiaCCS 2018*.

[46] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *CCS*, 2017.

[47] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *EUROCRYPT*, 2021.

[48] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *CCS*, 2019.

[49] Adi Shamir. How to share a secret. *Commun. ACM*, 1979.

[50] The OpenSSL Project. OpenSSL Cryptography and SSL/TLS Toolkit, https://www.openssl.org/. https://www.openssl.org/.

[51] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. *PoPETs 2019*.

[52] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS 1986*.

[53] Huaping Yuan, Zhenguo Yang, Xu Chen, Yukun Li, and Wenyin Liu. Url2vec: Url modeling with character embeddings for fast and accurate phishing website detection. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 2018.

[54] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *Int. J. Mach. Learn. Cybern.*, 2010.

A ADDITIONAL PRELIMINARIES

A.1 RCPA Security of Symmetric Key Encryption Scheme

A symmetric encryption scheme $\mathcal{E}$=(Gen, Enc, Dec) is said to be pseudorandom ciphertexts under chosen plaintext attack (RCPA) [7] secure if no PPT adversary $\mathcal{A}$ can win the $\text{RCPA}_{\mathcal{E},\mathcal{A}}$ game (described in Fig. 6) (i.e., $RCPA_{\mathcal{E},\mathcal{A}}(1^\lambda) = 1$), except with probability at most $\frac{1}{2} + \text{neg}(\lambda)$. The advantage of $\mathcal{A}$ in $\text{RCPA}_{\mathcal{E},\mathcal{A}}$, $\mathbf{Adv}^{\text{RCPA}}_{\mathcal{E},\mathcal{A}}(\lambda)$, is $2.\Pr\left[RCPA_{\mathcal{E},\mathcal{A}}(1^\lambda) = 1\right] - 1$. For a list X such that for all $i \in |X|$, $X[i] \in \mathcal{M}$, $Y \leftarrow \mathcal{E}.\mathsf{Enc}(k, X)$ denotes encryption is performed element-wise on list X and it outputs a list of ciphertexts Y such that $Y[i] = \mathcal{E}.\mathsf{Enc}(k, X[i])$ for all $i \in |X|$. We abuse the notation of $\mathcal{E}.\mathsf{Dec}(\cdot)$ analogously.

Parameters: Symmetric Key Encryption Scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.
The game proceeds as follows:

1. Choose a uniform bit b. If $b = 0$, $k \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$.
2. The adversary $\mathcal{A}$ is given an oracle access to $\mathsf{Enc}(\cdot)$. For a message $m \in \mathcal{M}$, if $b = 0$ then $\mathsf{Enc}(\cdot)$ oracle outputs $\mathcal{E}.\mathsf{Enc}(k, m)$ else it outputs $c \overset{\$}{\leftarrow} \mathcal{C}$.
3. The adversary $\mathcal{A}$ outputs bit b'.
4. The output of the game is defined to be 1 if $b' = b$, and 0 otherwise. In the former case, $\mathcal{A}$ is adjudged to win the game.

Fig. 6: $\text{RCPA}_{\mathcal{E},\mathcal{A}}$ game

A.2 Stash overflow in Cuckoo Hashing

Stash overflow occurs on insertion of $s + 1$ elements to a stash of size s and this event is termed as a hashing failure. The probability (over the sampling of hash functions) that a stash overflows is called the failure probability. It was shown in [22] that cuckoo hashing of n_h elements into $(1 + \varepsilon)n_h$ bins with $\varepsilon \in (0, 1)$ for $s \geqslant 0$ and $d \geqslant 2(1+\varepsilon)\ln(\frac{1}{\varepsilon})$ has failure probability $O(n_h^{1-c(s+1)})$, for some constant $c > 0$ as $n_h \mapsto \infty$. *Pinkas et al.* [42] empirically determined that in order to achieve a concrete failure probability of less than 2^{-40} for stash size $s = 0$ and d=3, 4 and 5, the number of bins (β) required are $1.27n_h$, $1.09n_h$ and $1.05n_h$ respectively. Empirical analysis to bound the concrete failure probability in the no stash setting has been widely considered in literature [9, 24, 31, 40, 42, 46, 48]. To bound the overall failure probability of our protocols to 2^{-40}, we require the failure probability of Cuckoo hashing in no stash setting to be at most 2^{-41}. Extrapolating, similar to [42], we get $\beta = 1.28n_h$ to ensure that the failure probability of Cuckoo hashing in no stash setting is at most 2^{-41}.

B CORRECTNESS AND SECURITY PROOF OF ONE-SERVER PROTOCOL

We now give a complete proof of Theorem 1, by proving the correctness and security of the protocol given in Fig. 3.

Proof. **Correctness.** Here, we prove that our protocol correctly realizes $\mathcal{F}_{\mathsf{SF}}$ functionality parameterized by function g of the underlying $\mathcal{F}_{\mathsf{CPSI}}$ functionality. We need to show that for all $j \in [cn_0]$, if there exists $i, p \in [n_1]$ s.t. $D[i] = T[p]$ and $f(i) = j$ then $\mathsf{SS}_A.\mathsf{Reconstruct}^A(U_0[j], U_1[j]) = E[p]$, where $f : [n_0] \to [cn_0]$ ($c \geqslant 1$) is an injective function output by g on input D and random tape $R_{\mathcal{S}}$, else $\mathsf{SS}_A.\mathsf{Reconstruct}^A(U_0[j], U_1[j]) = 0$. This can be shown as follows. Firstly, from the correctness of $\mathcal{F}_{\mathsf{CPSI}}$ functionality, we have shares of intersection $\mathcal{I}_0, \mathcal{I}_1$ and Q_0, Q_1 (step 1) such that for all $j \in [cn_0]$, if $\exists i, p$ s.t. $f(i) = j$ and $D[i] = T[p]$,

$$\mathsf{SS}_B.\mathsf{Reconstruct}^B(\mathcal{I}_0[j], \mathcal{I}_1[j]) = 1 \;\&$$
$$\mathsf{SS}_A.\mathsf{Reconstruct}^A(Q_0[j], Q_1[j]) = E[p],$$

else,

$$\mathsf{SS}_B.\mathsf{Reconstruct}^B(\mathcal{I}_0[j], \mathcal{I}_1[j]) = 0 \;\&$$
$$\mathsf{SS}_A.\mathsf{Reconstruct}^A(Q_0[j], Q_1[j]) = r_j,$$

where r_j is a random value in $\mathbb{Z}_L$. Let i_j denote the boolean value whose secret-shares are $\mathcal{I}_0[j]$ and $\mathcal{I}_1[j]$ and let q_j denote the value whose secret shares are $Q_0[j]$ and $Q_1[j]$. From the correctness of $\mathcal{F}_{\mathsf{MUX}}$ (step 2), we have that $U_0[j]$ and $U_1[j]$ are secret-shares of q_j, if $i_j = 1$, else are secret-shares of 0. Thus, $\mathsf{SS}_A.\mathsf{Reconstruct}^A(U_0[j], U_1[j]) = \mathsf{E}[p]$, if $\exists i, p$ s.t. $f(i) = j$ and $D[i] = T[p]$, else $\mathsf{SS}_A.\mathsf{Reconstruct}^A(U_0[j], U_1[j]) = 0$, which concludes our proof.
Security. The security of the protocol follows immediately from security of $\mathcal{F}_{\mathsf{CPSI}}$ and $\mathcal{F}_{\mathsf{MUX}}$ functionalities. □

C CORRECTNESS AND SECURITY PROOF OF TWO-SERVER PROTOCOL

We now give a complete proof of Theorem 2, by proving the correctness and security of the protocol given in Fig. 4.

Proof. **Correctness.** We need to prove that our protocol correctly outputs the output of $\mathcal{F}_{\mathsf{SF}}$ functionality parameterized with a constant function g that outputs identity map $f : [n_0] \to [n_0]$. Stated differently, we need to show that $\mathsf{FS}_0[t] + \mathsf{FS}_1[t] = \mathsf{E}[p]$, if $D[t] \in T$ ($D[t] = T[p]$), $\mathsf{FS}_0[t] + \mathsf{FS}_1[t] = 0$ otherwise. In steps 1 and 2, the collisions in hash digests of two distinct element pairs of arbitrary length in D and T happens with $\frac{n_0 \cdot n_1}{2^\mu}$ probability. Hence, setting $\mu = \sigma + \log(n_0) + \log(n_1) + 1$ ensures that the probabilty of collision is $2^{-(\sigma+1)}$. Moreover, we set the parameter ε in cuckoo hashing scheme s.t. the probability of hashing failure in no stash setting is $2^{-(\sigma+1)}$, where $\sigma = 40$ as discussed in Section 2.2. Hence, the overall failure probability is at most 2^{-40}.
Case 1 ($D[t] \in T$). Let $p \in [n_1]$ be an index s.t. $D[t] = T[p]$. From the correctness of hashing and use of the same hash functions by the client and the servers in steps 3 and 18, it holds that for $t \in [n_0]$, if $D[t] \in T$ then there exists a unique $j \in \mathsf{PSet} = \{h_1(A[t]), h_2(A[t]), h_3(A[t])\}$ s.t. $\mathsf{HT}_C[j] = A[t]$. Hence, $K_0[j]$ and $K_1[j]$ are the DPF keys generated for point function $f_{A[t],\mathsf{E}[p]}$. For pos $= j$ in step 20, $\mathcal{S}_b$ updates $\mathsf{FS}_b[t]$ by adding

$F.\mathsf{Eval}(b, K_b[j], A[t])$ to the current value. From the correctness of DPF scheme F it holds that $\sum_{b=0}^{1} F.\mathsf{Eval}(b, K_b[j], A[t]) = \mathsf{E}[p]$. And for $(j' \in \mathsf{PSet}) \neq j$, the keys $K_0[j']$ and $K_1[j']$ are DPF keys generated for point function $f_{a_{j'}, b_{j'}}$, where $a_{j'} \neq A[t]$ or $a_{j'}$ is dummy element and $b_{j'} = 0$. In this case too, $\mathcal{S}_b$ updates $\mathsf{FS}_b[t]$ by adding $F.\mathsf{Eval}(b, K_b[j'], A[t])$ to the current value. From the correctness of DPF scheme F it holds that $\sum_{b=0}^{1} F.\mathsf{Eval}(b, K_b[j'], A[t]) = 0$. Hence, $\mathsf{FS}_0[t]$ and $\mathsf{FS}_1[t]$ are shares of $\mathsf{E}[p]$.
Case 2 ($D[t] \notin T$). As $\mathsf{HT}_C[j] \neq A[t]$ for all $j \in \beta$, it holds that all pairs of DPF keys $K_0[j]$ and $K_1[j]$ correspond to point function f_{a_j, b_j}, where $a_j \neq A[t]$ or $a_{j'}$ is dummy element and $b_{j'} = 0$. For all $\mathsf{pos} \in \mathsf{PSet}$, $\mathcal{S}_b$ updates $\mathsf{FS}_b[t]$ by adding $F.\mathsf{Eval}(b, K_b[\mathsf{pos}], A[t])$ to the current value. And from the correctness of DPF scheme F it holds that $\sum_{b=0}^{1} F.\mathsf{Eval}(b, K_b[\mathsf{pos}], A[t]) = 0$. Hence, $\mathsf{FS}_0[t]$ and $\mathsf{FS}_1[t]$ are shares of 0.

Security.
Case 1 (C is corrupt). The entire computation of C is local. Hence all the steps in the protocol computed at client's end can be executed by the simulator using the inputs of corrupted client.
Case 2 ($\mathcal{S}_0/\mathcal{S}_1$ is corrupt). Observe that, except for step 14 in the protocol all the computations of $\mathcal{S}_b$ is done locally. Simulator for corrupted server $\mathcal{S}_b$ can be constructed as follows: 1) Computations in steps 1 can be executed by the simulator using input of corrupted $\mathcal{S}_b$. 2) View of corrupted $\mathcal{S}_b$ can be simulated by invoking $\mathsf{Sim}(\cdot)$ of the DPF scheme F, β-many times. $\mathsf{Sim}(\cdot)$ is invoked with μ and description of $\mathbb{Z}_L$, i.e., the description of input domain and output group of the underlying point function family $\mathcal{F}$. 3) The rest of the computation of $\mathcal{S}_b$ can be simulated by the simulator in a straight-forward fashion as it is all local. □

D CORRECTNESS AND SECURITY PROOF OF THREE-SERVER PROTOCOL

We now give a complete proof of Theorem 3, by proving the correctness and security of the protocol given in Fig. 5.

Proof. **Correctness.** We need to prove that our protocol correctly outputs the output of $\mathcal{F}_{\mathsf{SF}}$ functionality parameterized with a constant function g that outputs identity map $f : [n_0] \rightarrow [n_0]$. Stated differently, we need to show that $\mathbf{F}'_0[t] + \mathbf{F}'_1[t] = \mathsf{E}[p]$, if $D[t] \in T$ $(D[i] = T[p])$, $\mathbf{F}'_0[t] + \mathbf{F}'_1[t] = 0$ otherwise. Based on the analysis in proof of Theorem 2, we set $\mu = \sigma + \log(n_0) + \log(n_1)$ to ensure that the probabilty of collision is $2^{-\sigma}$ in steps 1 and 2. In the rest of the proof, we will assume that there are no collisions in hash digests of distinct elements.
Case 1 ($D[t] \in T$). Let $p \in [n_1]$ be an index s.t. $D[t] = T[p]$. For $i = \pi_A(t)$ and $j = \pi_B(p)$, $A'[i] = G(k_G, A[t])$ and $B'[j] = G(k_G, B[p])$. $\mathsf{E}_0[p]$ and $\mathsf{E}_1[p]$ are secret shares of $\mathsf{E}[p]$. Thus, $EF_0[j]$ is encryption of share $\mathsf{E}_0[p]$ and $F_1[j] = \mathsf{E}_1[p]$. If $D[t] = T[p]$, then $A[t] = B[p]$ and $A'[i] = B'[j]$. Thus, in step 11, $\mathcal{S}_2$ sets $\mathbf{F}_1[i] = F_1[j]$. Similarly, in step 20, $\mathcal{S}_2$ sets $\mathbf{EF}_0[i] = EF_0[j]$. This implies that, $\mathbf{F}_b[i] = \mathsf{E}_b[p]$, for $b \in \{0, 1\}$. Hence, $\mathbf{F}'_b[t] = \mathsf{E}_b[p]$, for $b \in \{0, 1\}$.
Case 2 ($D[t] \notin T$). Let $i = \pi_A(t)$, then $A'[i] = G(k_G, A[t])$. As $D[t] \notin T$, $A[t] \notin T$ and $A'[i] \notin B'$. Thus $\mathcal{I}[i] = 0$. In step 13, $\mathcal{S}_2$ sets $\mathbf{F}_1[i]$ with a uniformly random value in $\mathbb{Z}_L$. Observe that in step 16, $\mathcal{S}_1$ sets $\mathbf{EF}_1[i] = \mathcal{E}.\mathsf{Enc}(k_{\mathcal{E}}, -\mathbf{F}_1[i])$. This implies that, $\mathbf{EF}_0[i] = \mathcal{E}.\mathsf{Enc}(k_{\mathcal{E}}, -\mathbf{F}_1[i])$. Hence, $\mathbf{F}'_0[t]$ and $\mathbf{F}'_1[t]$ are shares of 0.

Security.
Case 1 (C is corrupt). Simulating the view in case of a corrupted C is straight-forward as its entire computation in the protocol is local.
Case 2 ($\mathcal{S}_0$ is corrupt). Once steps 3 and 17 are simulated correctly, all the steps in the protocol that are computed at $\mathcal{S}_0$'s end can be simulated using the inputs of corrupted $\mathcal{S}_0$ as the rest of the computation at $\mathcal{S}_0$'s end is local. This can be done by sampling keys $k_G \xleftarrow{\$} \{0,1\}^\lambda$ and $k_{\mathcal{E}} \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$ (step 3) and by sending an encrypted list (of size n_0) of elements sampled uniformly at random from $\mathbb{Z}_L$ in step 17. This is indistinguishable from the view of $\mathcal{S}_0$ in real execution because of the security of secret-sharing scheme.
Case 3 ($\mathcal{S}_1$ is corrupt). The view of $\mathcal{S}_1$ can be simulated in a similar way as the case when $\mathcal{S}_0$ is corrupt.
Case 4 ($\mathcal{S}_2$ is corrupt). In the protocol, $\mathcal{S}_2$ receives A' and B' from $\mathcal{S}_0$ and C which can be simulated by creating lists of appropriate lengths that comprise of elements sampled uniformly at random from $\{0,1\}^\mu$ with the constraint that any q elements in A' are equal to any q elements in B'. The simulation of these steps (4 and 5) is indistinguishable from view of $\mathcal{S}_2$ in real interaction as G is a PRP and the lists A and B are permuted by random permutations. Step 6 can be simulated by sending two lists of size n_1 to $\mathcal{S}_2$; in the first list the elements are sampled uniformly at random from $\mathbb{Z}_L$ and in the second list the elements are sampled uniformly at random from the ciphertext space C. The simulation is indistinguishable from the view of $\mathcal{S}_2$ in real interaction due to the security of secret sharing scheme and RCPA security of SKE scheme. Finally, step 16 can be simulated by sending a list of size n_0 comprising ciphertexts sampled uniformly at random from C. □