Toward Verifiable and Privacy Preserving Machine Learning Prediction

Chaoyue Niu, *Student Member, IEEE,* Fan Wu, *Member, IEEE,* Shaojie Tang, *Member, IEEE,* Shuai Ma, *Senior Member, IEEE,* and Guihai Chen, *Senior Member, IEEE*

Abstract—The ubiquitous needs for extracting insights from data are driving the emergence of service providers to offer predictions given the inputs from customers. During this process, it is important and highly nontrivial for the service providers to generate proofs of honest predictions without leaking the key parameters of their trained models. In addition, the customers are usually unwilling to reveal their sensitive inputs. In this paper, we proposed MVP, which enables Machine learning prediction in a Verifiable and Privacy preserving fashion. MVP features the properties of polynomial decomposition and prime-order bilinear groups to simultaneously facilitate oblivious evaluation and batch outcome verification while maintaining function privacy and input privacy. We further instantiated MVP with Support Vector Machines (SVMs) and extensively evaluated its performance for the spam detection task on three practical Short Message Service (SMS) datasets. Our analysis and evaluation results reveal that MVP achieves the desired properties while incurring low computation and communication overhead.

Index Terms—Machine Learning Prediction, Verifiability, Function Privacy, Input Privacy

1 Introduction

With the proliferation of artificial intelligence (AI), many companies, institutions, and cloud platforms (e.g., Amazon Web Services, Google Cloud, Microsoft Azure, and Alibaba Cloud) have offered common machine learning (ML) prediction services to customers on a pay-per-prediction basis, such as natural language understanding, image classification, video annotation, and anomaly detection. In addition, the cloud platforms have also rolled out Machine Learning as a Service (MLaaS), enabling expert or even non-expert users to build customized models over their private data and further to provide a wider range of prediction services. However, there exists a critical security problem in the ecosystem, i.e., it is difficult to verify the correctness of predictions without compromising function privacy and input privacy. Let's examine a classification service as follows. Simultaneously ensuring outcome verifiability and privacy preservation requires the service provider not only to perform classifications without knowing the sensitive test data from customers but also to generate proofs of correct computations while keeping the key parameters of the classifier in secret.

Ensuring verifiability and preserving privacy are both important to the long term healthy development of AI services. On the one hand, a service provider undertakes the end-to-end cycle of ML development: collecting and preprocessing data, training model, and supporting concurrent prediction queries. Thus, the service provider has a heavy workload and might return incorrect answers in face of software/hardware failures, human errors, and external malicious attacks [1]. In addition, to reduce operation cost, a strategic service provider may have strong incentives to return incorrect results, if such results consume less work and are unlikely to be detected by the customers [2]–[4]. In the case of the classification example raised above, an opportunistic way is to return random class labels without processing the test data. However, if such speculative and illegal behaviors cannot be identified and prohibited, it may cause heavy losses to customers, especially in integrity-sensitive applications, such as risk assessments and investment suggestions in financial markets [5], disease diagnosis and drug dosing control in medical services [6], face and object recognition in home monitoring [7], etc. Therefore, the property of *outcome verifiability* is necessary for accountable, transparent, and robust prediction services. On the other hand, securing confidential information and protecting the personal data from customers are important objectives for every enterprise. First, building a high-quality trained model relies not only on large volumes of high-quality training data but also on a well-balanced exploration and exploitation of ML algorithms. Thus, the fine-tuned trained model incurs significant data collection and data processing overhead and has a great commercial value to the service provider [7]–[10]. This indicates that the key model parameters (e.g., support vectors in Support Vector Machines (SVMs) and weights and biases in neural networks) should be hidden from greedy customers and external attackers, i.e., *function privacy*[1]. Second, when

- C. Niu, F. Wu, and G. Chen are with the Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: rvince@sjtu.edu.cn; {fwu, gchen}@cs.sjtu.edu.cn.
- S. Tang is with the Naveen Jindal School of Management, University of Texas at Dallas, Richardson, TX 75080. E-mail: shaojie.tang@utdallas.edu.
- S. Ma is with Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing 100083, China. E-mail: mashuai@buaa.edu.cn.
- F. Wu is the corresponding author.

[1] Function privacy denotes protecting the parameters of a function in general computation. In the special context of ML prediction, it refers to "model privacy" for protecting the parameters of a trained model.

handling private inputs (e.g., personal communications, user profiles, commercial transactions, and health records) from customers, it is the bottom line of every business to respect their privacy. According to the survey report of 2016 TRUSTe/NCSA Consumer Privacy Infographic - US Edition, 89% of audiences say they avoid companies that do not protect their privacy [11]. Furthermore, the successive Facebook data scandals highlight that the sensitive test data from customers should not be disclosed to the service provider to guarantee *input privacy*.

To integrate outcome verifiability, function privacy, and input privacy in practical ML prediction services, there are three major challenges. The first and the thorniest challenge is that verifying the correctness of returned predictions and preserving the function privacy seem to be contradictory objectives. Ensuring outcome verifiability allows customers to know the key parameters of the trained model, whereas protecting function privacy tends to prevent them from learning these confidential contents. In particular, verifiable computation in outsourced scenarios enables the service provider to generate a proof vouching for the correctness of the returned result [12], [13]. However, the customer as a verifier, who outsources the computation, should know the concrete function before doing verification.

The second challenge comes from input privacy, which makes guaranteeing outcome verifiability even harder. To get a tradeoff between privacy and functionality, homomorphic encryption can be exploited to enable practical computation over encrypted test data [7], [9], [10], [14]–[16]. However, a hidden problem is that the proof of correct computation normally involves the test data, while the service provider, only knowing ciphertexts, may fail to generate such a proof. In addition, it is highly nontrivial to design an efficient verification scheme on top of existing homomorphic encryption protocols, let alone incorporating the function privacy. Moreover, there still exist some work parallel to this work (e.g., differentially private ML training [17]–[22]), which targets on protecting the sensitive training set in the training phase, rather than the test set from the customer in the prediction phase.

The ultimate challenge is verifying the integrity of massive underlying operations on the test data. For example, SVMs with the polynomial kernel (resp., the RBF kernel) require to compute dot product (resp., squared Euclidean distance) between the test data and each support vector. In addition, according to our statistics over practical SMS datasets, the maximum percentage of the support vectors, accounting for the training set, could reach 20.98%. If the customer simply performs outcome verification in sequence, it would be prohibitively slow, especially when feeding in a large-scale test set. Meanwhile, transmitting a mass of proofs also incurs serious communication overhead. Under such circumstances, verifying outcomes sequentially certainly becomes the processing bottleneck at the customer.

We summarize the key contributions of this work:

- To the best of our knowledge, the proposed secure scheme MVP is, for the first time, to achieve outcome verifiability, function privacy, and input privacy simultaneously in ML prediction services. The design of MVP uses some cryptographic primitives that will be reviewed in Section 2.2, but does not rely on any trusted hardware.

TABLE 1
Frequently Used Notations.

Notation	Remark
f	A general function or a hypothesis in ML prediction
K, ψ	The kernel function, its underlying polynomial function
i, j, k	The indices of feature, support vector, test data
n	The default dimension of a vector
$\mathbf{x}, \mathbf{x}_j$	The coefficient vector of f, a support vector in SVMs
$\mathcal{SV}$	The indices of the support vectors in the training set
$\mathbf{z}, \mathbf{z}_k, \phi$	An input of f (i.e., a data vector) in general computation, a test point in ML prediction, the size of test set
$\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$	A bilinear map over three multiplicative cyclic groups
q	The prime group order of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$
g, h	A generator of $\mathbb{G}_1$, a generator of $\mathbb{G}_2$
$\mathbb{Z}_q^* = \{1, \ldots, q-1\}$	The multiplicative group of integers modulo q
$G = \mathbb{G}_1^2$	The product group over $\mathbb{G}_1$
$(g_1, g_2) \in G$	An element randomly chosen from G
$s \in \mathbb{Z}_q^*$	The private key of the adapted BGN scheme
$G_1 \subset G$	A linear subgroup of G
(g, g^s)	The generator of G_1
$C \in G$	A BGN-type ciphertext
π	The projection function for BGN decryption
PK, SK	A pair of system public and secret keys
$\mathbf{t}$	A commitment point
FK(f)	A function public key of f
σ	A signature of correct computation
$w_i(\mathbf{t})$	The decomposed polynomial used to generate σ
V, v	The encrypted result, the (decrypted) result
$\mathcal{D}, \mathcal{A}, \mathcal{S}, \mathcal{C}$	A distinguisher, an adversary, a simulator, a challenger
$x_i, z_i, t_i, x_i^{(j)}, z_i^{(k)}$	The i-th elements of $\mathbf{x}, \mathbf{z}, \mathbf{t}, \mathbf{x}_j, \mathbf{z}_k$

- MVP first features the properties of polynomial decomposition and prime-order bilinear groups to allow outcome verification while keeping the function parameters secret. MVP then integrates privacy preservation with polynomial evaluation and outcome verification via an adapted BGN homomorphic cryptosystem over the prime-order bilinear groups. On the one hand, the homomorphic properties enable the service provider to obliviously and efficiently compute two multivariate polynomials (i.e., dot product and squared Euclidean distance) underlying common ML algorithms. On the other hand, the BGN-type ciphertext also can be conveniently embedded into or stripped from the signature of correct computation, reconciling the contradiction between outcome verifiability and input privacy. To further support a large scale of test data, MVP incorporates batch verification and signature aggregation through the bilinearity of asymmetric pairing, dramatically reducing computation and communication overhead. Please refer to Section 4 for the design overview and design principles of MVP in detail.
- We instructively instantiate MVP with SVMs and evaluate on three SMS datasets. When MVP supports 1000 test data with 1000 features, the major evaluation results are: (1) the maximum total overhead of oblivious evaluation per test data is 0.95 s under the sparse encryption strategy; and (2) the total outcome verification overhead of MVP is 9.86% and 1.23% (particularly, 9.47 ms and 11.66 ms per test data) of that of the naïve method in the polynomial kernel and the RBF kernel, respectively, under the sparse encryption strategy. These two ratios decrease to 0.83% and 0.23% under the dense encryption strategy; and (3) the communication overhead of the customer is 0.241 MB and 0.483 MB in the polynomial kernel and the RBF kernel, respectively.

2 Preliminaries

In this section, we introduce technical preliminaries. We first introduce a classical supervised learning algorithm, called support vector machines (SVMs) [23]. We then introduce some cryptographic primitives. For clarify, we list the frequently used notations in Table 1

2.1 Support Vector Machines

Given a training set of m data points $\{\mathbf{x}_l, y_l\}_{l=1}^{m}$ with n-dimensional feature vector $\mathbf{x}_l = (x_1^{(l)}, \dots, x_n^{(l)})^T \in \mathbb{R}^n$ and two-class label $y_l \in \{-1, 1\}$, SVMs search for hyperplanes that separate the training data by a maximum margin. The training instances that lie closest to the hyperplanes are called *support vectors*. In the case when the original training data are not linearly separable, SVMs allow to implicitly project them to a higher dimensional feature space via a Mercer kernel operator $K(\cdot, \cdot)$. We thus express the optimal margin classifier in the general form:

$$f\left(\mathbf{z}_k\right) = \sum_{l=1}^{m} y_l \alpha_l K\left(\mathbf{x}_l, \mathbf{z}_k\right) + b = \sum_{j \in \mathcal{SV}} y_j \alpha_j K\left(\mathbf{x}_j, \mathbf{z}_k\right) + b,$$

where $\mathbf{z}_k$ is a new test data from the test set $\{\mathbf{z}_k\}_{k=1}^{\phi}$, α_l's are the Lagrange multipliers, b is the intercept term, and $\mathcal{SV}$ denotes the index set of the support vectors. In addition, α_l's are nonzero only for the support vectors, i.e., $\forall j \in \mathcal{SV}, \alpha_j > 0$. Moreover, we predict $\mathbf{z}_k$ as 1 if $f(\mathbf{z}_k) \geq 0$ and -1 otherwise.

2.2 Cryptographic Primitives

We introduce the cryptographic primitives underlying our design. We first show the definition of prime-order bilinear groups, then review some relevant hardness assumptions, and finally present an adapted Boneh-Goh-Nissim (BGN) homomorphic cryptosystem.

Definition 1 (Prime-Order Bilinear Groups)**.** The three multiplicative cyclic groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ are of the same prime order q. Let g be a generator of $\mathbb{G}_1$ and h be a generator of $\mathbb{G}_2$. An asymmetric bilinear pairing/map is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following three properties:

- *Bilinearity:* $\forall X, Y \in \mathbb{G}_1, \forall Z \in \mathbb{G}_2, \forall a, b \in \mathbb{Z}_q^*$,
$$\hat{e}\left(X^a, Z^b\right) = \hat{e}\left(X, Z\right)^{ab},$$
$$\hat{e}\left(X, Z\right) \hat{e}\left(Y, Z\right) = \hat{e}\left(XY, Z\right).$$
- *Non-degeneracy:* $\hat{e}\left(g, h\right) \neq 1_{\mathbb{G}_T}$.
- *Computability:* Given $X \in \mathbb{G}_1, Z \in \mathbb{G}_2$, there exists an efficient algorithm to compute $\hat{e}\left(X, Z\right)$.

We call $\mathbb{G}_1, \mathbb{G}_2$ bilinear groups if there exists such a group $\mathbb{G}_T$ and an asymmetric pairing $\hat{e}$ as above.

Definition 2 (Discrete Logarithm Problem (DLP) [24])**.** Given $X, Y \in \mathbb{G}_1$, it is computationally intractable to find an integer a such that $Y = X^a$.

Definition 3 (Symmetric External Diffie-Hellman (SXDH) Assumption [25])**.** Given $g, g^a, g^b, g^c \in \mathbb{G}_1$ for unknown $a, b, c \in \mathbb{Z}_q^*$, it is computationally hard to determine whether $c \equiv ab \mod q$. Equivalently, it is computationally infeasible to determine whether (g^b, g^c) is in the cyclic subgroup of $\mathbb{G}_1^2$ generated by (g, g^a).

Definition 4 (ℓ-Strong Diffie-Hellman (ℓ-SDH) Assumption [26])**.** Given a $(\ell + 3)$-tuple

$$\left(g, g^{\lambda}, \dots, g^{\lambda^{\ell}}, h, h^{\lambda}\right) \in \mathbb{G}_1^{\ell+1} \times \mathbb{G}_2^2$$

for $\lambda \in_R \mathbb{Z}_q^*$, there is no probabilistic polynomial-time (PPT) algorithm that can output the pair $(c, \hat{e}(g, h)^{1/(\lambda+c)}) \in \mathbb{Z}_q^* \backslash \{-\lambda\} \times \mathbb{G}_T$, except with negligible probability.

Based on the prime-order bilinear groups and the SXDH assumption, Freeman [25] designed a stripped-down version of the somewhat homomorphic encryption scheme initially proposed by Boneh, Goh, and Nissim [27]. Without a limit as in the original BGN scheme that the composite group order must be infeasible to factor, the bilinear groups in this adapted version have a much smaller prime order. Thus, at the same security level, it allows faster group/pairing operations and shorter ciphertext length. Furthermore, efficiency improvement is more remarkable at higher security levels. We describe three major algorithms of the adapted BGN scheme as follows.

KeyGen (τ) : Given a security parameter $\tau \in \mathbb{Z}^+$, generate bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order $q > 2^\tau$ as well as an asymmetric pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. In addition, g is a generator of $\mathbb{G}_1$, and h is a generator of $\mathbb{G}_2$. Then, define the product group $G = \mathbb{G}_1^2$ and its random linear subgroup $G_1 \subset G$ generated by (g, g^s) for some random $s \in \mathbb{Z}_q^*$. Moreover, randomly choose an element (g_1, g_2) from G. The public key is $\mathcal{PK} = \{q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, h, (g, g^s), (g_1, g_2)\}$. The private key is $\mathcal{SK} = s$.

Encrypt $(\mathcal{PK}, m)$: To encrypt a message z with the public key $\mathcal{PK}$, pick a random integer $r \in \mathbb{Z}_q^*$ and then compute the ciphertext as

$$C = \left(g_1, g_2\right)^z \left(g, g^s\right)^r = \left({g_1}^z g^r, {g_2}^z g^{sr}\right) \in G.$$

Note that the message has been encoded by the element (g_1, g_2) from G and further blinded with a random element from G_1, namely (g^r, g^{sr}).

Decrypt $(C, \mathcal{SK})$: Decryption in G is essentially achieved by "projecting" the ciphertext away from the blinding term and taking a logarithm to recover z. Here the projection function π on $(g_1, g_2) \in G$ is defined as

$$\pi\left(\left(g_1, g_2\right)\right) = \left(g_1\right)^s \left(g_2\right)^{-1} = {g_1}^s {g_2}^{-1},$$

where s is the private key. Now, do projection on the ciphertext C and derive

$$\begin{aligned}\pi\left(C\right) &= \pi\left(\left({g_1}^z g^r, {g_2}^z g^{sr}\right)\right) = \left({g_1}^z g^r\right)^s \left({g_2}^z g^{sr}\right)^{-1} \\ &= \left({g_1}^s {g_2}^{-1}\right)^z = \pi\left(\left(g_1, g_2\right)\right)^z.\end{aligned}$$

To recover z, it suffices to take the logarithm of $\pi\left(C\right)$ to the base $\pi\left(\left(g_1, g_2\right)\right)$. In addition, to make decryption occur in constant time, we can precompute a polynomial-size table of powers of $\pi\left(g_1, g_2\right)$. Moreover, without knowing the private key s, it is infeasible for any adversary to construct the projection function, thus performing decryption.

We further present some useful and efficient homomorphic properties of the adapted BGN cryptosystem, including multiplication by a constant, homomorphic addition, and addition by a constant. In particular, given two encryptions

$$C_1 = ({g_1}^{z_1} g^{r_1}, {g_2}^{z_1} g^{sr_1}) \in G,$$
$$C_2 = ({g_1}^{z_2} g^{r_2}, {g_2}^{z_2} g^{sr_2}) \in G,$$

of messages z_1, z_2 as well as a constant integer d, we have

- *Multiplication by a constant:*

$$\begin{aligned}(C_1)^d &= ({g_1}^{z_1} g^{r_1}, {g_2}^{z_1} g^{sr_1})^d \\ &= \left({g_1}^{dz_1} g^{dr_1}, {g_2}^{dz_1} g^{sdr_1}\right),\end{aligned}$$

  which is an encryption of dz_1;
- *Homomorphic addition:*

$$\begin{aligned}C_1 \cdot C_2 &= ({g_1}^{z_1} g^{r_1}, {g_2}^{z_1} g^{sr_1})\,({g_1}^{z_2} g^{r_2}, {g_2}^{z_2} g^{sr_2}) \\ &= ({g_1}^{z_1} g^{r_1} {g_1}^{z_2} g^{r_2}, {g_2}^{z_1} g^{sr_1} {g_2}^{z_2} g^{sr_2}) \\ &= \left({g_1}^{z_1+z_2} g^{r_1+r_2}, {g_2}^{z_1+z_2} g^{s(r_1+r_2)}\right),\end{aligned}$$

  which an encryption of $z_1 + z_2$;
- *Addition by a constant:*

$$\begin{aligned}C_1 \cdot (g_1, g_2)^d &= ({g_1}^{z_1} g^{r_1}, {g_2}^{z_1} g^{sr_1})\left({g_1}^d, {g_2}^d\right) \\ &= \left({g_1}^{z_1+d} g^{r_1}, {g_2}^{z_1+d} g^{sr_1}\right),\end{aligned}$$

  which is an encryption of $z_1 + d$.

3 PROBLEM FORMULATION

In this section, we present system and security models.

3.1 System Model

We first introduce a general system model for ML prediction services. A service provider first collects a large amount of raw data from diverse sources as a training set. Then, it applies a certain ML algorithm (e.g., linear regression, SVMs, k-means clustering, or neural networks) to the training set, thereby learning a trained model. Based on the trained model, the service provider takes new inputs from a customer as a test set and derive prediction results, which are finally returned to the customer. For tracking and authentication purposes, the service provider needs to register its trained models to a model manager. The model manager book-keeps historic and current versions of the trained models from different service providers. Also, the model manager needs to initialize some system parameters and publish on a certificated bulletin board[2] if necessary. To avoid being a single point of failure or bottleneck, redundant model managers with identical functionalities and databases should be installed.

We next introduce the syntax (intuitively, the workflow) of the building block of our secure scheme, which is specific to the polynomial function underlying ML algorithms.

Definition 5. A verifiable and privacy preserving scheme, for a polynomial function f with the coefficient vector $\mathbf{x}$ from a service provider and a data vector $\mathbf{z}$ from a customer, consists of 6 PPT algorithms defined as follows:

- $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\tau, f)$: The algorithm KeyGen takes a security parameter τ and the function f as inputs and outputs a pair of system public and secret keys (PK, SK). This algorithm is run only once at system initialization by the model manager.
- $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{PK}, f)$: The algorithm Setup, run by the model manager, takes the public key PK and the function f as inputs and generates a function public key $\mathsf{FK}(f)$.
- $\mathbf{C} \leftarrow \mathsf{Encrypt}(\mathsf{PK}, \mathbf{z})$: The algorithm $\mathsf{Encrypt}$, run by the customer, takes the public key PK and the data vector $\mathbf{z}$ as inputs and outputs a ciphertext set $\mathbf{C}$.
- $(V, \sigma) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{C})$: The algorithm $\mathsf{Compute}$, run by the service provider, takes the public key PK, the function f, and the ciphertext set $\mathbf{C}$, as inputs. It outputs the encrypted result V (which should be an encryption of $f(\mathbf{z})$ if computed correctly) and a proof of correct computation σ.
- $v \leftarrow \mathsf{Decrypt}(\mathsf{SK}, V)$: The algorithm $\mathsf{Decrypt}$, run by the model manager, takes the secret key SK and the encrypted result V as inputs and returns the decrypted result v to the customer.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{z}, v, \sigma)$: The algorithm Verify, run by the customer, takes the public key PK, the function public key $\mathsf{FK}(f)$, the data vector $\mathbf{z}$, the claimed result v, and the proof of correct computation σ, as inputs. If the result is accepted, the algorithm outputs 1; otherwise, it outputs 0.

3.2 Security Model

We identify attacks in ML prediction services and define corresponding security requirements. We further define the adversary model using the simulation-based technique.

3.2.1 Security Requirements

First, a service provider needs to support large-scale data contributions and perform data preprocessing. In addition, it is required to carefully train model and concurrently handle multiple queries from customers. Thus, the heavy-laden service provider might return incorrect answers. However, this case would cause heavy losses to the customer, especially in integrity-sensitive scenarios (e.g., financial, medical, and safeguarding applications). Moreover, the service provider may have strong motivations to return incorrect answers, if such answers require less work and are unlikely to be detected by the customer. Hence, the customer should have the capability to verify the integrity of returned answers, i.e., *outcome verifiability*. By following the principles of conventional verifiable computation, we state the outcome verifiability in our new context.

Definition 6 (Outcome Verifiability (Informal))**.** A secure outcome verification scheme in ML prediction services should satisfy:

- *Correctness*: Whenever the service provider has executed the prediction algorithm honestly, the customer, as a verifier, never rejects a correct proof.
- *Unforgeability*: The service provider, as an adversary, cannot forge valid proofs to convince the customer to accept

[2] A bulletin board is essentially a public broadcast channel with memory and is widely used in previous work/systems [28]–[30]. Park et al. from MIT also presented a blockchain-based design of a decentralized bulletin board in Section 3 of [31].

a wrong answer on an input of the customer's choice, except with negligible probability.

Second, a fine-tuned trained model has great commercial value to the service provider and should be paid for usage. However, some greedy customers and external attackers (e.g., business competitors) may try to obtain the trained model so that they can use it for free or to make profits. Therefore, the key parameters of the trained model (e.g., support vectors in SVMs and weight matrix and bias vector in neural networks) should be kept secret from the customer and external attackers, i.e., *function privacy*.

Third, we consider that the service provider attempts to glean sensitive information from the customer's private test data, such as personal communications, face images, daily activities, and health records. We here use the term *input privacy* to represent the security requirement of hiding the customer's test data from the service provider.

Last, just as [32], [33], we initially assume that the model manager can be trusted. For example, the role is played by the commercial companies with high reputations or the organizations that are strictly supervised with great transparency. In particular, Microsoft Azure [34], Google AI Platform [35], and Amazon Machine Learning [36] have already launched the model management service. Nevertheless, our design should support distributing the trust on the model manager. We will revisit the trust assumption in Remark 1.

3.2.2 Adversary Model

Guided by the practical security requirements above, we formally define the adversary model for our secure building block under Definition 5. Before that, we introduce a common concept in simulation-based definitions/proofs from [37], called *computational indistinguishability*.

Definition 7 (Computational Indistinguishability)**.** Two probability ensembles X and Y are two infinite sequences of random variables. If for every nonuniform polynomial-time algorithm $\mathcal{D}$ (intuitively, distinguisher), and for any element x uniformly sampled either from X or from Y, there exists a negligible probability such that $\mathcal{D}$ can ascertain x from X or x from Y, then X and Y are said to computationally indistinguishable, denoted as $X \overset{c}{\equiv} Y$.

We now formally define outcome verifiability, input privacy, and function privacy as follows.

Definition 8 (Outcome Verifiability)**.** The property of correctness states that $\forall \mathbf{z}$, if $(V, \sigma) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{C})$ and $V \leftarrow \mathsf{Encrypt}(\mathsf{PK}, f(\mathbf{z}))$, then $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{z}, v, \sigma)$.

The property of unforgeability states that no PPT adversary $\mathcal{A}$ has more than negligible probability in winning the following game between $\mathcal{A}$ and an simulator $\mathcal{S}$ as follows. (1) **Initialization:** $\mathcal{A}$ commits to a random data vector $\mathbf{z}$. $\mathcal{S}$ runs the algorithm KeyGen, gives PK to $\mathcal{A}$, and maintains SK secret; (2) **Setup:** $\mathcal{A}$ makes an oracle query the the algorithm Setup, specifying the function f. $\mathcal{S}$ answers the query by returning $\mathsf{FK}(f)$; and (3) **Forgery:** $\mathcal{A}$ outputs a forgery for the committed point $\mathbf{z}$, including a claimed outcome v' of f at $\mathbf{z}$ and a signature of correct computation σ'. If the forgery passes the algorithm Verify but $v' \neq f(\mathbf{z})$, namely, if $1 \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{z}, v', \sigma')$ and $v' \neq f(\mathbf{z})$, then $\mathcal{A}$ wins the game.

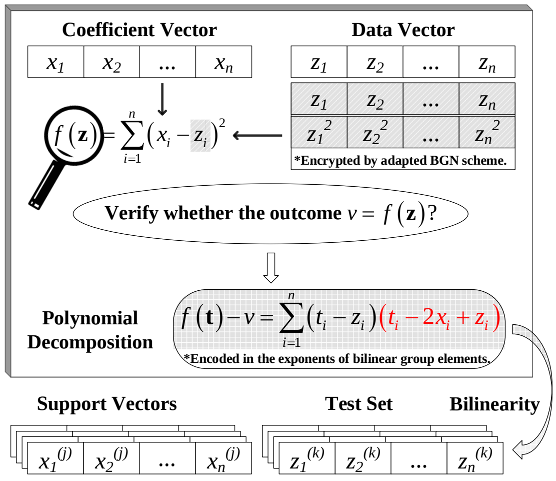

Coefficient Vector
Data Vector
x_1 x_2 ... x_n
z_1 z_2 ... z_n
z_1^2 z_2^2 ... z_n^2
$f(\mathbf{z}) = \sum_{i=1}^{n} (x_i - z_i)^2$
*Encrypted by adapted BGN scheme.
Verify whether the outcome $v = f(\mathbf{z})$?
Polynomial Decomposition
$f(\mathbf{t}) - v = \sum_{i=1}^{n} (t_i - z_i)(t_i - 2x_i + z_i)$
*Encoded in the exponents of bilinear group elements.
Support Vectors
Test Set
Bilinearity
$x_1^{(j)}$ $x_2^{(j)}$ ... $x_n^{(j)}$
$z_1^{(k)}$ $z_2^{(k)}$ ... $z_n^{(k)}$

Fig. 1. An Illustration of MVP Supporting SVMs with the RBF Kernel.

Definition 9 (Function Privacy)**.** Let $\mathcal{V}_C$ denote the view of a customer in the execution of the scheme in Definition 5. $\mathcal{V}_C$ mainly covers the customer's input data vector $\mathbf{z}$ and all the outputs of the scheme returned to the customer, including the result v and the signature of correctness σ. The guarantee of function privacy requires that there exists a polynomial-time simulator $\mathcal{S}_C$ such that $\mathcal{S}_C(\mathbf{z}, v) \overset{c}{\equiv} \mathcal{V}_C$. In other words, the coefficient vector $\mathbf{x}$ is hidden from $\mathcal{V}_C$.

Definition 10 (Input Privacy)**.** Let $\mathcal{V}_S$ denotes the view of the service provider in the execution of the scheme in Definition 5. The guarantee of input privacy requires the existence of a polynomial-time simulator $\mathcal{S}_S$ such that $\mathcal{S}_S(\mathbf{x}) \overset{c}{\equiv} \mathcal{V}_S$. In other words, the data vector $\mathbf{z}$ is hidden from $\mathcal{V}_S$.

We interpret the intuitions behind the definitions of function privacy and input privacy. After the execution of the secure scheme, the customer learns only its data vector $\mathbf{z}$ and the returned result v, while the coefficient vector $\mathbf{x}$ of the function f is hidden. Similarly, the service provider holds only its coefficient vector $\mathbf{x}$, but does not learn any information about the customer's data vector $\mathbf{z}$, not even the result v returned to the customer.

4 Design Principles of MVP

We give an overview of our proposed scheme MVP and illustrate its key design principles.

We adopt a bottom-up design holistically. In Section 5, we first consider two basic multivariate polynomials underlying common ML algorithms, namely dot product and squared Euclidean distance, and thus develop two building blocks of MVP, satisfying the desired security requirements. In Section 6, we then apply and further upgrade the building blocks to support practical ML predictions. With the help of Fig. 1, we illustrate the design principles as follows.

A *naïve method*, enabling the customer to verify the correctness of polynomial evaluation while maintaining function privacy, is that the service provider can employ homomorphic encryption to encrypt the coefficient vector $\mathbf{x}$ of the polynomial f and let the customer obliviously reevaluate its input $\mathbf{z}$. However, it defeats the requirement of computation efficiency at the verifier. Therefore, we turn to the following lemma for verification.

Lemma 1 (Polynomial Decomposition)**.** *Let $f(\mathbf{u})$ be an n-variate polynomial in $\mathbb{Z}$, where $\mathbf{u} = (u_1, u_2, \ldots, u_n) \in \mathbb{Z}^n$. For any $\mathbf{z} \in \mathbb{Z}^n$, there exist polynomials $w_i(\mathbf{u})$'s such that*

$$f(\mathbf{u}) - f(\mathbf{z}) = \sum_{i=1}^{n} (u_i - z_i)\, w_i(\mathbf{u}). \tag{1}$$

In addition, there exists an efficient algorithm to find $w_i(\mathbf{u})$'s [38].

Based on Lemma 1, we let the model manager first choose a random point $\mathbf{t}$ (i.e., the variable $\mathbf{u}$ now takes the value of $\mathbf{t}$) and then create the function public key involving $f(\mathbf{t})$. Later in the computation stage, given the data vector $\mathbf{z}$ from the customer, when the service provider wants to prove that v is indeed the result of $f(\mathbf{z})$, it needs to compute the polynomials $w_i(\mathbf{t})$'s such that Lemma 1 holds. In addition, $w_i(\mathbf{t})$'s will be used to construct the signatures vouching for the correctness of computation. If the claimed outcome v is correct, then $f(\mathbf{t}) - v = \sum_{i=1}^{n}(t_i - z_i)w_i(\mathbf{t})$ must hold, since it is equivalent to both sides of Equation (1) evaluated at the commitment point $\mathbf{t}$. We note that in real construction, considering the secrecy of f and $\mathbf{t}$, the terms in the above equation are all encoded in the exponents of bilinear group elements. In particular, the pairing operation over the bilinear groups allows to express one multiplication in the exponent. Additionally, to assist the service provider in raising $w_i(\mathbf{t})$'s to the exponents, the public key should contain appropriate helper terms. Moreover, $w_i(\mathbf{t})$'s normally contain the coefficients x_i's, and directly encoding $w_i(\mathbf{t})$'s in the exponents can still leak function privacy. Thus, the service provider needs to perform some extra maskings. For example, it can bring in random integers for shifting and scaling, whose securities can reduce to the hardness of the discrete logarithm problem.

However, the input privacy causes significant obstacles to implementing the above verification program. Specifically, the terms $w_i(\mathbf{t})$'s may also involve the data vector $\mathbf{z}$ (e.g., in the case of squared Euclidean distance shown in Fig. 1). In other words, the service provider needs to encode $\mathbf{z}$ as exponents of group elements, without knowing its contents. Additionally, it also needs to compute dot product and squared Euclidean distance over the ciphertexts of $\mathbf{z}$. To jointly handle these two problems, we employ the stripped-down BGN cryptosystem introduced above. On the one hand, the BGN-type ciphertext is essentially an encoding of the plaintext in the exponent and thus can be exploited in the proof generation. On the other hand, homomorphic properties facilitate evaluating two designated types of multivariate polynomials on ciphertexts efficiently.

Last, to further reduce latency when handling numerous test data in practice, we incorporate the bilinear property of the pairing operation to support batch verification. Accordingly, we generate the proofs of correct computations in an aggregate mode. Hence, the communication overhead can be substantially reduced. The main insight behind our optimization is that when performing verifications for single test data and multiple test data, the data vector (i.e., test data) and the coefficient vectors (e.g., support vectors in SVMs) remain unchanged, respectively. Therefore, by applying bilinearity, we can aggregate the verification formulas involving the coefficient vectors (e.g., support vectors) and the data vectors (i.e., test set), respectively. We note that our batch verification scheme and the amortized efficiency are general, because the parameters of all ML models (e.g., the weights and biases in neural networks) are fixed in the prediction phase, and the bilinearity can always be applied.

5 Building Blocks of MVP

In this section, we present the building blocks of MVP: verifiable and privacy preserving dot product and squared Euclidean distance computations. In particular, dot product can be regarded as an elementary mathematical operation of most ML algorithms, ranging from linear models for regression/classification to neural networks. Yet, Euclidean distance, also known as L_2 norm in regularization theory, is an important distance measure widely used in clustering and classification, such as k-means clustering, learning vector quantization, SVMs with RBF kernels, etc.

5.1 Secure Dot Product Computation

We first consider the dot product between a coefficient vector $\mathbf{x}$ and a data vector $\mathbf{z}$, i.e., $f(\mathbf{z}) = \mathbf{x}^T\mathbf{z} = \sum_{i=1}^{n} x_i z_i$.

Algorithm $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\tau, f)$: The model manager sets up the parameters for the adapted BGN cryptosystem as shown in Section 2.2. In addition, it selects a random point $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ where $t_i \in \mathbb{Z}_q^*$ and computes the signature generation set as

$$W = \left\{ \left\{ {g_1}^{t_i}, h^{t_i} | i \in [n] \right\}, {g_1}^{\sum_{i=1}^{n} {t_i}^2} \right\}. \tag{2}$$

The public key $\mathsf{PK} = \{q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, h, (g, g^s), (g_1, g_2), W\}$ is posted on the certificated bulletin board. The secret key SK, containing the private key s for decryption and the commitment point $\mathbf{t}$, is maintained by the model manager. For consistency with the adapted BGN cryptosystem, g_1 is used as the major base throughout this work.

Algorithm $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{PK}, f)$: By using the signature generation set W contained in PK, the model manager computes the function public key $\mathsf{FK}(f)$, where

$$\mathsf{FK}(f) = \prod_{i=1}^{n} \left({g_1}^{t_i}\right)^{x_i} = {g_1}^{\sum_{i=1}^{n} x_i t_i} = {g_1}^{\mathbf{x}^T\mathbf{t}} = {g_1}^{f(\mathbf{t})}. \tag{3}$$

Algorithm $\mathbf{C} \leftarrow \mathsf{Encrypt}(\mathsf{PK}, \mathbf{z})$: To guarantee input privacy against the service provider, the customer picks a set of n random numbers $\left\{r_i \in \mathbb{Z}_q^* | i \in [n]\right\}$ and employs the adapted BGN scheme to encrypt its data vector $\mathbf{z}$ as

$$\mathbf{C} = \left\{ \left({g_1}^{z_i} g^{r_i}, {g_2}^{z_i} g^{s r_i}\right) \in G | i \in [n] \right\}. \tag{4}$$

Algorithm $(V, \sigma) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{C})$: The service provider can obliviously evaluate $f(\mathbf{z})$ over the ciphertext set $\mathbf{C}$. It first applies multiplication by a constant and then homomorphic addition, and it gets the encrypted result:

$$\begin{aligned} V &= \prod_{i=1}^{n} \left({g_1}^{z_i} g^{r_i}, {g_2}^{z_i} g^{s r_i}\right)^{x_i} = \prod_{i=1}^{n} \left({g_1}^{x_i z_i} g^{x_i r_i}, {g_2}^{x_i z_i} g^{s x_i r_i}\right) \\ &= \left({g_1}^{\sum_{i=1}^{n} x_i z_i} g^{\sum_{i=1}^{n} x_i r_i}, {g_2}^{\sum_{i=1}^{n} x_i z_i} g^{s \sum_{i=1}^{n} x_i r_i}\right) \\ &= \left({g_1}^{f(\mathbf{z})} g^{\sum_{i=1}^{n} x_i r_i}, {g_2}^{f(\mathbf{z})} g^{s \sum_{i=1}^{n} x_i r_i}\right) \in G. \end{aligned} \tag{5}$$

We note that if correctly computed, V should be an encryption of $f(\mathbf{z})$ under the random number $\sum_{i=1}^{n} x_i r_i$.

To facilitate outcome verification, using Lemma 1, the service provider is able to find an appropriate set of polynomials $w_1(\mathbf{t}), w_2(\mathbf{t}), \ldots, w_n(\mathbf{t})$ to express $f(\mathbf{t}) - f(\mathbf{z})$ as $\sum_{i=1}^{n}(t_i - z_i)w_i(\mathbf{t})$. Here in the dot product, we can observe that $w_i(\mathbf{t}) = x_i$ for all $i \in [n]$. Additionally, to preserve function privacy, namely to hide the coefficients x_i's, the service provider needs to do some maskings on the exponents of these coefficients. It first chooses a random integer $d \in \mathbb{Z}_q^*$ and computes $\sigma_0 = h^d \in \mathbb{G}_2$. It then generates a signature of correctness $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ such that

$$\sigma_i = {g_1}^{dx_i}\left({g_1}^{t_i}\right)^d = \left({g_1}^{x_i+t_i}\right)^d = \left({g_1}^{w_i(\mathbf{t})+t_i}\right)^d. \tag{6}$$

Here, ${g_1}^{t_i}$ from the signature generation set W is added for shifting, while the random number d is added for further scaling. In addition, the witness σ_0 is to balance d introduced in the signature of correctness σ for later verification. Moreover, σ_0 is kept secret by the service provider and remains unchanged in the following designs.

The service provider finally sends the encrypted result V to the model manager for decryption and forwards the signature of correctness σ to the customer for verification.

Algorithm $v \leftarrow \mathsf{Decrypt}(\mathsf{SK}, V)$: The model manager first uses the private key s to project the ciphertext V:

$$\begin{aligned}\pi\left(V\right) &= \left({g_1}^{f(\mathbf{z})} g^{\sum_{i=1}^{n} x_i r_i}\right)^s \left({g_2}^{f(\mathbf{z})} g^{s\sum_{i=1}^{n} x_i r_i}\right)^{-1} \\ &= \left({g_1}^{s}{g_2}^{-1}\right)^{f(\mathbf{z})} = \pi\left((g_1, g_2)\right)^{f(\mathbf{z})}.\end{aligned} \tag{7}$$

Then, given a limited domain of f, the model manager can recover the result v by computing the logarithm of $\pi(V)$ to the base $\pi((g_1, g_2))$. After decryption, the model manager returns v to the customer.

Algorithm $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{z}, v, \sigma)$: To verify that v is indeed the correct outcome of f evaluated for the data vector $\mathbf{z}$, the customer first uses the signature generation set W to generate a helper value:

$$\begin{aligned}\mathsf{H}(f) &= {g_1}^{\sum_{i=1}^{n} {t_i}^2}\left(\prod_{i=1}^{n}\left({g_1}^{t_i}\right)^{z_i}\right)^{-1} \\ &= {g_1}^{\sum_{i=1}^{n} t_i(t_i - z_i)},\end{aligned} \tag{8}$$

and then checks whether the following equation holds:

$$\hat{e}\left(\mathsf{FK}(f){g_1}^{-v}\mathsf{H}(f), \sigma_0\right) \overset{?}{=} \prod_{i=1}^{n}\hat{e}\left(\sigma_i, h^{t_i - z_i}\right). \tag{9}$$

Here, since the witness σ_0 on the left hand side (LHS) is maintained by the service provider, it needs to assist the customer in performing the pairing operation, i.e., $\hat{e}(\cdot, \sigma_0)$. In addition, $\mathsf{FK}(f)$ equals ${g_1}^{f(\mathbf{t})}$, and h^{t_i} is contained in W. Moreover, by using the bilinearity of asymmetric pairing, the helper value $\mathsf{H}(f)$ on the LHS is able to balance each shifting term ${g_1}^{t_i}$ added in the signature σ_i on the right hand side (RHS). The customer accepts v if the above equation holds; otherwise, it rejects.

5.2 Secure Squared Euclidean Distance Computation

We next consider the squared Euclidean distance between a coefficient vector $\mathbf{x}$ and a data vector $\mathbf{z}$, i.e., $f(\mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_2^2 = \sum_{i=1}^{n}(x_i - z_i)^2$.

Algorithm $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{KeyGen}(\tau, f)$: The key initialization phase is the same as that in the dot product protocol. The difference lies in the signature generation set W, where

$$W = \left\{g^{t_i}, {g_1}^{t_i}, {g_1}^{{t_i}^2}, h^{t_i} | i \in [n]\right\}. \tag{10}$$

Here, g^{t_i} is to support the helper value generation, and ${g_1}^{{t_i}^2}$ is to facilitate the function public key generation.

Algorithm $\mathsf{FK}(f) \leftarrow \mathsf{Setup}(\mathsf{PK}, f)$: The model manager uses the signature generation set W to compute

$$\begin{aligned}\mathsf{FK}(f) &= \prod_{i=1}^{n}\left({g_1}^{{x_i}^2}\left({g_1}^{t_i}\right)^{-2x_i}{g_1}^{{t_i}^2}\right) \\ &= {g_1}^{\sum_{i=1}^{n}(x_i - t_i)^2} = {g_1}^{\|\mathbf{x}-\mathbf{t}\|_2^2} = {g_1}^{f(\mathbf{t})}\end{aligned} \tag{11}$$

as the function public key.

Algorithm $\mathbf{C} \leftarrow \mathsf{Encrypt}(\mathsf{PK}, \mathbf{z})$: To facilitate computing squared Euclidean distance while preserving input privacy, the customer first picks a set of $2n$ random integers $\left\{r_i^1, r_i^2 \in \mathbb{Z}_q^* | i \in [n]\right\}$. Then, it computes $\mathbf{C} = \left\{C_i^1, C_i^2 | i \in [n]\right\}$ as the ciphertext set, where

$$C_i^1 = \left({g_1}^{z_i} g^{r_i^1}, {g_2}^{z_i} g^{sr_i^1}\right) \in G, \tag{12}$$

$$C_i^2 = \left({g_1}^{{z_i}^2} g^{r_i^2}, {g_2}^{{z_i}^2} g^{sr_i^2}\right) \in G, \tag{13}$$

are BGN encryptions of $z_i, {z_i}^2$ under r_i^1, r_i^2, respectively.

In general, compared with the time-consuming computation on ciphertexts, the evaluation of plaintexts is quite more efficient. Therefore, in addition to original data, we also let the customer encrypt its square, which can avoid homomorphic multiplications at the service provider while incurring a small encryption overhead at the customer.

Algorithm $(V, \sigma) \leftarrow \mathsf{Compute}(\mathsf{PK}, f, \mathbf{C})$: The service provider can obliviously evaluate $f(\mathbf{z})$ via homomorphic properties. It first computes

$$\begin{aligned}V_i &= (g_1, g_2)^{{x_i}^2}\left({g_1}^{z_i} g^{r_i^1}, {g_2}^{z_i} g^{sr_i^1}\right)^{-2x_i}\left({g_1}^{{z_i}^2} g^{r_i^2}, {g_2}^{{z_i}^2} g^{sr_i^2}\right) \\ &= \left({g_1}^{(x_i - z_i)^2} g^{-2x_i r_i^1 + r_i^2}, {g_2}^{(x_i - z_i)^2} g^{s\left(-2x_i r_i^1 + r_i^2\right)}\right)\end{aligned}$$

for all $i \in [n]$. It then applies homomorphic additions to all V_i's and obtains

$$\begin{aligned}V = \prod_{i=1}^{n} V_i &= \prod_{i=1}^{n}\left({g_1}^{(x_i - z_i)^2} g^{\bar{r}_i}, {g_2}^{(x_i - z_i)^2} g^{s\bar{r}_i}\right) \\ &= \left({g_1}^{\sum_{i=1}^{n}(x_i - z_i)^2} g^{\sum_{i=1}^{n}\bar{r}_i}, {g_2}^{\sum_{i=1}^{n}(x_i - z_i)^2} g^{s\sum_{i=1}^{n}\bar{r}_i}\right) \\ &= \left({g_1}^{f(\mathbf{z})} g^{\sum_{i=1}^{n}\bar{r}_i}, {g_2}^{f(\mathbf{z})} g^{s\sum_{i=1}^{n}\bar{r}_i}\right) \in G,\end{aligned} \tag{14}$$

where $\bar{r}_i$ denotes $-2x_i r_i^1 + r_i^2$ for the sake of brevity. If evaluated correctly, V is actually an encryption of $f(\mathbf{z})$ under the random number $\sum_{i=1}^{n}\bar{r}_i = \sum_{i=1}^{n}\left(-2x_i r_i^1 + r_i^2\right)$.

To help verify the correctness of evaluation, the service provider computes $w_1(\mathbf{t}), w_2(\mathbf{t}), \ldots, w_n(\mathbf{t})$ such that the relation of Lemma 1 holds. In the squared Euclidean distance considered here, $w_i(\mathbf{t}) = t_i - 2x_i + z_i$. We note that $w_i(\mathbf{t})$ in the dot product only contains x_i, which is known to the service provider. In contrast, $w_i(\mathbf{t})$ in the squared Euclidean distance also comprises t_i and z_i, which are both kept secret from the service provider. This condition makes the task of encoding $w_i(\mathbf{t})$ in the exponent extremely hard, although ${g_1}^{t_i}$ can be fetched from the signature generation set W. To deal with z_i, the service provider can first embed the first part of its BGN-type ciphertext, i.e., ${g_1}^{z_i} g^{r_i^1}$, into the signature. Later, with the help of g^{t_i} in W, the customer, knowing its input z_i and corresponding random number r_i^1, can eliminate the part involving $g^{r_i^1}$ in the verification formula. Furthermore, to guarantee function privacy, just as in the dot product, the service provider first chooses a random integer $d \in \mathbb{Z}_q^*$ and computes $\sigma_0 = h^d$. It then generates the signature of correctness $\sigma = (\sigma_1, \ldots, \sigma_n)$, where

$$\sigma_i = \left({g_1}^{t_i} {g_1}^{-2x_i} \left({g_1}^{z_i} g^{r_i^1}\right)\right)^d = \left({g_1}^{w_i(\mathbf{t})} g^{r_i^1}\right)^d . \tag{15}$$

Algorithm $v \leftarrow \mathsf{Decrypt}(\mathsf{SK}, V)$: The same as that in the dot product protocol.

Algorithm $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{PK}, \mathsf{FK}(f), \mathbf{z}, v, \sigma_0, \sigma)$: To verify the correctness of v, the customer first uses g and $\{g^{t_i} | i \in [n]\}$ in the public key PK to compute a helper value:

$$\begin{aligned} \mathsf{H}(f) &= \left(g^{\left(\sum_{i=1}^n r_i^1 z_i\right)}\right)^{-1} \prod_{i=1}^n \left(g^{t_i}\right)^{r_i^1} \\ &= g^{\sum_{i=1}^n r_i^1 (t_i - z_i)}, \end{aligned} \tag{16}$$

and then checks whether the following equation holds:

$$\hat{e}\left(\mathsf{FK}(f){g_1}^{-v}\mathsf{H}(f), \sigma_0\right) \stackrel{?}{=} \prod_{i=1}^n \hat{e}\left(\sigma_i, h^{t_i - z_i}\right). \tag{17}$$

We note that the helper term $\mathsf{H}(f)$ on the LHS is to balance the term $g^{r_i^1}$ introduced in the signature σ_i on the RHS.

5.3 Performance Analysis

We analyze the time and communication complexities of the proposed secure dot product and squared Euclidean distance computation protocols. For the time complexity analysis, we focus only on the expensive group operations, including pairing and exponentiation, while ignoring other operations with quite low overhead (e.g., the operations over plaintexts). We let T_{pair} and T_{exp} denote one-time overhead of pairing and exponentiation, respectively, where $T_{pair} \gg T_{exp}$. For both protocols, the time complexities of the customer, the service provider, and the model manager are $nT_{pair} + O(n)T_{exp}$, $1T_{pair} + O(n)T_{exp}$, and $O(n)T_{exp}$, respectively. Their communication complexities are $O(n)$, $O(n)$, and $O(1)$, respectively.

5.4 Security Analysis

We give the following three propositions to analyze outcome verifiability, function privacy, and input privacy in two building blocks of MVP. We also revisit and distribute the trust on the model manager.

Proposition 1. *Outcome verifiability is guaranteed in terms of correctness and unforgeability under Definition 8.*

Proof. We give a proof sketch here and defer the detailed proof to Appendix A. First, we prove correctness by showing the validnesses of the verification formulas, i.e., Equation (9) and Equation (17) in the dot product and the squared Euclidean distance protocols, respectively. These two equations follow from Lemma 1 and the bilinearity of asymmetric pairing. Second, we prove unforgeability by reducing to the ℓ-SDH assumption in Definition 4. The key idea is to build a simulator $\mathcal{S}$, which obtains an instance of the ℓ-SDH assumption from a challenger $\mathcal{C}$, where $\ell = 1$ and $\ell = 2$ for the dot product and the squared Euclidean distance protocols, respectively. $\mathcal{S}$ then embeds this ℓ-SDH instance into an instance of the algorithm Verify such that if an adversary $\mathcal{A}$ can break the security of Verify with more than negligible probability, $\mathcal{S}$ can leverage $\mathcal{A}$ to break the ℓ-SDH instance also with non-negligible probability, contradicting the ℓ-SDH assumption. □

Proposition 2. *Function privacy is preserved from the customer and external attackers under Definition 9.*

Proof. Guaranteeing function privacy is to hide the coefficient vector $\mathbf{x}$. In addition, the information involving $\mathbf{x}$, available to the adversaries (e.g., the customer), contains the signature of correctness σ and the outcome v. We prove no information leakage in terms of these two parts.

First, the adversaries cannot derive $\mathbf{x}$ from σ. We prove this point in two building blocks separately. For the dot product, the coefficient x_i in the signature $\sigma_i = \left({g_1}^{x_i + t_i}\right)^d$ has been shifted by a random integer t_i and further scaled by another d. We give the formal proof through the relation of computational indistinguishability. We consider the following two computationally indistinguishable pairs:

$$\forall i \in [n], \sigma_i = \left({g_1}^{x_i + t_i}\right)^d \stackrel{c}{\equiv} X \in {}_R\mathbb{G}_1, \tag{18}$$

$$\forall i \neq i' \in [n], \sigma_i = \left({g_1}^d\right)^{x_i + t_i} \stackrel{c}{\equiv} \sigma_{i'} = \left({g_1}^d\right)^{x_{i'} + t_{i'}}, \tag{19}$$

where the first pair says that each signature is a uniform element of the cyclic group $\mathbb{G}_1$, and the second pair indicates that any two distinct signatures have no correlation. In addition, the first and the second pairs follow from the randomnesses of d and t_i, by regarding ${g_1}^{x_i + t_i}$ and ${g_1}^d$ as generators of $\mathbb{G}_1$, respectively. Moreover, the confidentialities of d and t_i can be reduced to the intractabilities of the DLP in $\mathbb{G}_T$ and $\mathbb{G}_1$, i.e.,

$$\hat{e}\left(\cdot, h\right), \hat{e}\left(\cdot, \sigma_0\right) = \hat{e}\left(\cdot, h\right)^d \in \mathbb{G}_T \Rightarrow d, \tag{20}$$

$$g_1, {g_1}^{t_i} \in \mathbb{G}_1 \Rightarrow t_i, \tag{21}$$

are both hard for the PPT adversaries. Therefore, each coefficient x_i cannot be leaked from its corresponding signature σ_i and the other signatures, namely the signature of correctness σ. For the squared Euclidean distance, we first rewrite g as ${g_1}^{s_1}$ with some implicit $s_1 \in \mathbb{Z}_q^*$. We then consider two computationally indistinguishable pairs:

$$\forall i \in [n], \sigma_i = ({g_1}^{t_i - 2x_i + z_i} g^{r_i^1})^d \stackrel{c}{\equiv} X \in {}_R\mathbb{G}_1, \tag{22}$$

$$\begin{aligned} \forall i \neq i' \in [n], \sigma_i &= \left({g_1}^d\right)^{t_i - 2x_i + z_i + s_1 r_i^1} \\ &\stackrel{c}{\equiv} \sigma_{i'} = \left({g_1}^d\right)^{t_{i'} - 2x_{i'} + z_{i'} + s_1 r_{i'}^1}. \end{aligned} \tag{23}$$

Here, the first and the second pairs follow from the randomnesses of d and t_i (or r_i^1), by viewing ${g_1}^{t_i-2x_i+z_i}g^{r_i^1}$ and ${g_1}^d$ as generators of $\mathbb{G}_1$, respectively. In addition, just as in the dot product, the securities of d and t_i can be reduced to the hardnesses of the DLP in $\mathbb{G}_T$ and $\mathbb{G}_1$, respectively. Thus, σ reveals no information about each coefficient x_i.

Second, although the customer knows the outcome $v = f(\mathbf{z})$ and its data vector $\mathbf{z}$, it cannot restore the coefficient vector $\mathbf{x}$ yet. The reason is that $f(\mathbf{z})$'s in the dot product and the squared Euclidean distance protocols are essentially two diophantine equations with n variables, i.e., $\{x_i | i \in [n]\}$. However, the customer just has one equation. Thus, when $n > 1$, it is computationally infeasible for the customer to learn x_i's. Furthermore, in Section 6.1.4, we will show how MVP thwarts such an equation solving attack, when the customer queries multiple test data. □

Proposition 3. *Input privacy is preserved from the service provider under Definition 10.*

Proof. The elements of the data vector $\mathbf{z}$ have been encrypted with the adapted BGN cryptosystem, which provides semantic security under the SXDH assumption [25]. By definition, the service provider, as a PPT adversary, cannot reveal the contents of the data vector $\mathbf{z}$. Furthermore, the input privacy cannot be breached from the outcome v, since it is returned/known to the customer rather than the service provider after the algorithm $\mathsf{Decrypt}$. □

Remark 1 (Trust Assumption on Model Manager Revisited)**.** We illustrate how to distribute the trust on the model manager from its functionalities.

First regards outcome verification. The model manager performs two related tasks to facilitate outcome verification: one is to select a random point $\mathbf{t}$ in the algorithm KeyGen; the other is to compute the function public key $\mathsf{FK}(f)$ in the algorithm Setup. Accordingly, the trust assumption on the model manager includes two aspects: first, it should keep $\mathbf{t}$ secret and cannot leak it to irrelevant system participants; second, it maintains the registration database of trained models and should correctly set up the function public key $\mathsf{FK}(f)$. We note that Setup in two building blocks executes with the signature generation set W in the public key PK rather than the random point $\mathbf{t}$ in the secret key SK. Such an insightful design enables us to distribute the trust on the model manager by introducing two non-colluding entities to perform KeyGen and Setup independently, which is a common distributed setting used by previous works [39], [40]. We further show that the major roles of the model manager can be substituted in feasible ways. First, the random point $\mathbf{t}$ can be cooperatively selected by a group of customers. For example, each of n customers selects one distinct dimension of $\mathbf{t}$ and then computes its corresponding elements in W, i.e., the i-th customer selects t_i and needs to compute $\{{g_1}^{t_i}, {g_1}^{{t_i}^2}, h^{t_i}\}$ (resp., $\{g^{t_i}, {g_1}^{t_i}, {g_1}^{{t_i}^2}, h^{t_i}\}$) for the dot product protocol (resp., the squared Euclidean distance protocol). In particular, keeping $\mathbf{t}$ secret is to ensure outcome verifiability at the customers. Hence, we let themselves cooperatively select $\mathbf{t}$, and they have no proper incentives to leak it to others. Second, the function public key $\mathsf{FK}(f)$ can be set up by the service provider. In this situation, a natural analogue of the trust assumption on the service provider is that it should correctly execute Setup without knowing the random point $\mathbf{t}$. An alternative assumption turns into defining the correctness of outcome verifiability as the consistency between the function public key $\mathsf{FK}(f)$ and the outcome $f(\mathbf{z})$, where the former is computed at the random point $\mathbf{t}$ in Setup, and the latter is evaluated at the data vector $\mathbf{z}$ in $\mathsf{Compute}$. Specifically, both $\mathbf{t}$ and $\mathbf{z}$ are unknown to the service provider.

Second regards decryption. We can adopt a threshold version of the adapted BGN scheme and introduce multiple model managers to jointly perform decryption. In particular, by using the standard Pedersen's technique [41], the private key s can be split across these model managers such that only if a minimum (no less than the threshold) number of models manager participate, they can decrypt a ciphertext.

6 Design of MVP

In this section, we use the building blocks to support practical ML algorithms and thus propose MVP. We first elaborate on SVMs and then illustrate how to support others.

6.1 Supporting SVMs

We first present two key parts of MVP, namely, oblivious evaluation and outcome verification. We then analyze performance and security. In what follows, we focus mainly on two types of kernels widely used in SVMs, including the polynomial kernel $K(\mathbf{x}_j, \mathbf{z}_k) = (\gamma {\mathbf{x}_j}^T \mathbf{z}_k + c)^p$ and the radial basis function (RBF) kernel $K(\mathbf{x}_j, \mathbf{z}_k) = \exp(-\gamma \|\mathbf{x}_j - \mathbf{z}_k\|_2^2)$, where γ, c, and p are parameters for kernel functions. We also set the degree of the polynomial kernel $p \geq 2$ by default. In fact, when $p = 1$, the polynomial kernel degenerates into linear kernel, which can be handled by applying the dot product protocol only once. Detailed design for the linear kernel is deferred to Appendix B.

6.1.1 Oblivious Evaluation

We introduce the first part of MVP, i.e., how the service provider can obliviously evaluate each test data $\mathbf{z}_k$. We assume that without affecting the final classification results, the service provider and the customer agree on a common scaling and rounding formula, which transforms float-type parameters or data into integers if necessary [14]. In addition, for simplicity in notations, we use $\psi_j(\mathbf{z}_k)$ to represent the basic function inside the kernel operator $K(\mathbf{x}_j, \mathbf{z}_k)$ (i.e., $\psi_j(\mathbf{z}_k) = {\mathbf{x}_j}^T \mathbf{z}_k$ for the polynomial kernel, and $\psi_j(\mathbf{z}_k) = \|\mathbf{x}_j - \mathbf{z}_k\|_2^2$ for the RBF kernel). Thus, SVMs with the polynomial kernel and the RBF kernel should invoke the algorithms in the dot product and the squared Euclidean distance protocols, respectively. We introduce the details of oblivious evaluation as follows.

When the customer intends to query an SVM prediction service, it first encrypts $\mathbf{z}_k$ using the algorithm $\mathsf{Encrypt}$. Then, it sends the ciphertext set to the service provider. Considering high-degree polynomial or exponential function in $f(\mathbf{z}_k)$, the service provider computes step by step.

The service provider first obliviously evaluates the basic function inside the kernel for all the support vectors (i.e., $\{\psi_j(\mathbf{z}_k) |\ j \in \mathcal{SV}\}$) with the algorithm $\mathsf{Compute}$. Afterwards, it sends the encrypted intermediate results, denoted by $\{V_j^{(k)} | j \in \mathcal{SV}\}$, to the model manager. To facilitate further computation, the model manager executes the algorithm $\mathsf{Decrypt}$ for $\{V_j^{(k)} | j \in \mathcal{SV}\}$ and returns the plaintexts $\{v_j^{(k)} | j \in \mathcal{SV}\}$ to the customer. With these intermediate results, the customer can compute the complete kernel functions, i.e., $\{\bar{v}_j^{(k)} = (\gamma v_j^{(k)} + c)^p | j \in \mathcal{SV}\}$ in the polynomial kernel and $\{\bar{v}_j^{(k)} = \exp(-\gamma v_j^{(k)}) | j \in \mathcal{SV}\}$ in the RBF kernel. Now, the SVM classifier becomes

$$f(\mathbf{z}_k) = \sum_{j \in \mathcal{SV}} y_j \alpha_j \bar{v}_j^{(k)} + b. \tag{24}$$

Here, the service provider may also need to provide the dual coefficients $\{\alpha_j y_j | j \in \mathcal{SV}\}$ and the intercept term b, given which the customer can finally derive $f(\mathbf{z}_k)$.

In addition to support vectors, if the service provider intends to hide the dual coefficients and the intercept term, the dot product protocol can be applied. Specifically, $(y_j \alpha_j | j \in \mathcal{SV}, b)$ and $(\bar{v}_j^{(k)} | j \in \mathcal{SV}, 1)$ in Equation (24) serve as the coefficient vector and the data vector, respectively. The detailed design is similar to the design for linear kernel, and the analogy is deferred to Remark 2 in Appendix B.

6.1.2 Outcome Verification

We next introduce the other part of MVP, i.e., how the customer can verify the computations done by the service provider. In particular, the customer needs to check whether each intermediate result $v_j^{(k)}$ is correctly computed from the support vector $\mathbf{x}_j$ and the test data $\mathbf{z}_k$, i.e., whether $\forall k \in [\phi], \forall j \in \mathcal{SV}, v_j^{(k)} = \psi_j(\mathbf{z}_k)$. We show the detailed individual and batch verification schemes for the polynomial kernel and the RBF kernel separately. For the sake of brevity, if the term is independent of the test data $\mathbf{z}_k$, we will omit the superscript "(k)" from its top right corner; otherwise, we will reserve this superscript.

Design for Polynomial Kernel: We first consider the case of single test data $\mathbf{z}_k$. To verify the correctness of $\{v_j^{(k)} | j \in \mathcal{SV}\}$, an explicit way for the customer is to perform the algorithm Verify in the dot product protocol for $|\mathcal{SV}|$ times, where $\mathbf{z}_k$ functions as the data vector, and $\{\mathbf{x}_j | j \in \mathcal{SV}\}$ serves as $|\mathcal{SV}|$ distinct coefficient vectors. Specifically, the individual verification formulas are

$$\begin{aligned} \forall j \in \mathcal{SV}, \quad & \hat{e}\left(\mathsf{FK}(\psi_j) {g_1}^{-v_j^{(k)}} \mathsf{H}(\psi_j^{(k)}), \sigma_0\right) \\ \overset{?}{=} \quad & \prod_{i=1}^{n} \hat{e}\left(\sigma_i^{(j)}, h^{t_i - z_i^{(k)}}\right), \end{aligned} \tag{25}$$

where the expressions of the function public key $\mathsf{FK}(\psi_j)$, the signature $\sigma_i^{(j)}$, and the helper value $\mathsf{H}(\psi_j^{(k)})$ can be obtained by assigning values $x_i^{(j)}, z_i^{(k)}$ to the variables x_i, z_i, respectively, in Equations (3), Equations (6), and Equations (8). However, we note that σ_0 on the LHS and $h^{t_i - z_i^{(k)}}$ on the RHS are both independent of j. Therefore, by using the bilinearity of asymmetric pairing, we can aggregate both sides of Equation (25) for all $j \in \mathcal{SV}$ and deduce the batch verification formula at the support vector level:

$$\begin{aligned} & \hat{e}\left(\prod_{j \in \mathcal{SV}} \mathsf{FK}\left(\psi_j\right) {g_1}^{-\sum_{j \in \mathcal{SV}} v_j^{(k)}} \mathsf{H}\left(\psi_j^{(k)}\right)^{|\mathcal{SV}|}, \sigma_0\right) \\ \overset{?}{=} \quad & \prod_{i=1}^{n} \hat{e}\left(\prod_{j \in \mathcal{SV}} \sigma_i^{(j)}, h^{t_i - z_i^{(k)}}\right). \end{aligned} \tag{26}$$

Here, the aggregation of the function public keys, denoted as $\mathsf{FK}(\psi)$, can be optimally computed through

$$\begin{aligned} \mathsf{FK}\left(\psi\right) &= \prod_{i=1}^{n} \left({g_1}^{t_i}\right)^{\sum_{j \in \mathcal{SV}} x_i^{(j)}} = \prod_{j \in \mathcal{SV}} \prod_{i=1}^{n} \left({g_1}^{t_i}\right)^{x_i^{(j)}} \\ &= \prod_{j \in \mathcal{SV}} {g_1}^{\sum_{i=1}^{n} x_i^{(j)} t_i} = \prod_{j \in \mathcal{SV}} \mathsf{FK}\left(\psi_j\right). \end{aligned} \tag{27}$$

Additionally, the signatures can also be aggregated into σ_i:

$$\begin{aligned} \sigma_i &= {g_1}^{d \sum_{j \in \mathcal{SV}} x_i^{(j)}} \left({g_1}^{t_i}\right)^{d|\mathcal{SV}|} \\ &= \prod_{j \in \mathcal{SV}} \left({g_1}^{x_i^{(j)} + t_i}\right)^{d} = \prod_{j \in \mathcal{SV}} \sigma_i^{(j)}. \end{aligned} \tag{28}$$

Furthermore, the helper value $\mathsf{H}(\psi_j^{(k)})$ in the individual verification is independent of the support vector and thus can be aggregated by taking exponent to the power $|\mathcal{SV}|$.

We continue to consider how to efficiently conduct outcome verification for ϕ test data. From Equation (26), we can observe that σ_0 on the LHS and σ_i on the RHS are both independent of the test data $\mathbf{z}_k$. Hence, we can apply bilinearity again to Equation (26) for all $k \in [\phi]$ and derive

$$\begin{aligned} & \hat{e}\left(\left(\mathsf{FK}\left(\psi\right)\right)^{\phi} {g_1}^{-\sum_{k=1}^{\phi} \sum_{j \in \mathcal{SV}} v_j^{(k)}} \prod_{k=1}^{\phi} \mathsf{H}\left(\psi_j^{(k)}\right)^{|\mathcal{SV}|}, \sigma_0\right) \\ \overset{?}{=} \quad & \prod_{i=1}^{n} \hat{e}\left(\sigma_i, \left(h^{t_i}\right)^{\phi} h^{-\sum_{k=1}^{\phi} z_i^{(k)}}\right) \end{aligned} \tag{29}$$

as the batch verification formula at the test data level. Here, the aggregate function public key $\mathsf{FK}(\psi)$ is independent of test data, so we can raise ϕ to the exponent. In addition, we can aggregate the helper values more efficiently through

$$\begin{aligned} \prod_{k=1}^{\phi} \mathsf{H}\left(\psi_j^{(k)}\right)^{|\mathcal{SV}|} &= \left({g_1}^{\phi \sum_{i=1}^{n} {t_i}^2} \left(\prod_{i=1}^{n} \left({g_1}^{t_i}\right)^{\sum_{k=1}^{\phi} z_i^{(k)}}\right)^{-1}\right)^{|\mathcal{SV}|} \\ &= \prod_{k=1}^{\phi} \left({g_1}^{\sum_{i=1}^{n} t_i \left(t_i - z_i^{(k)}\right)}\right)^{|\mathcal{SV}|}. \end{aligned} \tag{30}$$

Design for RBF Kernel: We also start from single test data $\mathbf{z}_k$. Intuitively, the customer can perform $|\mathcal{SV}|$ instances of the algorithm Verify in the squared Euclidean distance protocol. In particular, we can assign values $x_i^{(j)}, z_i^{(k)}, r_i^{1(k)}$ to the variables x_i, z_i, r_i^1 in Equation (11), Equation (15), and Equation (16), thus obtaining the function public key $\mathsf{FK}(\psi_j)$, the signature $\sigma_i^{(j)(k)}$, and the helper value $\mathsf{H}(\psi_j^{(k)})$, respectively. In addition, we can similarly elicit the batch verification formula at the support vector level for the RBF kernel, which is the same as Equation (26) for the polynomial kernel. The differences lie in the concrete expressions of the aggregate function public key $\mathsf{FK}(\psi)$, the aggregate signature $\sigma_i^{(k)}$, and the helper value $\mathsf{H}(\psi_j^{(k)})$. Specifically, the aggregate function public key $\mathsf{FK}(\psi)$ here is given as

$$\begin{aligned}\mathsf{FK}(\psi) &= \prod_{i=1}^{n}\left({g_1}^{\sum_{j\in\mathcal{SV}} x_i^{(j)2}}\left({g_1}^{t_i}\right)^{-2\sum_{j\in\mathcal{SV}} x_i^{(j)}}\left({g_1}^{{t_i}^2}\right)^{|\mathcal{SV}|}\right)\\ &= \prod_{j\in\mathcal{SV}} {g_1}^{\sum_{i=1}^{n}\left(x_i^{(j)}-t_i\right)^2} = \prod_{j\in\mathcal{SV}} \mathsf{FK}(\psi_j),\end{aligned} \tag{31}$$

and the aggregate signature $\sigma_i^{(k)}$ is computed through

$$\begin{aligned}\sigma_i^{(k)} &= \left({g_1}^{t_i}\left({g_1}^{z_i^{(k)}}g^{r_i^1}\right)\right)^{d|\mathcal{SV}|}{g_1}^{-2d\sum_{j\in\mathcal{SV}} x_i^{(j)}}\\ = \prod_{j\in\mathcal{SV}}&\left({g_1}^{t_i}{g_1}^{-2x_i^{(j)}}\left({g_1}^{z_i^{(k)}}g^{r_i^1}\right)\right)^{d} = \prod_{j\in\mathcal{SV}} \sigma_i^{(j)(k)}.\end{aligned} \tag{32}$$

For the RBF kernel, its batch verification scheme at the test data level is much more complicated. The reason is that both two terms on the RHS of the batch verification formula at the support vector level, namely $\sigma_i^{(k)}$ and $h^{t_i-z_i^{(k)}}$, are related with the test data $\mathbf{z}_k$. We thus cannot directly apply bilinearity. Nevertheless, we can strip the part involving the test data (i.e., the ciphertext of $z_i^{(k)}$) from $\sigma_i^{(k)}$ and then perform aggregation. In the later verification phase, the customer, knowing its test data, can make up for this part. The aggregate signature $\sigma_i^{(k)}$ is now changed into

$$\sigma_i = \left({g_1}^{t_i}\right)^{d|\mathcal{SV}|}{g_1}^{-2d\sum_{j\in\mathcal{SV}} x_i^{(j)}} = \prod_{j\in\mathcal{SV}}\left({g_1}^{t_i-2x_i^{(j)}}\right)^{d} \tag{33}$$

and is independent of the test data $\mathbf{z}_k$. Moreover, the batch verification formula at the test data level is given as

$$\begin{aligned}&\hat{e}\left(\left(\mathsf{FK}\left(\psi\right)\right)^{\phi}{g_1}^{-\sum_{k=1}^{\phi}\sum_{j\in\mathcal{SV}} v_j^{(k)}}, \sigma_0\right)\\ \overset{?}{=}\;&\underbrace{\prod_{k=1}^{\phi}\prod_{i=1}^{n}\hat{e}\left(\sigma_i, h^{t_i-z_i^{(k)}}\right)}_{\text{RHS1}}\underbrace{\prod_{k=1}^{\phi}\prod_{i=1}^{n}\hat{e}\left({g_1}^{d|\mathcal{SV}|z_i^{(k)}}, h^{t_i-z_i^{(k)}}\right)}_{\text{RHS2}},\end{aligned}$$

where RHS1 and RHS2 can be more efficiently computed from the public parameters and the test data through

$$\prod_{i=1}^{n}\hat{e}\left(\sigma_i, \left(h^{t_i}\right)^{\phi}h^{-\sum_{k=1}^{\phi} z_i^{(k)}}\right), \tag{34}$$

$$\hat{e}\left({g_1}^{-\sum_{i=1}^{n}\sum_{k=1}^{\phi} z_i^{(k)2}}\prod_{i=1}^{n}\left({g_1}^{t_i}\right)^{\sum_{k=1}^{\phi} z_i^{(k)}}, {\sigma_0}^{|\mathcal{SV}|}\right), \tag{35}$$

respectively. Here, RHS2 is added to make up for the part split from the original signature involving $z_i^{(k)}$. This design is another feasible way to handle the problem of encoding $w_i(\mathbf{t})$ in the exponent without knowing its element $z_i^{(k)}$.

6.1.3 Performance Analysis

We analyze the time and communication complexities of oblivious evaluation and outcome verification in MVP.

For the oblivious evaluation of ϕ test data under both polynomial and RBF kernels, the time complexities of the customer, the service provider, and the model manager are $O(\phi n)T_{exp}$, $O(\phi|\mathcal{SV}|n)T_{exp}$, and $O(\phi|\mathcal{SV}|)T_{exp}$, respectively, where $|\mathcal{SV}|$ denotes the number of support vectors. Their communication complexities are $O(\phi n + \phi|\mathcal{SV}|)$, $O(\phi n + \phi|\mathcal{SV}|)$, and $O(\phi|\mathcal{SV}|)$, respectively.

For the batch outcome verification of ϕ test data, we start with the polynomial kernel. The time complexities of the customer, the service provider, and the model manager are $nT_{pair} + O(n)T_{exp}$, $1T_{pair} + O(n)T_{exp}$, and $O(n)T_{exp}$, respectively. Their communication complexities are $O(n)$, $O(n)$, and $O(1)$, respectively. We next analyze the RBF kernel. The time complexities of the customer, the service provider, and the model manager are $nT_{pair} + O(n)T_{exp}$, $2T_{pair} + O(n)T_{exp}$, and $O(n)T_{exp}$. Their communication complexities are $O(n)$, $O(n)$, and $O(1)$, respectively. From the above analysis, we can find that the batch outcome verification scheme in MVP can sharply reduce the verification latency, especially when supporting a large-scale test set. The reason is that $(n+1)$ and $(n+2)$ pairing operations dominate the overall verification overhead in the polynomial kernel and the RBF kernel, respectively. Also, the verification latency is related with the number of features n, but almost independent of the number of support vectors $|\mathcal{SV}|$ and the number of test data ϕ, which implies that our batch verification scheme is quite scalable.

6.1.4 Security Analysis

We can base the security of MVP on the security of two building blocks. First, because we have proven the signature's unforgeability in Proposition 1, our batch outcome verification scheme can be analogous to the classical batch verification of signatures [42]. Second, MVP can guarantee function privacy by thwarting the equation-solving attack against the support vectors through the intermediate results. In particular, the service provider can randomly permute the support vectors and thus shuffle the intermediate results. Such a construction prevents the PPT customer from recovering the support vectors by querying any number of test data, since it needs to try for solving $(\prod_{j=1}^{|\mathcal{SV}|} j)^{n-1}$ systems of equations, where each system contains n equations and n variables. Most importantly, this approach does not affect batch outcome verification. We defer the detailed analysis and proof to Appendix C. Third, all the test data of the customer are encrypted using the adapted BGN cryptosystem during the whole prediction process, and thus the input privacy is guaranteed, as formalized in Proposition 3.

6.2 Supporting Other ML Algorithms

We demonstrate the generality of MVP in terms of its application scope and extensibility.

Besides SVMs, MVP can support many other ML algorithms, only if their hypotheses can be decomposed into or approximated by two basic operations, i.e., dot product and Euclidean distance. Typical examples are linear models for regression and classification, Euclidean distance based prototype methods and nearest neighbors, and neural networks. In particular, the former two kinds of ML models can be handled by the dot product protocol and the squared Euclidean distance protocol, respectively. We take one ML primitive, called logistic regression, for example. The hypothesis of logistic regression is $f\left(\mathbf{z}_k\right) = \varphi\left(\mathbf{w}^T\mathbf{z}_k + \mathbf{b}\right)$, where $\varphi(v) = \frac{1}{1+\exp(-v)}$ is the logistic function, $\mathbf{w}$ is the weight vector, and $\mathbf{b}$ is the bias vector. Since $\varphi(\cdot)$ is public and can be computed by the customer herself, the service provider, holding the confidential model parameters $\mathbf{w}$ and

$\mathbf{b}$, just needs to return $v = \mathbf{w}^T\mathbf{z}_k + \mathbf{b}$. Therefore, logistic regression can be handled by our dot product protocol. In addition, given the fact that $\mathbf{w}$ and $\mathbf{b}$ remain unchanged for each test data $\mathbf{z}_k$, the batch outcome verification scheme can also be constructed.

We next expand on neural networks from computational feasibility and efficiency. We first consider computational feasibility. As illustrated in [7], [9], [10], [15], [40], neural networks in the prediction phase just use one iteration of the feedforward component, which consists of the following common functions: weighted sum (fully connected layers and convolution layers), max pooling, mean pooling, sigmoid, and rectified linear. First, the weighted sum function can be handled using the dot product protocol. Second, max pooling cannot be computed directly, since the max-function is non-polynomial. However, powers of it can be approximated due to the relation: $\max(z_1, z_2, \ldots, z_n) = \lim_{p\to\infty} \left(\sum_{i=1}^{n} z_i{}^p\right)^{\frac{1}{p}}$. When $p = 1$, max pooling degenerates to scaled mean pooling, whose feasibility has been validated by [15]. In addition, the scaled mean pooling can be handled with our dot product protocol. Moreover, MVP also supports another case of $p = 2$ with the squared Euclidean distance protocol. Third, the mathematical formula of mean pooling is sum dividing by a constant and can be processed with the dot product protocol. Fourth, as suggested by [2], [7], [15], [43], we can replace the non-polynomial sigmoid and rectified linear activation functions with square activation function. Meanwhile, the square activation function can be handled by our squared Euclidean distance protocol. We then consider computational efficiency. First, for the service provider, since it knows the weight matrix and bias vector, it can still use the efficient homomorphic properties for oblivious evaluation, including multiplication by a constant and homomorphic addition. Additionally, analogous to SVMs with the liner kernel, consecutive layers in neural networks that use only linear transformations (e.g., weighted sum and mean pooling) can be collapsed. Hence, the network depth together with the evaluation overhead should be reduced. Second, for the customer, it can still perform batch outcome verification by applying the bilinearity of asymmetric pairing, since the parameters of the trained neural networks keep the same for each test data.

In a nutshell, MVP is not just limited to SVMs and can apply to many other ML algorithms. In addition, due to the generality of Lemma 1, we can extend the building blocks of MVP to multivariate polynomials with higher degrees, which should support a wider range of ML algorithms.

7 Evaluation Results

In this section, we apply MVP to Short Message Service (SMS) spam detection and present its evaluation results in terms of computation and communication overhead. We also demonstrate the feasibility of the model manager.

Datasets: We use three real-world SMS datasets: SMS Spam Collection v.1 [44], DIT SMS Spam Dataset [45], and NUS SMS Corpus [46]. In particular, SMS Spam Collection v.1 is the first benchmark SMS dataset and contains 747 spam messages and 4,825 ham (legitimate) messages. The DIT dataset is composed of 1,353 unique spam messages extracted from two UK public consumer complaints websites, called GrumbleText and WhoCallsMe, and was gathered over the period from late 2003 to the middle of 2010. The NUS dataset is a corpus of 55,835 legitimate messages in Singapore English, and its most recent release was on March 9, 2015. In this evaluation, we randomly select 80% of SMS Spam Collection v.1 as the training set and regard the complement 20% as the default test set. We can enlarge the size of test set by adding part of the other datasets.

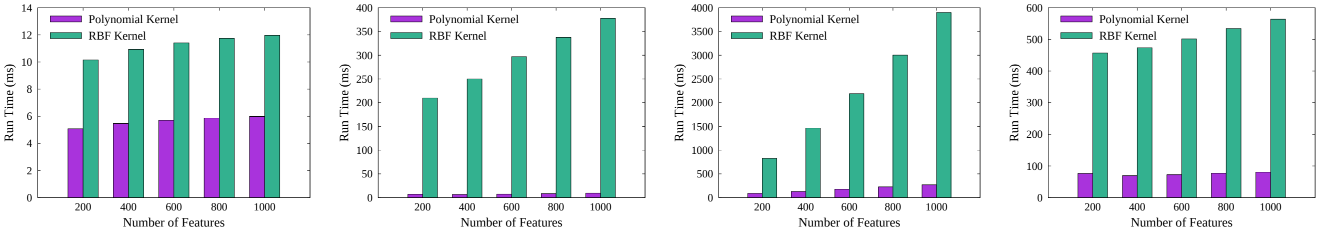
TABLE 2
Characteristics of SMS Datasets.

n	Θ(TS)	$\lvert\mathcal{SV}\rvert$		Θ(SVs)		Θ(IRs)	
		Poly	RBF	Poly	RBF	Poly	RBF
200	1.60%	561	570	1.73%	1.75%	8.83%	99.75%
400	1.04%	643	576	1.10%	1.10%	8.86%	99.95%
600	0.80%	745	611	0.88%	0.85%	9.03%	99.96%
800	0.65%	852	652	0.74%	0.69%	9.05%	99.97%
1000	0.55%	935	694	0.65%	0.60%	9.11%	99.98%

Preprocessing: We use some common text processing techniques. We first remove punctuation and stop words, then convert the text to lowercase, and finally build Term Frequency-Inverse Document Frequency (TF-IDF) based feature vectors. We feed the feature vectors into SVMs and thus learn optimal classifiers. Our implementation employs a popular ML library in Python, called scikit-learn v0.19.0 [47]. Specifically, we set the parameter *max_features* in the function *TfidfVectorizer* to control the size of vocabulary/features n and employ the function *GridSearchCV* to tune the hyper-parameters of two kernels. The best test accuracies reach 98.39% and 98.21% in the polynomial kernel with the degree of 3 and the RBF kernel, respectively.

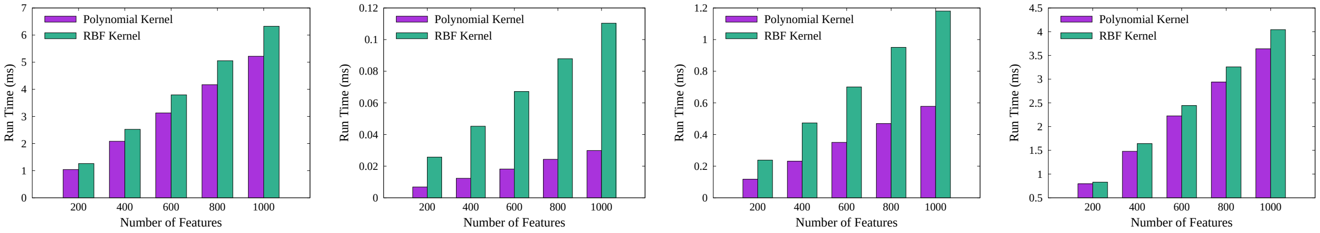
Data Characteristics: Table 2 presents the numbers of features n and corresponding support vectors $|\mathcal{SV}|$, along with the densities of test set Θ(TS), support vectors Θ(SVs), and intermediate results Θ(IRs). Here, $\Theta(\cdot)$ denotes the density of a matrix, which is defined as the number of nonzero elements divided by the size of the matrix. Thus, $n \times \Theta$(TS) and $n \times \Theta$(SVs) can generally capture the average numbers of nonzero values in one short message's feature vector for the test set and the support vectors, respectively. Similarly, $|\mathcal{SV}| \times \Theta$(IRs) can indicate the average number of nonzero elements within each test data's intermediate results (i.e., $\{v_j^{(k)} = \psi_j(\mathbf{z}_k) | j \in \mathcal{SV}\}$). Therefore, the test set, the support vectors, and the polynomial kernel's intermediate results are quite sparse, whereas the RBF kernel's intermediate results are relatively dense. These characteristics of SMS datasets should be considered in our implementation and analysis.

Configurations: We implemented MVP using the latest Pairing-Based Cryptography (PBC) library [48]. The elliptic curve in our implementation is a MNT curve with a base field size of 172 bits and an embedding degree of 6. In addition, the group order q is 163-bit long, and the fundamental discriminant is 3,447,443. Our curve choice is sufficient to defeat both generic discrete logarithm attacks and finite field discrete logarithm attacks [48]. The running environment is a standard 64-bit Ubuntu 14.04 Linux operation system on a desktop with Intel(R) Core(TM) $i5$ $3.10GHz$.



(a) Encryption (b) Evaluation (Sparse) (c) Evaluation (Dense) (d) Decryption

Fig. 2. Computation Overhead of Oblivious Evaluation in MVP Per Test Data.



(a) Signature Generation Set (b) Function Public Key (c) Signature of Correctness (d) Verification

Fig. 3. Computation Overhead of Outcome Verification in MVP Per Test Data.

7.1 Computation Overhead

We show the computation overhead of oblivious evaluation and outcome verification in MVP.

7.1.1 Oblivious Evaluation

The oblivious evaluation can be further divided into three phases, including encryption, evaluation, and decryption. Fig. 2 plots the time overhead of three phases per test data, when the number of features n increases from 200 to 1000 with a step of 200.

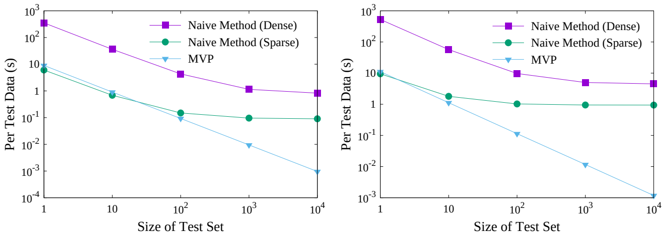
Encryption: Considering the sparsity of test data, we let the customer encrypt all the nonzero values and just randomly encrypt a small number of those zero elements. Here, the padded zero encryptions are to perturb the vocabulary of each short message, thereby strengthening the input privacy against the service provider. In this set of simulations, we set the number of padded zero encryptions for each test data to be 10, which is larger than both $n \times \Theta(\mathrm{TS})$ and $n \times \Theta(\mathrm{SVs})$ in Table 2. Fig. 2a plots the encryption overhead per test data. We can see from Fig. 2a that the encryption costs in two kernels both increase with n. We explain the reason through $n \times \Theta(\mathrm{TS})$, which indicates the average number of nonzero values per test data. This value increases with n and thus the total number of BGN encryptions. We can also observe from Fig. 2a that the encryption overhead in the RBF kernel is roughly twice as much as that in the polynomial kernel. This is because the RBF kernel requires an extra encryption of each test data's square. When n reaches 1000, the encryption overhead at the customer is only 5.98 ms and 11.98 ms per test data in the polynomial kernel and the RBF kernel, respectively.

Evaluation: We next investigate the evaluation overhead under two different encryption strategies: one is to pad 10 zero encryptions, and the other is to encrypt all the zero elements. The former strategy can embody the efficiency of MVP when dealing with sparse data, whereas the latter is able to validate its feasibility over dense data. We show the evaluation results in Fig. 2b and Fig. 2c, respectively.

The first key observation from Fig. 2b and Fig. 2c is that the evaluation overhead in the RBF kernel is much higher than that in the polynomial kernel. In terms of their corresponding building blocks, two major reasons account for this result. One reason is that when dealing with two vectors, the basic oblivious operation in the squared Euclidean distance is much more time-consuming than that in the dot product. The other reason lies in data sparsity and padded zero encryptions. For dot product, the service provider omits a pair, if either of its elements is zero. In contrast, for squared Euclidean distance, only when both of the elements are zero, the pair can be elided. Furthermore, the padded zero encryptions make this situation worse. For example, we assume that a support vector is $\mathbf{x}_j = (1, 0, 1, 0)$ and an encrypted test data is $\tilde{\mathbf{z}}_k = (0, \tilde{1}, 0, 0)$. The service provider performs 0 and 3 basic operations for the dot product and the squared Euclidean distance, respectively. If the customer inserts a zero encryption in the first place of $\tilde{\mathbf{z}}_k$, i.e., $\tilde{\mathbf{z}}_k = (\tilde{0}, \tilde{1}, 0, 0)$, it has no effect on the dot product but incurs one extra operation for the squared Euclidean distance. The second key observation is derived by comparing Fig. 2b with Fig. 2c. When the test data are encrypted in a denser way, the evaluation overhead in two kernels both increases. However, the growth trend is much smaller than that of density, which can be computed roughly through $n/(n \times \Theta(\mathrm{TS}) + 10)$. In particular, when $n = 1000$, the density increases $64.50\times$, while the evaluation overhead just grows $28.50\times$ and $10.33\times$, reaching 0.27 s and 3.90 s in the polynomial kernel and the RBF kernel, respectively.

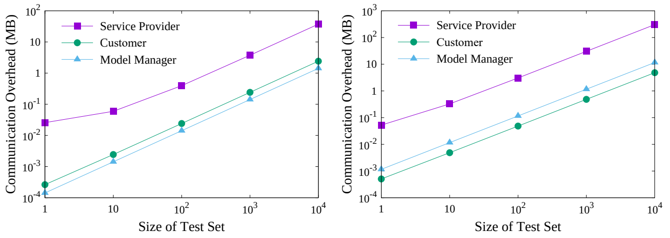
From the above results and analysis, we can find that MVP performs well over both sparse and dense data. A lower growth factor of the evaluation overhead shows MVP's extensibility at the service provider.

Decryption: In Fig. 2d, we plot the decryption overhead



(a) Polynomial Kernel (b) RBF Kernel

Fig. 4. MVP vs. Naïve Method (Total Outcome Verification Overhead)



(a) Polynomial Kernel (b) RBF Kernel

Fig. 5. Communication Overhead.

under the sparse encryption strategy. From Fig. 2d, we can see that the trend of decryption overhead is generally consistent with that of $|\mathcal{SV}| \times \Theta(\text{IRs})$ in Table 2. Here, we implement the algorithm Decrypt by pre-computing a polynomial-size table of the powers of the base $\pi(g_1, g_2)$, and the latency of taking logarithm once is in microsecond. The dominant operation in Decrypt now is to compute the projection on ciphertext, whose one-time overhead is 0.63 ms and 0.81 ms in the polynomial kernel and the RBF kernel, respectively. Furthermore, the decryption overhead scales up far less than the evaluation overhead under the dense encryption strategy, e.g., when $n = 1000$, the RBF kernel incurs additional 1.32% of the decryption overhead.

Overall Oblivious Evaluation: Due to the sparsity of feature vectors in SMS spam detection, the ratio of support vectors is very large, e.g., when $n = 1000$, the support vectors in the polynomial kernel occupy 20.98% of the training set. This is the worst case of MVP's decryption phase. Nevertheless, the maximum total overhead of oblivious evaluation occurs in the RBF kernel and consumes 0.95 s per test data with 1000 features, which can embody the practical efficiency of MVP's oblivious evaluation.

7.1.2 Outcome Verification

The batch outcome verification in MVP contains four major components: the aggregate generations of signature generation set, function public key, and signature of correctness in the preparation phase, as well as the final verification phase. Fig. 3 plots the evaluation results per test data when the number of features n varies from 200 to 1000. In addition, the size of test set ϕ is fixed at 1000.

As depicted in Fig. 3, the computation overhead of these four components all grows linearly with n. We explain reasons one by one. First, the cardinalities of the signature generation sets are $(2n+1)$ and $4n$ in the polynomial kernel and the RBF kernel, respectively. Second, the formulas of function public keys (i.e., Equation (27) and Equation (31)) imply that their generation overhead increases linearly with n. Third, the signatures of correctness (i.e., Equation (28) and Equation (33)) are with the size of n. Fourth, the verification phases are dominated by $(n + 1)$ and $(n + 2)$ pairing operations in the polynomial kernel and the RBF kernel, respectively. Last, when $n = 1000$, the total outcome verification overhead per test data is 9.47 ms and 11.66 ms in the polynomial kernel and the RBF kernel, respectively.

We note that the outcome verification scheme in MVP just depends on the parameters of a trained model and is independent of data sparsity/density. This implies that with a larger size of test set ϕ, the amortized verification overhead can be further reduced. To validate this point and to get a full view of MVP's verification performance, we compare with the naïve method claimed in Section 4. Fig. 4 details the comparisons when the size of test set ϕ increases from 1 to 10,000 by exponential growth. Here, we fix the number of features n at 1000. In addition, the sparse and dense encryption strategies applied to the support vectors in the naïve method are analogous to those applied to the test data. From Fig. 4, we can observe that under the sparse encryption strategy, MVP outperforms the naïve method when $\phi \geq 100$ in the polynomial kernel and when $\phi \geq 10$ in the RBF kernel. As ϕ becomes larger, MVP's advantage over the naïve method is more remarkable, e.g., when $\phi = 1000$, the total outcome verification overhead of MVP is 9.86% and 1.23% of that of the naïve method in the polynomial kernel and the RBF kernel, respectively. Regarding under the dense encryption strategy, MVP is far better than the naïve method in all sizes of test set. Even when $\phi = 1$, the total outcome verification overhead of MVP is 2.54% and 2.11% of that of the naïve method in the polynomial kernel and the RBF kernel, respectively.

In conclusion, MVP can indeed reduce the outcome verification overhead when supporting a large-scale test set.

7.2 Communication Overhead

Fig. 5 plots the communication overhead of the customer, the service provider, and the model manager, where the number of features n is fixed at 1000. In addition, the customer just transmits the ciphertexts of those nonzero elements. Moreover, the communication overhead here only counts in the amount of sending content.

We can see from Fig. 5 that the communication overhead of three entities grows linearly with the size of test set ϕ. The reason is that the customer mainly needs to submit the encryptions of ϕ test data for classification. In addition, the service provider primarily requires to send an average number of $\phi \times |\mathcal{SV}| \times \Theta(\text{IRs})$ encrypted intermediate results for decryption and to deliver the signature of correctness for batch outcome verification. Regarding the model manager, its communication overhead mainly comes from returning the decrypted intermediate results. We note that the x, y axes in Fig. 5 are log-scaled, and thus the communication overhead of the service provider, containing a constant transmission overhead of the signature of correctness, seems non-linear. Specifically, when $\phi < 10$, the signature of correctness transmission overhead dominates, and this interval looks a little flat. Last, when $\phi = 10000$, the bandwidth overhead of the customer is 2.41 MB and 4.83 MB in the polynomial kernel and the RBF kernel, respectively.

In summary, MVP can dramatically reduce the communication overhead, by generating the signature of correctness in an aggregate mode and by using the bilinear groups with a much smaller prime order.

7.3 Overhead of Model Manager

We finally report the computation, communication, and storage overhead of the model manager. We take SVMs with the most accurate polynomial kernel. We let the customer just encrypt those nonzero elements. We fix the number of features n at 1000 and set the size of test set ϕ to be 10000. First, the primary duty of the model manager is to set up the parameters for the adapted BGN cryptosystem and the outcome verification scheme. Also, it needs to perform roughly $\phi \times |\mathcal{SV}| \times \Theta(\text{IRs})$ decryptions in total. The computation overhead of these two parts is 57.50 s. Further, the one-time initialization overhead can be amortized over a larger size of test set. Second, the communication overhead of the model manager is 1.42 MB. Third, the storage overhead mostly comes from maintaining the table of powers of $\pi(g_1, g_2)$, which takes up 0.22 MB storage space. In a nutshell, the model manager has a light load.

8 Related Work

In this section, we briefly review related work.

8.1 Verifiable Polynomial Computation

Cloud computing contributes to the emergence of verifiable computation, where a server computes an outsourced function on behalf of a client over its input and then generates a proof of correct computation. Most of theoretical work focuses on specific functionalities (e.g., multivariate polynomials in this work) rather than generic constructions. Kate et al. [49] proposed a publicly verifiable commitment scheme for univariate polynomials. When extended to the multivariate case, Papamanthou et al. [38] designed a signature of correct computation scheme that supports incremental updates of polynomial coefficients[3]. For the same polynomial with different inputs, Catalano et al. [50] improved verification efficiency in an amortized sense. Fiore et al. [51] further considered verifiable computation of multivariate quadratic polynomials on encrypted data.

8.2 Secure ML Prediction

An explosive demand of ML prediction services attracts increasing attention to their security. One line of research focuses on privacy preservation, namely, protecting the customer's sensitive test data and/or the service provider's proprietary trained model while maintaining the functionality of prediction. Existing work mainly used secure two-party or multi-party computation (MPC) techniques, such as homomorphic encryption, garbled circuit, secret sharing, and oblivious transfer. Bost et al. [14] considered multiple classifiers, including linear classifiers, naïve Bayes, and decision trees. They further presented a library of reusable building blocks underlying common classifiers (e.g., dot product and argmax). In [15], Gilad-Bachrach et al. proposed CryptoNets, which can apply neural networks to encrypted test data with high throughput and accuracy. The following-up work [7], [9], [10], [16], [52] further reduced response latency and message size mainly by introducing offline precomputation/planning and optimizing online operations. Kumar et al. [53] developed CryptFlow, converting Tensorflow inference code to MPC protocols.

In addition to privacy preservation, ML prediction services also need verifiability. Using the terminologies of verifiable computation, the customer works as a client, the service provider works as a server, and the trained model works as a function. The key difference from the outsourced scenario is that because of function privacy, the customer as a verifier cannot learn the detailed model parameters, kept by the service provider as a prover, throughout the whole verification process. Meanwhile, because of input privacy, the service provider also cannot know the test data of the customer. The joint requirement of verifiablility and privacy preservation in the ML prediction context makes conventional verification computation schemes inapplicable and calls for new practical designs.

Ghodsi et al. [2] designed an interactive proof protocol, called SafetyNets, for verifiable execution of neural networks prediction. Lee et al. [54] proposed vCNN using a pairing-based zero-knowledge succinct non-interactive argument of knowledge (SNARK) to guarantee verifiability for convolutional neural networks prediction. Both SafetyNets and vCNN ignore input privacy and function privacy. Zhao et al. [55] considered a different setting from ours, where the customer outsources ML training and prediction tasks (including the training data and the test data) to the service provider. They proposed VeriML, which is based on SNARK and blockchain. VeriML ensures the integrity of a trained model and prediction results as well as the fair payments before returning the model and the results. However, VeriML does not guarantee the customer's input privacy. Xu et al. [56] investigated an emerging federated learning framework, where a server aggregates (basically, sums up) the model updates from multiple clients, and proposed VerifyNet. VerifyNet not only guarantees the privacy of the model updates with the celebrated secure aggregation protocol in [57] but also ensures the correctness of the aggregated result with homomorphic hash function. In essence, VerifyNet is a block of verifiable and privacy preserving summation, namely, a special case of verifiable and privacy preserving dot product by setting the weight vector to a public all-ones vector.

Parallel to the above cryptographic schemes, Tramèr and Boneh [4] assumed the existence of a hardware-based Trusted Execution Environment (TEE, e.g., Intel SGX, ARM TrustZone, and Sanctum) on the service provider and viewed the TEE as a trusted authority in security analysis. The TEE not only isolates and protects the program of neural networks prediction in an enclave from all the other programs/adversaries outside the enclave, including OS, but also allows the customer's remote attestation to correct program execution by establishing a secure channel with the

[3] From Fig. 3b, the maximum overhead of function public key generation is 0.11 ms per test data. Thus, it is reasonable to compute the function public key from scratch under a new trained model.

TEE. Nevertheless, the memory of the TEE is fully encrypted and authenticated and is very limited in size (e.g., 128 MB for SGX), while the operations within the TEE are expensive and are difficult to be parallelized. Tramèr and Boneh thus proposed a framework, called Slalom, to outsource the execution of all linear layers in neural networks (i.e., the operations of dot product) from the TEE to an untrusted, faster, and co-located processor. On the one hand, Slalom guarantees input privacy with additive stream cipher but requires the TEE to offline precompute blinding terms, the overhead of which is the same as that of directly evaluating the linear layers in the TEE. On the other hand, Slalom achieves verifiability with Freivalds' algorithm. However, Slalom does not guarantee model privacy. In addition, the outsourcing process incurs extremely high communication overhead between the TEE and the untrusted processor (e.g., over 50 MB per inference for VGG16).

8.3 Adversarial ML and Defenses

In the field of adversarial ML, many research work explores practical attacks against ML models and/or underlying training data, the defenses for which are complementary to the aforementioned security mechanisms. Typical attacks launched via the prediction APIs (i.e., with a black-box access) include model extraction [8], [58], [59], which aims to duplicate the functionality of the trained model, and model evasion [60]–[63], which seeks to develop strategies for avoiding detection (e.g., in spam identification, face recognition, malware classification, and network anomaly detection). In general, the model extraction attack is inevitable, while the root of the model evasion attack lies in the lack of interpretability and transparency in ML training. Effective mitigation methods for the model extraction attack include limiting the number and the output information of predictions [7], as well as detecting adversarial customers by analyzing query patterns [64]. Major countermeasures for the model evasion attack include designing robust training algorithms [65] and detecting adversarial samples [66].

There still exist some other common attacks with a white-box access or in the training phase. One is model inversion [67]–[69], which is launched to infer the contents or memberships of training data from the passively extracted or actively released model parameters. A potential countermeasure for the model inversion attack is perturbing the model parameters with random noise to achieve a strict differential privacy guarantee, such as [17]–[22]. Another is data poisoning [70]–[72] (including the dominant backdoor), which is launched in the ML training phase and shares a similar goal to the model evasion attack in the prediction phase. Specifically, the data poisoning attack intends to manipulate the trained model by injecting poisoning samples to the training set. Existing defenses for the data poisoning attack include training with a trimmed loss function for regression models [72], as well as analyzing inner neural behaviors through stimulation [73] and applying outlier detection [74] for neural networks. Please refer to the survey [75] for more defenses.

8.4 Key Differences of MVP from Existing Work

From the review about related work above, we can derive that the proposed MVP differs from existing work mainly in that: (1) from application scenario, MVP focuses on the ML prediction phase, where a customer queries the prediction API held by a service provider. In contrast, some existing work considered other different (e.g., outsourced) ML prediction scenarios, the ML training phase, or collaborative ML, which is parallel to MVP; (2) from security guarantees, MVP focuses on how to achieve outcome verifiability, function privacy, and input privacy simultaneously, whereas existing work just considered part of these three properties; and (3) from technical novelty, MVP relies on polynomial decomposition, prime-order bilinear groups, and an adapted BGN homomorphic encryption. MVP further supports batch outcome verification and signature aggregation by applying the bilinear properties, dramatically improving efficiency. To the best of our knowledge, MVP, is the first to leverage these cryptographic and non-cryptographic primitives to achieve verifiability and privacy preservation. Also, MVP does not rely on any trusted hardware, which instead was used in some previous work.

9 Conclusion and Future Work

In this paper, we have proposed the first secure scheme MVP for ML prediction services, simultaneously guaranteeing outcome verifiability, function privacy, and input privacy. In MVP, the service provider has to honestly process the test data, and the customer can verify the integrity of returned outcomes in batch mode. In addition, both the confidential model parameters and the sensitive test data are well protected. Furthermore, we have instantiated MVP with SVMs, and evaluated its performance on three practical SMS datasets. Evaluation results have demonstrated the effectiveness and efficiency of MVP.

Regarding future work, one interesting direction is to consider the ML training phase and further to explore the issues of verifiability and privacy preservation. For example, a data owner intends to train a ML model with the help of a service provider. The data owner wants to protect its training data and/or model. Meanwhile, the data owner also wants to efficiently verify the integrity of the whole training process. How the service provider can generate a proof without breaching privacy is an important problem. Another interesting direction is to further explore the decentralized setting of the model managers without the non-colluding requirement in Section 5.4. Potential techniques include secure multi-party computation and blockchain.

References

[1] "Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region," https://aws.amazon.com/cn/message/65648/, 2011.

[2] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Proc. of NeurIPS*, 2017, pp. 4672–4681.

[3] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *Proc. of EuroS&P*, 2018, pp. 399–414.

[4] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *Proc. of ICLR*, 2019.

[5] E. Angelini, G. di Tollo, and A. Roli, "A neural network approach for credit risk evaluation," *The quarterly review of economics and finance*, vol. 48, no. 4, pp. 733–755, 2008.

[6] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, pp. 115–118, 2017.

[7] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, “Delphi: A cryptographic inference service for neural networks,” in *Proc. of USENIX Security*, 2020.

[8] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction APIs,” in *Proc. of USENIX Security*, 2016, pp. 601–618.

[9] J. Liu, M. Juuti, Y. Lu, and N. Asokan, “Oblivious neural network predictions via MiniONN transformations,” in *Proc. of CCS*, 2017, pp. 619–631.

[10] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, “GAZELLE: A low latency framework for secure neural network inference,” in *Proc. of USENIX Security*, 2018, pp. 1651–1669.

[11] “2016 TRUSTe/NCSA Consumer Privacy Infographic - US Edition,” https://www.truste.com/resources/privacy-research/ncsa-consumer-privacy-index-us/.

[12] R. Gennaro, C. Gentry, and B. Parno, “Non-interactive verifiable computing: Outsourcing computation to untrusted workers,” in *Proc. of CRYPTO*, 2010, pp. 465–482.

[13] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno, “Hash first, argue later: Adaptive verifiable computations on outsourced data,” in *Proc. of CCS*, 2016, pp. 1304–1316.

[14] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data,” in *Proc. of NDSS*, 2015.

[15] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy,” in *Proc. of ICML*, 2016, pp. 201–210.

[16] H. Chen, W. Dai, M. Kim, and Y. Song, “Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference,” in *Proc. of CCS*, 2019, pp. 395–412.

[17] B. I. P. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft, “Learning in a large function space: Privacy-preserving mechanisms for SVM learning,” *Journal of Privacy and Confidentiality*, vol. 4, no. 1, pp. 65–100, 2012.

[18] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” in *Proc. of CCS*, 2016, pp. 308–318.

[19] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, “Learning differentially private recurrent language models,” in *Proc. of ICLR*, 2018.

[20] B. Jayaraman and D. Evans, “Evaluating differentially private machine learning in practice,” in *Proc. of USENIX Security*, 2019, pp. 1895–1912.

[21] L. Yu, L. Liu, C. Pu, M. E. Gursoy, and S. Truex, “Differentially private model publishing for deep learning,” in *Proc. of S&P*, 2019, pp. 332–349.

[22] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, “Privacy-preserving collaborative deep learning with unreliable participants,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1486–1500, 2020.

[23] C. Cortes and V. N. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[24] A. Menezes, S. A. Vanstone, and T. Okamoto, “Reducing elliptic curve logarithms to logarithms in a finite field,” in *Proc. of STOC*, 1991, pp. 80–89.

[25] D. M. Freeman, “Converting pairing-based cryptosystems from composite-order groups to prime-order groups,” in *Proc. of EUROCRYPT*, 2010, pp. 44–61.

[26] D. Boneh and X. Boyen, “Short signatures without random oracles and the SDH assumption in bilinear groups,” *Journal of Cryptology*, vol. 21, no. 2, pp. 149–177, 2008.

[27] D. Boneh, E. Goh, and K. Nissim, “Evaluating 2-dnf formulas on ciphertexts,” in *Proc. of TCC*, 2005, pp. 325–341.

[28] R. Cramer, R. Gennaro, and B. Schoenmakers, “A secure and optimally efficient multi-authority election scheme,” in *Proc. of EUROCRYPT*, 1997, pp. 103–118.

[29] B. Adida, “Helios: Web-based open-audit voting,” in *Proc. of USENIX Security*, 2008, pp. 335–348.

[30] M. R. Clarkson, S. Chong, and A. C. Myers, “Civitas: Toward a secure voting system,” in *Proc. of S&P*, 2008, pp. 354–368.

[31] S. Park, M. Specter, N. Narula, and R. L. Rivest, “Going from bad to worse: from internet voting to blockchain voting,” MIT, Tech. Rep., 2020. [Online]. Available: http://people.csail.mit.edu/rivest/pubs/PSNR20.pdf

[32] D. Naylor, M. K. Mukerjee, and P. Steenkiste, “Balancing accountability and privacy in the network,” in *Proc. of SIGCOMM*, 2014, pp. 75–86.

[33] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, “ADSNARK: nearly practical and privacy-preserving proofs on authenticated data,” in *Proc. of S*&P, 2015, pp. 271–286.

[34] “MLOps: model management, deployment, and monitoring with Azure Machine Learning,” https://docs.microsoft.com/en-us/azure/machine-learning/service/concept-model-management-and-deployment, last accessed on Dec. 4, 2019.

[35] “Managing models and jobs on Google AI Platform,” https://cloud.google.com/ml-engine/docs/managing-models-jobs, last accessed on Dec. 4, 2019.

[36] “Managing Amazon ML Objects,” https://docs.aws.amazon.com/machine-learning/latest/dg/managing_objects.html, last accessed on Dec. 4, 2019.

[37] Y. Lindell, “How to simulate it - A tutorial on the simulation proof technique,” in *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 2017, pp. 277–346.

[38] C. Papamanthou, E. Shi, and R. Tamassia, “Signatures of correct computation,” in *Proc. of TCC*, 2013, pp. 222–242.

[39] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, “Privacy-preserving ridge regression on hundreds of millions of records,” in *Proc. of S*&P, 2013, pp. 334–348.

[40] P. Mohassel and Y. Zhang, “Secureml: A system for scalable privacy-preserving machine learning,” in *Proc. of S&P*, 2017, pp. 19–38.

[41] T. P. Pedersen, “A threshold cryptosystem without a trusted party,” in *Proc. of EUROCRYPT*, 1991, pp. 522–526.

[42] J. Camenisch, S. Hohenberger, and M. Ø. Pedersen, “Batch verification of short signatures,” in *Proc. of EUROCRYPT*, 2007, pp. 246–263.

[43] R. Livni, S. Shalev-Shwartz, and O. Shamir, “On the computational efficiency of training neural networks,” in *Proc. of NeurIPS*, 2014, pp. 855–863.

[44] “SMS Spam Collection v. 1,” http://www.dt.fee.unicamp.br/\%7Etiago/smsspamcollection/, 2011.

[45] “DIT SMS Spam Dataset,” http://www.dit.ie/computing/research/resources/smsdata/, 2012.

[46] “NUS SMS Corpus,” https://github.com/kite1988/nus-sms-corpus, 2015.

[47] “Scikit-learn: Machine Learning in Python,” http://scikit-learn.org/.

[48] “PBC Library,” https://crypto.stanford.edu/pbc/.

[49] A. Kate, G. M. Zaverucha, and I. Goldberg, “Constant-size commitments to polynomials and their applications,” in *Proc. of ASIACRYPT*, 2010, pp. 177–194.

[50] D. Catalano, D. Fiore, and B. Warinschi, “Homomorphic signatures with efficient verification for polynomial functions,” in *Proc. of CRYPTO*, 2014, pp. 371–389.

[51] D. Fiore, R. Gennaro, and V. Pastro, “Efficiently verifiable computation on encrypted data,” in *Proc. of CCS*, 2014, pp. 844–855.

[52] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, “XONN: Xnor-based oblivious deep neural network inference,” in *Proc. of USENIX Security*, 2019, pp. 1501–1518.

[53] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, “CrypTFlow: Secure TensorFlow Inference,” in *Proc. of S&P*, 2020, pp. 1646–1663.

[54] S. Lee, H. Ko, J. Kim, and H. Oh, “vcnn: Verifiable convolutional neural network,” Cryptology ePrint Archive, Report 2020/584, Tech. Rep., 2020.

[55] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, X. Lin, S. Hu, and M. Du, “Veriml: Enabling integrity assurances and fair payments for machine learning as a service,” arxiv: 1909.06961, Tech. Rep., 2019.

[56] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, “Verifynet: Secure and verifiable federated learning,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2020.

[57] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. of CCS*, 2017, pp. 1175–1191.

[58] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, “High accuracy and high fidelity extraction of neural networks,” in *Proc. of USENIX Security*, 2020.

[59] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring connections between active learning and model extraction," in *Proc. of USENIX Security*, 2020.

[60] D. Lowd and C. Meek, "Adversarial learning," in *Proc. of KDD*, 2005, pp. 641–647.

[61] N. Srndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *Proc. of S&P*, 2014, pp. 197–211.

[62] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. of ECML PKDD*, 2013, pp. 387–402.

[63] K. T. Co, L. Muñoz-González, S. de Maupeou, and E. C. Lupu, "Procedural noise adversarial examples for black-box attacks on deep convolutional networks," in *Proc. of CCS*, 2019, pp. 275–289.

[64] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta, "Model extraction warning in mlaas paradigm," in *Proc. of ACSAC*, 2018, pp. 371–380.

[65] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. of S&P*, 2016, pp. 582–597.

[66] G. Tao, S. Ma, Y. Liu, and X. Zhang, "Attacks meet interpretability: Attribute-steered detection of adversarial samples," in *Proc. of NeurIPS*, 2018, pp. 7728–7739.

[67] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *Proc. of USENIX Security*, 2014, pp. 17–32.

[68] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *IJSN*, vol. 10, no. 3, pp. 137–150, 2015.

[69] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Proc. of S&P*, 2019, pp. 1021–1035.

[70] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. of ICML*, 2012.

[71] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," arXiv: 1712.05526, Tech. Rep., 2017.

[72] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *Proc. of S&P*, 2018, pp. 19–35.

[73] Y. Liu, W. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "ABS: scanning neural networks for back-doors by artificial brain stimulation," in *Proc. of CCS*, 2019, pp. 1265–1282.

[74] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. of S&P*, 2019, pp. 707–723.

[75] Y. Chen, X. Gong, Q. Wang, X. Di, and H. Huang, "Backdoor attacks and defenses for deep neural networks in outsourced cloud environments," *IEEE Network*, pp. 1–7, 2020.

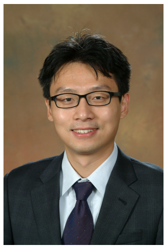
[76] E. J. Chou, A. Gururajan, K. Laine, N. K. Goel, A. Bertiger, and J. W. Stokes, "Privacy-preserving phishing web page classification via fully homomorphic encryption," in *Proc. of ICASSP*, 2020, pp. 2792–2796.

[77] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *Proc. of USENIX Security*, 2017, pp. 1041–1056.

[78] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. of S&P*, 2019, pp. 1–19.

Chaoyue Niu is working toward the PhD degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, P. R. China. His research interests include privacy preservation and verifiable computation in data management. He is a student member of the ACM and IEEE.

Fan Wu is a professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He received his B.S. in Computer Science from Nanjing University in 2004, and Ph.D. in Computer Science and Engineering from the State University of New York at Buffalo in 2009. He has visited the University of Illinois at Urbana-Champaign (UIUC) as a Post Doc Research Associate. His research interests include wireless networking and mobile computing, data management, algorithmic network economics, and privacy preservation. He has published more than 150 peer-reviewed papers in technical journals and conference proceedings. He is a recipient of the first class prize for Natural Science Award of China Ministry of Education, NSFC Excellent Young Scholars Program, ACM China Rising Star Award, CCF-Tencent "Rhinoceros bird" Outstanding Award, and CCF-Intel Young Faculty Researcher Program Award. He has served as an associate editor of IEEE Transactions on Mobile Computing and ACM Transactions on Sensor Networks, an area editor of Elsevier Computer Networks, and as the member of technical program committees of more than 90 academic conferences. For more information, please visit http://www.cs.sjtu.edu.cn/~fwu/.



Shaojie Tang is currently an assistant professor of Naveen Jindal School of Management at University of Texas at Dallas. He received the PhD degree in computer science from Illinois Institute of Technology, in 2012. His research interests include social networks, mobile commerce, game theory, e-business, and optimization. He received the Best Paper Awards in ACM MobiHoc 2014 and IEEE MASS 2013. He also received the ACM SIGMobile service award in 2014. Dr. Tang served in various positions (as chairs and TPC members) at numerous conferences, including ACM MobiHoc and IEEE ICNP. He is an editor for International Journal of Distributed Sensor Networks.

Shuai Ma is a professor at the School of Computer Science and Engineering, Beihang University, China. He obtained his PhD degrees from University of Edinburgh in 2010, and from Peking University in 2004, respectively. He was a postdoctoral research fellow in the database group, University of Edinburgh, a summer intern at Bell labs, Murray Hill, USA and a visiting researcher of MSRA. He is a recipient of the best paper award for VLDB 2010 and the best challenge paper award for WISE 2013. He is an Associate Editor of VLDB Journal since 2017. His current research interests include database theory and systems, and big data.

Guihai Chen earned the BS degree from Nanjing University, in 1984, the ME degree from Southeast University, in 1987, and the PhD degree from the University of Hong Kong, in 1997. He is a distinguished professor of Shanghai Jiaotong University, China. He had been invited as a visiting professor by many universities including Kyushu Institute of Technology, Japan, in 1998, University of Queensland, Australia, in 2000, and Wayne State University, USA during September 2001 to August 2003. He has a wide range of research interests with focus on sensor network, peer-to-peer computing, high-performance computer architecture and combinatorics. He has published more than 200 peer-reviewed papers, and more than 120 of them are in well-archived international journals such as IEEE Transactions on Parallel and Distributed Systems, Journal of Parallel and Distributed Computing, Wireless Network, The Computer Journal, International Journal of Foundations of Computer Science, and Performance Evaluation, and also in well-known conference proceedings such as HPCA, MOBIHOC, INFOCOM, ICNP, ICPP, IPDPS, and ICDCS.