Blindfolded Evaluation of Random Forests with Multi-Key Homomorphic Encryption

Asma Aloufi, Peizhao Hu, Harry W. H. Wong, and Sherman S. M. Chow

Abstract—Decision tree and its generalization of random forests are a simple yet powerful machine learning model for many classification and regression problems. Recent works propose how to privately evaluate a decision tree in a two-party setting where the feature vector of the client or the decision tree model (such as the threshold values of its nodes) is kept secret from another party. However, these works cannot be extended trivially to support the outsourcing setting where a third-party who should not have access to the model or the query. Furthermore, their use of an *interactive* comparison protocol does not support branching program, hence requires interactions with the client to determine the comparison result before resuming the evaluation task.
In this paper, we propose the first secure protocol for collaborative evaluation of random forests contributed by multiple owners. They outsource evaluation tasks to a third-party evaluator. Upon receiving the client's encrypted inputs, the cloud evaluates obliviously on individually encrypted random forest models and calculates the aggregated result. The system is based on our new secure comparison protocol, secure counting protocol, and a multi-key somewhat homomorphic encryption on top of symmetric-key encryption. This allows us to reduce communication overheads while achieving round complexity lower than existing work.

Index Terms—Applied Cryptography, Decision Tree, Homomorphic Encryption, Machine Learning, Random Forest

1 Introduction

Decision tree is one of the most widely used nonparametric machine learning techniques for classification and regression. The evaluation process is a series of comparison at each *decision node* of the tree, which compares the input from a client with the threshold of the node as specified in the model. The boolean results decide which descendant node to traverse and eventually leads to a leaf node representing a result. Similar to some other machine learning frameworks, relying on a single such tree may incur the model-overfitting problem. A *random forest* (Fig. 1a) which aggregates the results from individual decision trees can provide more accurate results. The final result is either a list of classification labels together with counts associated with each label, or a classification label that most of the trees agreed on.

In this paper, we focus on a scenario where predictive models from multiple owners are sent to an evaluator for *collaborative* evaluation (Fig. 1b). Collaborative machine learning becomes a commonplace as it provides more accurate prediction due to the diversity in data [1]. Medical diagnostics is one example (EU WITDOM project) in which multiple hospitals and medical laboratories collaborate to offer a better diagnosis. A natural source for the evaluator is to rely on an external cloud. But, privacy leakage can occur [2], [3] due to security breaches or insider attacks. According to the statistics [4], there are around 2 declared breaches per week, each affecting $500+$ people. More importantly, leaking of sensitive personal data (e.g., genome, fingerprint, eye-iris-scan) is irreversible. Strict data protection regulations, such as HIPAA and EU GDPR[1], embrace data utility for medical diagnosis and drug discovery but demand provable security of private data when it is in storage and being processed.

1. https://www.hhs.gov/hipaa and https://www.eugdpr.org

1.1 Current Solutions

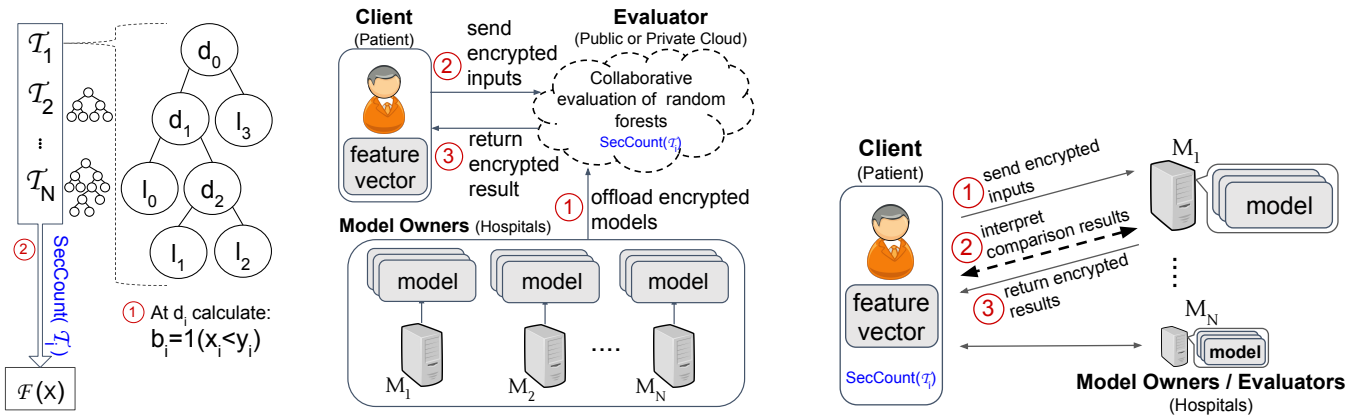
Existing privacy-preserving protocols [5]–[7] follow the client-server model where the server owns the random forest and the client inputs encrypted features to start the evaluation. Comparison at each node is carried out using the secure comparison protocol proposed by Damgård, Geisler, and Krøigaard (DGK protocol) [8] which takes inputs in binary and produces a list of intermediate results that are either encryption of zero or non-zero integer. These intermediate results cannot be used directly to perform branching program and traversal through the decision tree because the server requires the client's help to determine whether any ciphertext decrypts to zero. The client then generates and sends back an encrypted bit to resume the evaluation on the server side. This process based on the DGK protocol is interactive and makes the existing works to be a synchronous system design.

These state-of-the-art works [5]–[7] use additive homomorphic encryption (HE) and cannot be easily extended to work in a collaborative setting which naturally requires multiplication of two ciphertexts. A best-effort workaround is to have the client sends separate requests to each model owner, as illustrated in Fig. 1c. Leaving aside the communication overhead caused by the exchange of intermediate results, this naïve extension reveals the individual decision made by each model owner to the client.

Alternatively, model owners can outsource their models to a third-party evaluator. This is undesirable due to concerns of security breaches. One may encrypt each model under the key of its respective owner. Yet, such a multi-key usage is not considered in the existing privacy-preserving decision tree evaluation protocols.

1.2 Our Contributions

We propose a privacy-preserving protocol which allows multiple model owners to delegate the evaluation of the random forests (resulted from the implicit collaborative effort of combining individual models) to an untrusted party. As depicted in Fig. 1b,



(a) Random Forests Evaluation (b) Collaborative Setting (c) Two-Party Setting

Fig. 1: System models for collaborative evaluation of random forests $\mathcal{F}$, consisting of N decision trees: Each decision tree $\mathcal{T}_i$ with a depth of δ contains m decision nodes $d_i \in \mathcal{D}$, and $(m+1)$ leaf nodes $l_k \in \mathcal{L}$. Each leaf node contains a classification value $v_k \in V$.

individual model owners encrypt their decision trees such that none of them can get the decision tree of everyone else, while the cloud remains oblivious to the models or the query.

Our technical contributions are a new secure comparison protocol SecComp (see Sec. 5.5), a new secure counting protocol SecCount (see Sec. 5.6), the incorporation of techniques for computing over data encrypted under different keys, and various techniques for efficiency improvement.

SecComp directly produces an encrypted bit that is used for branching in decision tree evaluation. The extra interactions for processing the list of encrypted integers output by the DGK protocol (or other DGK-based protocols [9]) in the existing works are not necessary. It thus achieves a lower round complexity.

SecCount calculates the count of each class label resulted from the evaluation of individual decision trees and performs majority voting. To our best knowledge, this method has not been used in similar works. Recent work only considered a specific form of random forest evaluation based on the mean of the trees' results. It is only suitable for regression or binary classification problems but not for multi-class classification we consider.

A core building block of our system is a somewhat homomorphic encryption (SWHE) scheme proposed by Brakerski, Gentry, and Vaikuntanathan (BGV SWHE) [10]. Our system incorporates multiple techniques for efficiency improvement: (a) optimization techniques to speed-up homomorphic evaluation through parallelization, (b) hybrid encryption to lower the communication overhead, and (c) leveraging threshold encryption such that ciphertext extension does not grow linearly with respect to the number of (differnt encryption keys used by different) model owners. This new design significantly reduces the overhead caused by ciphertext extension as discussed in the original work [11].

While SWHE is relatively heavyweight, we aim to exploit as much as SWHE can provide for overcoming multiple limitations of existing works and at the same time improve the practical aspects of using SWHE by the aforementioned efficiency improvements. A side benefit of using lattice-based SWHE is that it provides post-quantum protection of user sensitive data, such as patients' genomic data. We also performed empirical experiments over multiple real-world datasets which validated the performance of our proposed solutions.

1.3 Organization

The next section presents the cryptographic techniques used in our system. Sec. 3 describes the system design and security model. In Sec. 4, we discuss the related work in more details. Sec. 5 presents our protocol and Sec. 6 discusses its correctness and security. After that, we provide complexity analysis and empirical study of the system performance. Finally, Sec. 7 concludes the paper.

2 Preliminaries

We give a brief description of the cryptographic primitives used in our protocol. In particular, we overview the primitives used to support homomorphic computation with multiple keys. We also overview hybrid encryption used for efficient communication, and techniques for privately retrieving information.

2.1 The RLWE Problem

For a security parameter λ, let $\Phi(x) = x^\eta + 1$ be a cyclotomic polynomial where $\eta = \eta(\lambda)$ is a power of 2, and $q = q(\lambda) \geq 2$ be an integer. Define the ring R over polynomials with integer coefficients $R = \mathbb{Z}[x]/(\Phi(x))$. Let $\chi = \chi(\lambda)$ be a Gaussian error distribution over R_q and bounded by $\mathcal{B} = \mathcal{B}(\lambda)$ such that $\mathcal{B} \ll q$.

Definition 2.1 (Ring-Learning-With-Errors (RLWE) [12]). Let a and s be uniformly sampled elements from R_q, and let $e \leftarrow \chi$ be a sampled error term. The RLWE problem is to distinguish the pair of $(a_i, b_i = a_i s + e)$ from any uniformly sampled pair $(a_i, b'_i) \leftarrow R_q^2$. The RLWE assumption is that the RLWE problem is computationally infeasible to solve.

An amortized version of the RLWE problem [12], [13] shows that it is equivalent to sampling s from a small distribution instead of the ring R_q. This yield a smaller secret key in an RLWE-based cryptosystem, e.g., BGV SWHE scheme [10] (Appendix A).

2.2 Multi-Key SWHE Scheme

There are two common extensions of SWHE to support computation with different keys: threshold HE [14] and multi-key HE [11], [15], [16]. In threshold HE scheme (Appendix C), n model owners $\mathcal{M}_i$ and p clients $\mathcal{C}_i$ leverage key homomorphism to generate a

joint key, this yields p distinct joint keys $\{\mathsf{pk}_{\mathcal{M},\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{M},\mathcal{C}_p}\}$. Beside generating these joint keys in advance, model owners must send p encrypted copies of the models must to the cloud, one for each distinct joint key, which increases the space and communication overhead of the system.

Multi-key BGV scheme (MKBGV) [11] (Appendix D) dynamically extends a ciphertext encrypted under one key to a concatenation of keys from all parties. For example, a ciphertext $c_{\mathcal{C}} = (c_0, c_1) \in R_q^2$ encrypted under a client's key $\mathsf{pk}_{\mathcal{C}}$ is extended to one under $\{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}_i}, \ldots, \mathsf{pk}_{\mathcal{M}_n}\}$, such that $\bar{c} = \{c_{\mathcal{C}}, c_{\mathcal{M}_1}, \ldots, c_{\mathcal{M}_n}\} \in R_q^{2(n+1)}$. Note the size of the ciphertext increases linearly with $(n+1)$ involved keys.

Observing the inefficiency in these two methods, we propose a new multi-key HE (MKHE) scheme. The set of model owners in our collaborative setting in Fig. 1b do not often change; hence, they can set up one joint key $\mathsf{pk}_{\mathcal{M}}$ based on their keys $\mathsf{pk}_{\mathcal{M}_i}$. They encrypt their models using this joint key. At time of evaluation, each ciphertext is extended under the two keys $\{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}$, resulting $\bar{c} = \{c_{\mathcal{C}}, c_{\mathcal{M}}\} \in R_q^4$. By leveraging both threshold and multi-key HE techniques, model owners send *one* copy of their encrypted models to the cloud, who extends them when needed to *two* keys, reducing the size in ciphertext expansion.

Our new MKHE scheme involves the following algorithms (Setup, KeyGen, Enc, Ext, $\mathsf{EvalKeyGen}$, Eval, Dec):

- $\mathsf{Setup}(1^\lambda, 1^L, 1^K) \rightarrow \mathsf{pp}$: A probabilistic algorithm inputs the security parameter λ, a bound on circuit depth L, a bound on number keys K, outputs public parameter pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow ((\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}}), \mathsf{ek}'_{\mathcal{C}})$ and $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow ((\mathsf{pk}_{\mathcal{M}_i}, \mathsf{sk}_{\mathcal{M}_i}), \mathsf{ek}'_{\mathcal{M}_i})$: Each system user (i.e., clients and model owners, respectively) generates a key-pair $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}}), (\mathsf{pk}_{\mathcal{M}_i}, \mathsf{sk}_{\mathcal{M}_i})$. They also generate the corresponding evaluation helper element $\mathsf{ek}'_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{M}_i}$, which encrypts powers-of-two of the secret key $\mathsf{sk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{M}_i}$ under GSW HE scheme [17].
- $\mathsf{JointkeyGen}(\mathsf{pk}_{\mathcal{M}_1}, \ldots, \mathsf{pk}_{\mathcal{M}_n}, \mathsf{ek}'_{\mathcal{M}_1}, \ldots, \mathsf{ek}'_{\mathcal{M}_n}) \rightarrow (\mathsf{pk}_{\mathcal{M}}, \mathsf{ek}'_{\mathcal{M}})$: The set of n model owners perform interactive protocol to establishes a joint key $\mathsf{pk}_{\mathcal{M}}$ by combining their keys as discussed. The corresponding evaluation helper element $\mathsf{ek}'_{\mathcal{M}}$ must encrypt the bits of the secret key $s_{\mathcal{M}}$, which is shared among the model owners. The evaluation helper element can be generated by homomorphically adding all $\mathsf{ek}'_{\mathcal{M}_i}$ using a binary addition $\mathsf{BinAdd}()$ that performs a fast full adder on the encrypted bits [18], [19].
- $\mathsf{Enc}(\mathsf{pk}, \mu) \rightarrow c$: A probabilistic algorithm that performs the standard BGV encryption on given message $\mu \leftarrow R_t$ and a public key pk and outputs $c = (c_0, c1) \in R_q^2$. Each ciphertext is associated with set I that holds the index of the used key. We indicate the index 1 as the client's key $\mathsf{pk}_{\mathcal{C}}$ and the index 2 as the model owners' joint key $\mathsf{pk}_{\mathcal{M}}$.
- $\mathsf{Ext}(\bar{\mathsf{pk}}, c) \rightarrow \bar{c}$: A deterministic algorithm extends a given ciphertext c to one encrypted under the concatenated keys $\bar{\mathsf{pk}} = \{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}$. Simply output a concatenated two sub-vectors $\bar{c} = (c_{\mathcal{C}}, c_{\mathcal{M}}) \in R_q^4$, such that $c_i = c_i$ if the index $i \in I$, and $c_i = 0$ otherwise. We denote $[[\cdot]]$ as an extended BGV ciphertext.
- $\mathsf{EvalKeyGen}(\mathsf{ek}'_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{M}_i}) \rightarrow \bar{\mathsf{ek}}$: A deterministic algorithm generates an evaluation key $\bar{\mathsf{ek}}$ for the concatenated public key $\bar{\mathsf{pk}} = \{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}$ based on given helper elements $\{\mathsf{ek}'_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{M}}\}$. The evaluation key is generated as follows. First, extend each evaluation helper element $\mathsf{ek}'_{\mathcal{C}}$ (or $\mathsf{ek}'_{\mathcal{M}}$) to the other key $\mathsf{pk}_{\mathcal{M}}$ (or $\mathsf{pk}_{\mathcal{C}}$). Then, perform a homomorphic multiplication to compute the encryption of $\bar{s} \otimes \bar{s}$, where $\bar{s} = \{s_{\mathcal{C}}, s_{\mathcal{M}}\}$. This encrypts the powers-of-two of the concatenated keys, which is used in $\mathsf{KeySwitch}$ (i.e., relinearization) after each homomorphic evaluation to bring $\bar{s}^2$ to $\bar{s}$.
- $\mathsf{Eval}(\bar{c}, \bar{c}', \bar{\mathsf{ek}}) \rightarrow \bar{c}_{\mathtt{add}}$ or $\bar{c}_{\mathtt{mult}}$: A deterministic algorithm inputs two extended ciphertexts $\bar{c}, \bar{c}' \in R_q^4$ encrypted under the concatenated keys $\bar{\mathsf{pk}}$, perform homomorphic addition as element-wise addition $\bar{c}_{\mathtt{add}} = \bar{c} + \bar{c}' \in R_q^4$ or the homomorphic multiplication as the tensor product $\bar{c}_{\mathtt{mult}} = \bar{c} \otimes \bar{c}' \in R_q^{16}$. After the homomorphic evaluation, perform $\mathsf{KeySwitch}$ technique using the generated evaluation key $\bar{\mathsf{ek}}$ to output a ciphertext in R_q^4, and the $\mathsf{ModulusSwitch}$ technique to reduce the resultant noise by switching to a smaller ciphertext modulus. Note that during evaluation, for example in homomorphic addition, the underlying messages $\mu_{\mathcal{C}}$ and $\mu_{\mathcal{M}}$ do not directly add up. The sub-ciphertexts are added together after decryption, yielding the correct evaluation result.
- $\mathsf{Dec}(\bar{\mathsf{sk}}, \bar{c}) \rightarrow \mu$ or $\perp$: An interactive protocol decrypting an extended ciphertext $\bar{c} = \{c_{\mathcal{C}}, c_{\mathcal{M}}\} \in R_q^4$ with the concatenated secret keys $\bar{s} = \{s_{\mathcal{C}}, s_{\mathcal{M}}\}$. The client and model owners collaborate to decrypt the message as $\langle \bar{c}, \bar{s} \rangle = \langle c_{\mathtt{C}}, s_{\mathcal{C}} \rangle + \langle c_{\mathtt{M}}, s_{\mathcal{M}} \rangle$. The client obtains $\mu_{\mathcal{C}}$ through straightforward BGV decryption. Since $s_{\mathcal{M}}$ is shared among n model owners, they perform the partial decryption $\langle c_{\mathtt{M}}, s_{\mathcal{M}} \rangle = (c_{\mathcal{M},1} - c_{\mathcal{M},0} s_{\mathcal{M}})$ in a threshold manner (Appendix C). Each model owner $\mathcal{M}_i$ computes a smudged[2] component $\rho_{\mathcal{M}_i} = c_{\mathcal{M},0} s_{\mathcal{M}_i} + t e_{\mathcal{M}_i}$, then all model owners compute together $c_{\mathcal{M},1} - \sum_{i=1}^{n} c_{\mathcal{M},0} s_{\mathcal{M}_i}$, yielding $\mu_{\mathcal{M}}$. The protocol outputs the message $\mu = \mu_{\mathcal{C}} + \mu_{\mathcal{M}}$ for the client and nothing abort $\perp$ for model owners and evaluator.

Definition 2.2 (Correctness and Compactness). Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^K)$. Consider a correctly generated key pairs $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{C}})$ and $(\mathsf{pk}_{\mathcal{M}_i}, \mathsf{sk}_{\mathcal{M}_i}, \mathsf{ek}'_{\mathcal{M}_i}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and any two messages $\mu_{\mathcal{C}}, \mu_{\mathcal{M}}$. Let $\{c_{\mathcal{C}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathcal{C}}, \mu_{\mathcal{C}})\}$ and $\{c_{\mathcal{M}} \leftarrow \mathsf{Enc}(\mathsf{pk}_{\mathcal{M}}, \mu_{\mathcal{M}})\}$. Let $\bar{c} \leftarrow \mathsf{Ext}(\bar{\mathsf{pk}}, c_{\mathcal{C}})$ and $\bar{c}' \leftarrow \mathsf{Ext}(\bar{\mathsf{pk}}, c_{\mathcal{M}})$, where $\bar{\mathsf{pk}} = \{\mathsf{pk}_{\mathcal{C}}, \mathcal{M}\}$. Let f be function with depth bounded by L, let $\bar{c}_f \leftarrow \mathsf{Eval}(\bar{c}, \bar{c}', \bar{\mathsf{ek}})$ be the evaluated ciphertext:

- *Correctness*: For any negligible function $\epsilon(\lambda)$, we have

$$|\mathsf{Pr}[\mathsf{Dec}(\bar{\mathsf{sk}}, \bar{c}) = f(\mu_{\mathcal{C}}, \mu_{\mathcal{M}})]| \geq 1 - \epsilon$$

- *Compactness*: There exists a polynomial $p(\cdot)$ such that $|\bar{c}| \leq p(\lambda, L, K)$.

Definition 2.3 (Semantic Security). Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^K, 1^L)$, and $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{C}})$ and $(\mathsf{pk}_{\mathcal{M}}, \mathsf{sk}_{\mathcal{M}}, \mathsf{ek}'_{\mathcal{M}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. For any polynomial $d = d(\lambda)$ and any two messages μ_0, μ_1 the following distribution are computationally indistinguishable:

$$(\mathsf{pp}, \mathsf{pk}_{\mathcal{C}}, \mathsf{Enc}(\mathsf{pk}_{\mathcal{C}}, \mu_0)) \overset{\text{comp}}{\approx} (\mathsf{pp}, \mathsf{pk}_{\mathcal{C}}, \mathsf{Enc}(\mathsf{pk}_{\mathcal{C}}, \mu_1))$$

and

$$(\mathsf{pp}, \mathsf{pk}_{\mathcal{M}}, \mathsf{Enc}(\mathsf{pk}_{\mathcal{M}}, \mu_0)) \overset{\text{comp}}{\approx} (\mathsf{pp}, \mathsf{pk}_{\mathcal{M}}, \mathsf{Enc}(\mathsf{pk}_{\mathcal{M}}, \mu_1))$$

2. Smudging noise is added to hide the secret key $s_{\mathcal{M}_i}$ (Appendix B.2)

2.3 Hybrid Approach of Homomorphic Encryption

Despite the recent advancement, SWHE schemes produce ciphertexts that are large in size. It is thus more efficient to use a hybrid approach [20], [21] which firstly uses an efficient block-cipher (AES) to encrypt the data.

Let $\mathsf{Enc}_{\mathtt{AES}}(\mu)$ be a ciphertext of the message μ encrypted under an AES key k, the cloud encrypts the ciphertext again using SWHE $\mathsf{Enc}_{\mathtt{BGV}}(\mathsf{Enc}_{\mathtt{AES}}(\mu))$. To decrypt, the cloud homomorphically decrypts the AES ciphertext with an SWHE encryption of the AES key $\mathsf{Enc}_{\mathtt{BGV}}(k)$. We denote this by $\mathsf{Enc}_{\mathtt{BGV}}(\mu) = \mathsf{homAESdec}(\mathsf{Enc}_{\mathtt{BGV}}(\mathsf{Enc}_{\mathtt{AES}}(\mu)), \mathsf{Enc}_{\mathtt{BGV}}(\mathsf{k}))$, where we use the homAESdec function of HElib [22].

2.4 Oblivious Transfer

Oblivious transfer (OT) [23] allows the receiver Bob to retrieve a message μ_i from a sender Alice who has a set of n messages $(\mu_1, \ldots, \mu_n)$, without Alice knowing his choice $i \in \mathbb{N}$. Furthermore, Bob only knows the message corresponding to his chosen index i but not the other messages. In our paper, such a 1-out-of-n OT is denoted by $\mathtt{OT}_n^1(\{\mu_1, \ldots, \mu_n\}, i) = \mu_i$.

3 System Setup

3.1 System Model

Each model owner $\mathcal{M}_i$ has a set of decision trees $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$. We model a *complete* decision tree as $\mathcal{T} = (\mathcal{D}, \mathcal{L})$, where $\mathcal{D}$ is a set of m decision nodes and $\mathcal{L}$ is a set of $(m+1)$ leaf nodes. For classification, each leaf node $l_k \in \mathcal{L}$ contains a class label v_k. In the case of textual class labels, such as different classes of blood diseases {"Anemia", "Leukemia", ..., "Hemophilia"} [24], we hash them into numerical values. We assume that this encoding is publicly known.

At each decision node $d_i = (f_i, y_i) \in \mathcal{D}$, there is a boolean function that takes a user input $x_i \in X$, compares it with a threshold value $y_i \in Y$, and produces a result, such that $b_i = f_i(x_i < y_i)$, as illustrated in Fig. 1a. Here, $X = \{x_0, \ldots, x_{m-1}\}$ is a vector of features provided by the client while $Y = \{y_0, \ldots, y_{m-1}\}$ is a vector of threshold values defined for all decision nodes in a tree. The collections of all (or a subgroup of) decision trees contributed by all model owners form a random forest $\mathcal{F} \subseteq \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, which is evaluated in the cloud and produce a classification result such that $v = \mathcal{F}(X); v \in V$.

3.2 Threat Model

A client adversary may attempt to learn, through sent queries, information about the random forest, such as the tree structure, threshold values, or class labels in the leaf node. A client should learn nothing other than what is known in public, such as tree depth δ and number of decision/leaf nodes m. Note that we can hide the tree structure by adding *dummy* nodes. On the other hand, a model owner may try to learn about models contributed by other model owners, or the client's sensitive data through the provided queries, or the (intermediate) evaluation result during decryption.

In our outsourced setting, the (cloud) evaluator is a potential adversary who may want to learn both of the above, i.e., the query of the client and its final result, and the models of the owners.

4 Related Work

4.1 Secure Comparison Protocol

A critical part in decision tree evaluation is to compute $b = (x < y)$ as shown in Fig. 1a. Given two encrypted ℓ-bit inputs x, y, many secure comparison protocols, such as those by Damgård *et al.* [8], Veugen [9], [25], operate over individual encryption of bits in $x = \{x_{\ell-1}, \cdots, x_0\}$, $y = \{y_{\ell-1}, \cdots, y_0\}$, and rely on arithmetic computations to determine whether the specified relation holds.

The DGK protocol [8] is widely used for comparing two encrypted ℓ-bit inputs without decryption. It has been proven that the comparison result bit $b = 1\{x < y\}$ is set to 1 if and only if there exists an index $i \in (0, \cdots, \ell - 1)$ such that for a bit-wise comparison $x_i < y_i$ and $x_j = y_j$ for all leading bits at position $j > i$. Thus, DGK performs the following arithmetic computation

$$z_i = x_i - y_i + 1 + \sum_{j>i}(x_j \oplus y_j)$$

which produces ℓ results z_i that can either be an encryption of zero or a non-zero integer. One thus needs to check within a vector of encrypted integers whether there is a z_i that decrypts to zero, that is, the result is true. Veugen [25] proposed an improvement (based on the proposition in [26]) to support the comparison relation of both $x < y$ and $x > y$ by adding a single bit input. The improved DGK protocol has the following arithmetic form.

$$z_i = x_i - y_i + \beta + 3\sum_{j>i}(x_j \oplus y_j)$$

where $\beta = 1 - 2 \cdot b'$ and $b' \xleftarrow{\$} \{0, 1\}$, i.e., a uniformly sampled bit. If $b' = 0$ then the protocol is checking $x < y$, otherwise $x > y$. This random flipping of the comparison rule can hide the structure of the decision tree (and prevent active probing of the threshold values in the decision nodes). Similarly, the output bit is true if there exists encryption of zero in z_i.

Veugen [9] also proposed an alternative protocol to compare two encrypted [27], [28] integers. This protocol outputs a single encrypted bit, which is also adapted by Bost *et al.* [5] for secure machine learning. Yet, it incurs three rounds, one of those uses the DGK protocol [8] to compare two intermediate results. It also introduces significant communication overheads.

4.2 Secure Evaluation of Decision Trees

We give some technical highlights of the existing protocols for a better understanding of either their intrinsic weaknesses or some similar working mechanisms which our protocol shares. A comparison of recent work will be discussed in Table 3.

The protocol of Wu *et al.* [6] uses the improved DGK protocol to evaluate all boolean functions in the decision nodes with the help of the client as discussed in Sec. 4.1. After that, the protocol concatenates the encrypted boolean results to assemble an encrypted binary string $b_0 b_1 \cdots b_{m-1}$ that corresponds to the index of the leaf node containing the evaluation result. The server sends this binary string to the client who uses OT (Sec. 2.4) to privately retrieve the evaluation result.

Bost *et al.* [5] proposed a set of homomorphic protocols for common operations such as dot product, argmax, comparison, which are the building blocks of many machine learning algorithms including hyperplane, naïve Bayes, and decision tree. Their approach uses two multiple-round secure comparison protocols, which produce a single encrypted bit at each decision node. This

allows the evaluation result to be revealed directly on the server, instead of relying on an OT protocol [6]. However, the comparison protocol still requires interaction between client and server. Once all Boolean functions have been evaluated, the authors proposed to transform a decision tree, like the example shown in Fig. 1a, into a polynomial form such as:

$$b_0 \cdot l_3 + (1 - b_0) \cdot (b_1 \cdot (b_2 \cdot l_2 + (1 - b_2) \cdot l_1) + (1 - b_1) \cdot l_0)$$

where $b_i = f_i(x_i \leq y_i)$ is a boolean function to be evaluated at each decision node $d_i \in \mathcal{D}$ using a secure comparison protocol and each leaf node has an assignment of a class label l_k.

The server homomorphically evaluates the above polynomial which then reveals the correct classification value. For polynomial evaluation, SWHE such as the Brakerski–Gentry–Vaikuntanathan (BGV) scheme [10] is needed. Evaluating a polynomial of a decision tree with high depth δ impacts the efficiency since the polynomial performs δ consecutive homomorphic multiplications. The authors suggested to compute the multiplications in pairs; thus; the multiplicative-depth decreases to $\lceil \log_2(\delta) \rceil$. We explore this idea further, develop an algorithm to speed up the evaluation, and conduct empirical experiments to study its efficiency.

Tai *et al.* [7] proposed the concept of *path cost* for transforming the tree into a set of linear equations which is compatible with efficient cryptographic operation. For each leaf node l_k, it computes the sum of the boolean results b_i along the path from the root to l_k as the path cost $\mathtt{pc}_k$. For example, the path costs in the decision tree shown in Fig. 1a are: $\mathtt{pc}_0 = b_0 + b_1$, $\mathtt{pc}_1 = b_0 + (1 - b_1) + b_2$, $\mathtt{pc}_3 = 1 - b_0$, and $\mathtt{pc}_2 = b_0 + (1 - b_1) + (1 - b_2)$. The classification values v_k to be retrieved by the client via the conditional OT is randomized if and only if $\mathtt{pc}_k$ is non-zero, which ensures that the client can only learn the result corresponding to his inputs. When compared with the use of OT protocols by Wu *et al.* [6], it avoids sending a binary string to the client who then retrieves the final result using the concatenated index.

Recently, Joye and Salehi [29] proposed a new privacy-preserving decision tree protocol using OT and additive HE. They proposed a secure comparison protocol by increasing the interactive rounds in favor of reducing the number of comparisons. The protocol of Tueno, Kerschbaum, and Katzenbeisser [30] represents a tree of depth δ as an array. They did not use HE, but used garbled circuits, OT, or Oblivious RAM, to evaluate the tree with only δ comparisons which is sublinear in the tree size. Both protocol works in the two-party setting and do not support data encrypted under different keys. Moreover, the boolean functions are evaluated interactively, which leads to high round complexity.

4.3 Other Related Approaches

Other works also considered the learning phase. Emekçi *et al.* [31] utilized secret sharing to build a decision tree model based on private datasets from multiple parties. Using SWHE, Bos *et al.* [32] aim to evaluate known predictive models, such as logistic regression and proportional hazards models, on encrypted medical data. Hu *et al.* [33] aim to support ridge regression.

Some recent works study privacy-preserving machine learning over other models such as neural networks. The work of Mohassel and Zhang [34] requires two non-colluding servers. Liu *et al.* [35] did not consider the outsourced setting and their protocol is highly interactive. Using SGX, Ohrimenko *et al.* [36] proposed a system which allows multiple parties to load their private datasets into the enclave, which is an isolated memory region where oblivious codes can train a machine learning model based on these datasets. It also supports evaluation. However, tailor-made protocols outperform the generic approaches in many cases because of they exploit the structure of the underlying model.

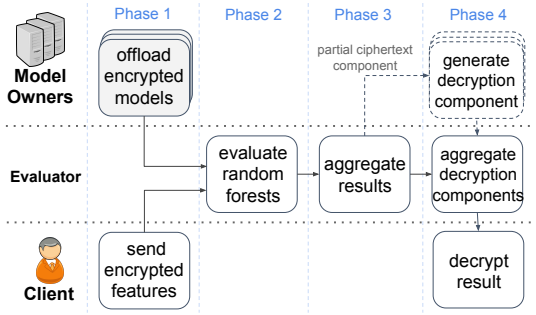


Fig. 2: Overview of Different Phases of Our Proposed System

5 Proposed Solution

5.1 Overview

Fig. 2 illustrates the four phases of interactions between different parties. In the first phase, each model owner $\mathcal{M}_i$ encrypts a set of decision trees $\{\mathcal{T}_i\}$, includes all the threshold values in their binary format. Delegating the encrypted models to the evaluator can be performed as a one-time setup before servicing the clients.

In the second phase, upon receiving a feature vector X encrypted under the key of the client, the evaluator evaluates every decision tree in the entire random forest. Once it is done, each decision tree outputs a class label. The evaluator will perform a secure counting protocol to obliviously aggregate the number of occurrences for each unique class label. The evaluator then sends the class labels with their associated counts to the client.[3]

In the final phase, each model owner will participate in the partial decryption, which sends a decryption component to the evaluator to convert the encrypted result to a ciphertext which is decryptable by the secret key of the client.

5.2 Notations and Steps in Different Phases

Fig. 3 details our proposed system. Data are sent at first as AES ciphertexts for efficient transmission. The evaluator homomorphically decrypts them into SWHE ciphertexts of decision tree $\mathcal{T}$, which can then be homomorphically evaluated resulting $\mathcal{T}(X) = v$. Similar to many existing work [6], [7], we convert each feature and threshold value into its binary form $x = \{x_{\ell-1}, \cdots, x_0\} \in X$ and $y = \{y_{\ell-1}, \cdots, y_0\} \in Y$. Then, the evaluator computes the final classification result of the random forest by a joint computation over the individual trees.

Table 1 lists commonly used notations in the proposed system. We use $\langle \cdot \rangle$, $[\cdot]$, and $[[\cdot]]$ to denote AES, SWHE, and MKHE encryption respectively. Sometimes we may omit these encryption notations for clarity, but all computations are on ciphertexts.

3. An alternative option is to engage with the client in an additional OT protocol to return the final class label with the highest vote.

Setup phase.

Model owner $\mathcal{M}_i$: (1) Train on the datasets and generate a set of decision trees $\{\mathcal{T} = (\mathcal{D}, \mathcal{L})\}$. Each decision node $d_i = (f_i, y_i) \in \mathcal{D}$ consists of a threshold value $y_i \in Y$. Each leaf node $l_k \in \mathcal{L}$ contains a class label $v_k \in V$.
(2) Generate an AES key $k_{\mathcal{M}_i}$.
(3) Generate SWHE key pair and evaluation helper element $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow ((\mathsf{pk}_{\mathcal{M}_i}, \mathsf{sk}_{\mathcal{M}_i}), \mathsf{ek}'_{\mathcal{M}_i})$.
(4) Generate a joint encryption key and evaluation helper element with all other model owners $\mathsf{JointkeyGen}(\mathsf{pk}_{\mathcal{M}_1}, \ldots, \mathsf{pk}_{\mathcal{M}_n}, \mathsf{ek}'_{\mathcal{M}_1}, \ldots, \mathsf{ek}'_{\mathcal{M}_n}) \rightarrow (\mathsf{pk}_{\mathcal{M}}, \mathsf{ek}'_{\mathcal{M}})$.

Client $\mathcal{C}$: (1) Provide a set of features $\{x_1, \ldots, x_n\}; x_i \in X$.
(2) Generate AES key $(k_{\mathcal{C}})$.
(3) Generate SWHE public key, secret key and evaluation helper element $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{C}})$.

Phase 1: Outsourcing computations.

Model owner $\mathcal{M}_i$: (1) Convert each threshold y_i and label v_k into bit-wise representation $\{y_{\ell-1}, \cdots, y_0\}$ and $\{v_{\ell-1}, \cdots, v_0\}$.
(2) Encrypt each bit in y_i and v_k using AES $\{\langle y_{\ell-1}\rangle, \cdots, \langle y_0\rangle\}$ and $\{\langle v_{\ell-1}\rangle, \cdots, \langle v_0\rangle\}$.
(3) Encrypt the AES key using SWHE joint key: $[k_{\mathcal{M}_i}]_{\mathcal{M}}$.
(4) Send $(\mathsf{pk}_{\mathcal{M}}, [k_{\mathcal{M}_i}]_{\mathcal{M}}, \{\langle y_j\rangle\}, \{\langle v_j\rangle\})$ to the evaluator.

Client $\mathcal{C}$: (1) Convert each feature into bit-wise representation of length ℓ: $x_i = \{x_{\ell-1}, \cdots, x_0\} \in X$.
(2) Encrypt each bit in x_i using AES: $\{\langle x_{\ell-1}\rangle, \cdots, \langle x_0\rangle\}$.
(3) Encrypt AES key using SWHE: $[k_{\mathcal{C}}]_{\mathcal{C}}$.
(4) Send $(\mathsf{pk}_{\mathcal{C}}, [k_C]_{\mathcal{C}}, \{x_j\})$ to the evaluator to start the evaluation.

Evaluator: (1) Upon receiving $\{y_j\}$, convert ciphertexts from AES to SWHE: $[y_j]_{\mathcal{M}} = \mathsf{homAESdec}([\langle y_j\rangle], [k_{\mathcal{M}}])$;
(2) Apply similar procedure for $\{v_j\}$ and $\{x_j\}$, yielding $\{[v_j]_{\mathcal{M}}\}$ and $\{[x_j]_{\mathcal{C}}\}$.
(3) Extend each of $[y_j]$ and $[v_j]$ via $[[y_j]] = \mathsf{Ext}(\bar{\mathsf{pk}}, [y_j])$ and $[[v_j]] = \mathsf{Ext}(\bar{\mathsf{pk}}, [v_j])$ for extended SWHE key $\bar{\mathsf{pk}} = \{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}$.
(3) Extend each of $[y_j]$ and $[v_j]$ via $[[y_j]]$ and $[[v_j]]$ via the subroutine of Eval for extended SWHE key $\bar{\mathsf{pk}} = \{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}$.
(4) Similarly, extend ciphertexts $\{[x_j]\}$ from encryptions under $\mathsf{pk}_{\mathcal{C}}$, yielding $[[x_j]] = \mathsf{Ext}(\bar{\mathsf{pk}}, [x_j])$. (4) Similarly, extend ciphertexts $\{[x_j]\}$ from encryptions under $\mathsf{pk}_{\mathcal{C}}$, yielding $[[x_j]]$ via the subroutine of Eval.
(5) Generate evaluation key $\bar{\mathsf{ek}} = \mathsf{EvalKeyGen}(\{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}, \{\mathsf{ek}'_{\mathcal{C}}, \mathsf{ek}'_{\mathcal{M}}\})$ via the subroutine of Eval.

Phases 2&3: Evaluating decision trees and random forest.

Evaluator: (1) Evaluate $[[b_i]] = \mathsf{SecComp}([[x_i]], [[y_i]])$; the result is $[[b_i]] = [[1]]\{x_i < y_i\}$.
(2) Evaluate the polynomial representation of the tree using all $[[b_i]]$, yielding $[[\mathcal{T}_i(X)]] = [[v_k]]$ for all trees in random forest $\mathcal{F}$.
(3) Invoke $\mathsf{SecCount}$ over all $[[\mathcal{T}_i(X)]]$ to get $[[\mathcal{F}(X)]] = \{([[v_1]], [[z_1]]), \ldots, ([[v_n]], [[z_n]])\}$ or $[[v_k]]$ where z_k is the maximum.
(4) Send the ciphertext element $c_{\mathcal{M},0}$ from the encrypted result $[[\mathcal{F}(X)]]$ to each model owner $\mathcal{M}_i$.

Phase 4: Decrypting result.

Model owner $\mathcal{M}_i$: Use the secret share $s_{\mathcal{M}_i}$ to construct $\rho_i = c_{\mathcal{M},0} s_{\mathcal{M}_i} + te_{\mathcal{M}_i}$ and send this ρ_i back to the evaluator.
Evaluator: Send encrypted results $[[\mathcal{F}(X)]]$ and aggregated decryption component $\rho = \sum_{i=1}^{N} \rho_i = c_{\mathcal{M},0} s_{\mathcal{M}} + te_{\mathcal{M}}$ to the client.
Client $\mathcal{C}$: Use the provided component ρ and the secret key $\mathsf{sk}_{\mathcal{C}}$ to decrypt $[[\mathcal{F}(X)]]$ that is under the extended key $\bar{\mathsf{pk}}$.

Fig. 3: Details of Different Phases of Our Proposed System

TABLE 1: Notations used in the proposed system

Notation	Description
$\mathcal{C}, \mathcal{M}_i$	A client and i-th model owner, respectively
$\mathcal{F}$	A random forest with the trees $\{\mathcal{T}\}$ of all owners $\mathcal{M}$
$\mathcal{T}_{i,w}$	w-th decision tree of model owner $\mathcal{M}_i$
V	A vector of class labels, $V = \{v_0, \cdots, v_n\}$
$Y_{i,w}$	A threshold vector of $\mathcal{T}_{i,w}$; $Y_{i,w} = \{y_0, \cdots, y_{m-1}\}$
X	A feature vector of a client $\mathcal{C}$, $X = \{x_0, \cdots, x_{m-1}\}$
$\mathsf{pk}_{\mathcal{C}}$	SWHE public key of a client $\mathcal{C}$
$\mathsf{pk}_{\mathcal{M}_i}$	SWHE public key of a model owner $\mathcal{M}_i$
$\mathsf{pk}_{\mathcal{M}}$	A joint key $\sum_{i=1}^{n} \mathsf{pk}_{\mathcal{M}_i}$
$\bar{\mathsf{pk}}$	An MKHE concatenated key $\{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}}\}$
$\langle\cdot\rangle_i$	AES encryption with party i symmetric key k_i
$[\cdot]_i$	SWHE encryption under pk_i, also as c
$[[\cdot]]$	MKHE extended encryption w.r.t. $\{\mathsf{pk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{M}}\}$, also as $\bar{c}$

5.3 Establishing Multiple Keys

As shown in our collaborative setting in Fig. 1b, there are two main roles, client and model owner. Before participating in the protocol, both must run a key setup to generate SWHE keys.

Each client $\mathcal{C}$ independently generates an SWHE key pair $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}})$ and $\mathsf{ek}'_{\mathcal{C}}$ which encrypts information about the secret key and used later to generate the evaluation key $\mathsf{ek}_{\mathcal{C}}$ as in Sec. 2.2.

The model owners set up a joint key as the sum of their independently generated SWHE keys $\mathsf{pk}_{\mathcal{M}} = \sum_{i=1}^{n} \mathsf{pk}_{\mathcal{M}_i}$, once before the start of the protocol. It is used to encrypt the models before sending them to the cloud. Similarly, they generate the corresponding evaluation helper element $\mathsf{ek}'_{\mathcal{M}}$ as the combination of their individual evaluation helper elements $\mathsf{ek}'_{\mathcal{M}_i}$ which encrypts information about the secret keys.

Each client $\mathcal{C}$ and model owner $\mathcal{M}_i$ also generates their own AES keys, $k_{\mathcal{C}}$ and $k_{\mathcal{M}_i}$ respectively, which will be used to encrypt their data at transmission to lower the communication overhead.

5.4 Outsourcing Computations

As a one-time setup (Phase 1 in Fig. 2), each model owner sends to the evaluator AES encrypted decision trees with each represented as a vector of threshold values $y_i \in Y$ corresponds

to decision nodes $d_i \in \mathcal{D}$, and a vector of class labels in leaf nodes $l_k = v; l_k \in \mathcal{L}; v \in V$. Each threshold value and class label are encrypted bit-wise in the form of $\langle y_{\ell-1}\rangle, \ldots, \langle y_0\rangle$ and $\langle v_{\ell-1}\rangle, \ldots, \langle v_0\rangle$. The encrypted models can also be updated.

Hiding model structure. To preserve the privacy of model structure from the evaluator, the model owner can optionally introduce *dummy* nodes [6] to the decision tree $\mathcal{T}$ to transform it to a *complete* tree, which has a depth δ, $(2^\delta - 1)$ decision nodes, and 2^δ leaf nodes. In this case, the evaluator will obliviously evaluate each decision node (including the dummy nodes), which will add overhead on the performance cost. One might add dummy nodes with no computation requirement, but this method fails if a malicious evaluator launches a timing attack against the protocol execution. Hence, for stronger security, we suggest adding dummy nodes that have random threshold values, and the resulting branches will point to the same leaf node.

Processing a query. When the client requests an evaluation from the evaluator, the evaluator converts each feature bit-wise from AES encryption to SWHE, such that $[x_j] = \mathsf{homAESdec}([\langle x_j\rangle], [k_\mathcal{C}])$. Then, the evaluator further extends each ciphertext under the set of the two keys $\bar{\mathsf{pk}} = \{\mathsf{pk}_\mathcal{C}, \mathsf{pk}_\mathcal{M}\}$ using the extension function $\mathsf{Ext}(\bar{\mathsf{pk}}, c)$ described in Sec. 2.2. The extended client's ciphertext is $[[x_j]] = \{[x_j]_\mathcal{C}, 0\} \in R_q^4$. Similarly, the evaluator extends each model's ciphertext from one under model owners' joint key $\mathsf{pk}_\mathcal{M}$ to one under $\bar{\mathsf{pk}}$ to obtain $[[y_j]] = \{0, [y_j]_\mathcal{M}\}$ and $[[v_j]] = \{0, [v_j]_\mathcal{M}\}$.

As mentioned, the concatenated key $\{\mathsf{pk}_\mathcal{C}, \mathsf{pk}_\mathcal{M}\}$ consists of the client's key and all model owners' joint key. The joint key is a combination of partial keys of all involved model owners to prevent information leakage due to collusion between the evaluator and a malicious model owner/client. It can be revoked by any model owner simply by refreshing their own new partial key pair. However, this will require to run the threshold key setup again and encrypting the models with the new joint key.

This design is both secure and efficient since model owners do not have to send different encryptions of their models for each registered client. The evaluator only extends these individually encrypted models to ciphertexts under the two keys $\{\mathsf{pk}_\mathcal{C}, \mathsf{pk}_\mathcal{M}\}$ when it receives the client's request.

5.5 Evaluating Encrypted Decision Trees

Upon receiving the client's encrypted feature vector X, the evaluator evaluates each decision tree in the random forest $\mathcal{T}_i \in \mathcal{F}$. Below, we focus our descriptions on the evaluation of a single decision tree and omit some indexes for clarity. The same evaluation procedures are applied in parallel to each tree.

Secure Comparison. Given a feature $x \in X$ and a threshold value $y \in Y$, the evaluator evaluates a boolean function $b = 1(x < y)$. We use our new protocol $\mathsf{SecComp}(x, y)$ which computes the single-bit encrypted output b as follows.

$$b = \begin{cases} (x_{\ell-1} < y_{\ell-1}) \vee \\ (x_{\ell-1} = y_{\ell-1}) \wedge (x_{\ell-2} < y_{\ell-2}) \vee \\ \vdots \\ (x_{\ell-1} = y_{\ell-1}) \wedge \cdots \wedge (x_1 = y_1) \wedge (x_0 < y_0) \end{cases} \quad (1)$$

where $(x_j < y_j) \equiv (\neg x_j \wedge y_j) \equiv (1 - x_j)y_j$ and $(x_j = y_j) \equiv (x_j \oplus y_j + 1) \equiv x_j + y_j + 1$.

The intuition is similar to the DGK protocol [8] (Sec. 4.1), but we remove the linear dependency on ℓ for transferring and interactively processing ℓ ciphertexts encrypting zero or non-zero integers. Our protocol can be extended to support equality checking if needed, such that:

$$b' = (x_{\ell-1} = y_{\ell-1}) \wedge \cdots \wedge (x_1 = y_1) \wedge (x_0 = y_0) \quad (2)$$

The evaluator can then evaluation the comparison $b = 1(x \leq y)$ by simply combining Eqns. 1 and 2 to compute $b = (b \vee b')$.

Existing works [5]–[7] are using an additional technique to randomly flip the branches and comparison rules at each decision node to prevent a malicious client from probing the threshold value. However, we do not apply this feature because our comparison protocol can be evaluated without interaction with the client.

Optimizing Secure Comparison. The multiplicative depth of SecComp increases with the increase of bit-length ℓ due to the increase of consecutive homomorphic multiplications performed at each bit-comparison. For example, a 4-bit comparison protocol illustrated in Fig. 4(a) requires 3 OR, 5 AND, 3 bit-by-bit equality checks, and 4 bit-by-bit *less-than* comparison checks. We find that for comparing two ℓ-bit values, we evaluate $(\ell-1)$ number of OR gates, $(2\ell-3)$ number of AND gates, $(\ell-1)$ equality checks, and ℓ bit-by-bit *less-than* comparison checks. So, the multiplicative depth of an ℓ-bit comparison protocol is $(3\ell - 2)$. Consecutive homomorphic multiplication impacts the efficiency greatly.

We speed up the SecComp evaluation through parallelization at a reduced multiplicative depth. This is done by translating the comparison into a binary evaluation tree as shown in Fig. 4(b). The evaluation tree reveals the dependencies between computations for comparison. All the computations in the same level can be evaluated in parallel. The tree is arranged in a way for balancing the computations being assigned to each processors. The result is acquired by merging the branches of the tree bottom up.

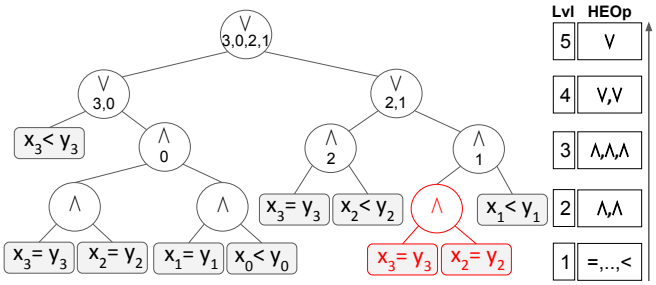
Further, we identify that all bit-by-bit equality checks and *less-than* comparisons can be done in parallel. They become inputs for the subsequent AND and OR gates. Recall that the bit-by-bit equality checks are performed using homomorphic additions. From the evaluation tree, we extract all parallelizable computations at each level into a vector for easier access to different computations, as shown on the right of Fig. 4(b). With these observations, evaluating SecComp is highly parallel. The results are cached in a lookup table. The cached values are reused when repeated evaluation of the same terms is called. For example, the second call to evaluate $(x_3 = y_3) \wedge (x_2 = y_2)$ is skipped.

All boolean functions can also be evaluated in parallel since they are independent. The multiplicative depth significantly decreases in this parallelized approach. For efficiency, the evaluation tree can be created once before the start of the protocol as a set of instructions to evaluate any two inputs of a specific bit-length ℓ. Using this approach facilitates the extension to use multi-cloud instances to evaluate multiple decision nodes in parallel.

Although the approach still requires the evaluation of ℓ bit-by-bit less-than comparison checks (rounded boxes in Fig. 4(b)), the evaluation tree now requires $\log_2 \ell$ consecutive multiplications for the OR gates and $\frac{\ell}{2}$ for the AND gates. Therefore, the multiplicative depth of the parallel SecComp for ℓ-bit is $(\log_2 \ell + \frac{\ell}{2} + \ell)$. The $(\ell - 1)$ bit-by-bit equality checks are evaluated using homomorphic addition; therefore, their evaluation does not significantly affect the multiplicative depth. For example in Fig. 4(b), the evaluation tree for comparing 4-bit inputs requires 4 bit-by-bit less-than comparison checks, 2 levels of OR logic gates and 2 levels of AND logic gates. Thus, the multiplicative

3	$(x_3 < y_3) \vee$
2	$(x_3 = y_3) \wedge (x_2 < y_2) \vee$
1	$(x_3 = y_3) \wedge (x_2 = y_2) \wedge (x_1 < y_1) \vee$
0	$(x_3 = y_3) \wedge (x_2 = y_2) \wedge (x_1 = y_1) \wedge (x_0 < y_0)$

(a) SecComp protocol for 4-bit inputs



(b) Evaluation tree and vector

Fig. 4: An example instantiation of SecComp (Eqn. 1) for 4-bit inputs, and the corresponding evaluation tree and vector

depth is 8 compared to the sequential approach which requires 10 consecutive homomorphic multiplications.

Once the evaluation of all decision nodes is done in parallel, we got the bits $b_0, b_1, \ldots, b_{m-1}$. Our system evaluates the decision tree by plugging in these bits into a polynomial [5], as discussed in Sec. 4.2. The evaluation of this polynomial outputs a single ciphertext which decrypts to the classification result, such that $\mathcal{T}_i(X) = v$. We employ a similar speedup technique for evaluating SecComp to reduce the multiplicative depth and to parallelize homomorphic operations. The system evaluates the polynomial ℓ times, one for each bit of v, to assemble v in the bit-wise format for our secure counting protocol to be discussed. Lower multiplicative depth allows us to choose smaller parameters for the SWHE scheme, hence smaller ciphertext size.

Our approach, in its worst, will only send unique class labels with their counts. Alternative methods for evaluating the tree [6], [7] require the server to send the entire list of class labels to the client who extracts the result through an OT protocol.

5.6 Combining Results of a Forest

After evaluating individual trees, we will get a set of decision tree results $\{\mathcal{T}_i(X)\}$. For a *regression* problem, these results are numerical values that we can compute the random forest result; for example, calculating the average of all values. This can be achieved by homomorphically adding the values output from decision tree evaluation $\mathcal{F}(X) = \frac{1}{n}\sum_{i=1}^{n}(\mathcal{T}_i(X))$. The evaluator then sends the sum to the client who decrypts and divides it by the number of decision trees in the random forest.

For a *classification* problem, we normally want to know either the count of each class label or the class label that has the maximum count. To achieve this, we propose a secure counting algorithm — SecCount, which counts the number of times each unique class label has been chosen as the result.

We associate a vector of counters $\{z_1, z_2, \ldots, z_n\}$ (initially all zeros) with the vector of class labels $V = \{v_1, v_2, \ldots, v_n\}$. For each evaluation result $v_j \in \{\mathcal{T}_i(X)\}$, we perform a matching algorithm using the bit-wise representations of v_j and v_k:

$$z_j = (v_{j,\ell-1} \odot v_{k,\ell-1}) \wedge (v_{j,\ell-2} \odot v_{k,\ell-2}) \wedge \cdots \wedge (v_{j,0} \odot v_{k,0})$$

where $\wedge$ is logical AND, $\odot$ is logical XNOR, and $z_j = 1$ if and only if $v_j = v_k$; otherwise $z_j = 0$. We then calculate the sum of $z_k = \sum_{j=0}^{n-1} z_j$ for each v_k. In other words, z_k contains the total count corresponding to the number of decision trees which outputs v_k as the evaluation result. To maintain correctness of addition with respect to binary message space, we use a regular binary full-adder circuit to perform the addition. The result of the addition is the count encoded in bits and can be decoded by performing the equation $\sigma_{i=0}^{\ell-1}(z_{k,i}2^i)$. Note that the bit length for the count can be set to be $\log_2(n)$ instead of ℓ for space efficiency.

After the counting, a simple approach to return the random forest evaluation result is to have the evaluator return directly the two vectors for the client to decrypt and obtains the counts for each class. This incurs a high communication overhead.

Alternatively, if we only want to provide the class label with the maximum count, the evaluator can permute the vector of counters and the vector of class labels using the same seed, such that v_k and z_k are correlated. This prevents the client from learning the count of each specific class. Then, the evaluator sends the vector of encrypted counters to the client who will decrypt and send back an encrypted *permuted* index corresponding to the maximum count as the input to a 1-out-of-n OT protocol to retrieve the class label. This approach is considered as more efficient despite the use of OT. As a result of the random forest evaluation, we get either $\mathcal{F}(X) = \{(v_1, z_1), (v_2, z_2), \ldots, (v_n, z_n)\}$ or $\mathcal{F}(X) = v$, which are encrypted under a joint key of multiple parties.

5.7 Decrypting the Classification Result

After the evaluation, the evaluator produces results that are encrypted under $\bar{\mathsf{pk}} = \{\mathsf{pk}_\mathcal{C}, \mathsf{pk}_\mathcal{M}\}$. Recall that the results are extended ciphertexts, where each is in the form $\bar{c} = \{(c_{\mathcal{C},0}, c_{\mathcal{C},1})|(c_{\mathcal{M},0}, c_{\mathcal{M},1})\}$. The client can only decrypt $(c_{\mathcal{C},0}, c_{\mathcal{C},1})$. Hence, model owners, who have shares of the secret key $s_\mathcal{M}$, have to help in decrypting $c_\mathcal{M} = (c_{\mathcal{M},0}, c_{\mathcal{M},1})$ so the client can decrypt the evaluation result.

Assume that we have an evaluation result $[[v_k]]_{\bar{\mathsf{pk}}} = \{c_\mathcal{C}|c_\mathcal{M}\} = \{(c_{\mathcal{C},0}, c_{\mathcal{C},1})|(c_{\mathcal{M},0}, c_{\mathcal{M},1})\}$. We need to construct an element ρ using the part $c_{\mathcal{M},0}$ to decrypt as described in Sec. 2.2. In our protocol, the semi-honest evaluator sends $c_{\mathcal{M},0}$ to all model owners. Each model owner $\mathcal{M}_i$ will construct $\rho_{\mathcal{M}_i} = c_{\mathcal{M},0}s_{\mathcal{M}_i} + te_{\mathcal{M}_i}$, where e_{M_i} is a large smudging noise (Appendix B.2) for hiding $s_{\mathcal{M}_i}$, and return it back to the evaluator. After collecting the responses from all model owners, the evaluator sends the extended encrypted results to the client along with the aggregated value of $\rho = \sum_{i=1}^{N}\rho_i = c_{\mathcal{M},0}s_\mathcal{M} + te_\mathcal{M}$. The client then computes:

$$\begin{aligned} v_k &\approx \langle c_\mathcal{C}, s_\mathcal{C}\rangle + (c_{\mathcal{M},1} - \rho) \\ &= (c_{\mathcal{C},1} - c_{\mathcal{C},0}s_\mathcal{C}) + (c_{\mathcal{M},1} - (c_{\mathcal{M},0}s_\mathcal{M} + te_\mathcal{M})) \\ &= (v_\mathcal{C} + te_\mathcal{C}) + (v_\mathcal{M} + te_\mathcal{M}) \\ &= (v_\mathcal{C} + v_\mathcal{M}) + (te_\mathcal{C} + te_\mathcal{M}) \\ &= v_k + te \approx v_k \mod t \end{aligned}$$

6 Evaluation and Discussion

We analyze the correctness, security, and complexity of the proposed system. We also validate the design through a number of empirical studies using synthetic and real-world datasets.

6.1 Correctness Analysis

The proposed system discussed in Sec. 5 correctly evaluates the multi-party random forest. The process of evaluating a random forest is deterministic by its inputs, the feature vector X and the random forest $\mathcal{F}$. The core of decision tree and random forest evaluation is the SecComp protocol for evaluating the boolean function at each decision node, and the SecCount protocol for computing a vector of counts that are associated with the vector of class labels. As shown in previous sections, both computations are made up of logic gates which can be translated into additions and multiplications. Hence, the proposed system directly evaluates the random forest using homomorphic addition and multiplication without having any side-effect that affects the evaluation results.

6.2 Security Analysis

Our proposed system privately evaluates the random forest and is secure against semi-honest adversaries under the assumption that the used encryption schemes are secure. While the use of AES introduces another assumption, it lowers the communication overhead as can be observed in Fig. 6a; otherwise, SWHE can be used directly. The AES ciphertexts are transformed to SWHE ciphertexts under the individual client's key $\mathsf{pk}_{\mathcal{C}}$ and the model owners' joint key $\mathsf{pk}_{\mathcal{M}}$ by the cloud. MKHE provides semantic security unless the adversary compromised all the secret keys.

Beside their own inputs, the view of each model owner in the decryption protocol consists of only the component $c_{\mathcal{M},0}$ of a ciphertext that is encrypted under the joint key derived from $\mathsf{pk}_{\mathcal{M}}$. For the decryption component to be sent to the cloud, the usage of the $\mathsf{sk}_{\mathcal{M}_i}$ is protected under the RLWE assumption. Moreover, the ciphertext $(c_{\mathcal{M},0}, c_{\mathcal{M},1})$ only encrypts a partial $\mu_{\mathcal{M}}$ of the evaluation result, but does not contain information about $\mu_{\mathcal{C}}$ of the client. Therefore, the security against an adversarial model owner is ensured by the encryption and by the fact that the adversary does not have access to all ciphertext components.

Lastly, the view of a client consists of his feature vector X and unique class labels with their associated counts, which are already computed on the cloud. The probability of each models contribution to the evaluation result is uniformly distributed over the counts. Therefore, an adversarial client will not be able to learn information about individual models used in the random forest.

6.3 Complexity Analysis

With the notations in Table 2, we analyze the computation and communication complexities of the proposed system. Especially, we focus on the computation complexity of the core operations that rely on homomorphic multiplications, since an extra step is required to reduce the dimension of the resulting ciphertext. Fig. 5 shows a comparison of homomorphic operations implemented using HElib [22] on ciphertext C and plaintext P. Except $C+P$ and $C \times P$, other operations are performed over ciphertexts. Note, the logic gates, OR and AND, contain homomorphic multiplications. Hence, they are significantly slower than addition and subtraction that are used in NOT, XOR, and XNOR.

Upon receiving the client's request, the cloud evaluates each decision tree in the forest as discussed in Sec. 5. For each of the τ decision trees, the cloud requires a multiplicative depth (number of consecutive homomorphic multiplications) of $3\ell - 2$ to sequentially evaluate the boolean function SecComp at each of the m decision nodes in the tree. For parallel evaluation, the multiplicative depth decreases to $(\log_2 \ell + \frac{\ell}{2} + \ell)$. Evaluating

TABLE 2: Notations for complexity analysis

Notation	Description
τ	num. of decision trees
δ	max. depth of decision tree
m	num. of decision nodes; $(m+1)$ leaf nodes
φ	num. of unique class labels known in the system
ℓ	bit length of features, thresholds, class labels

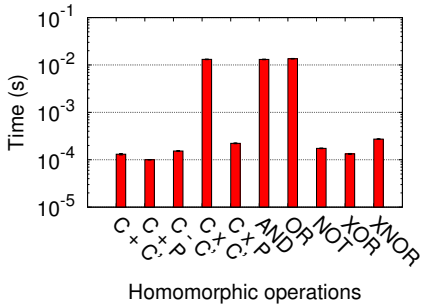


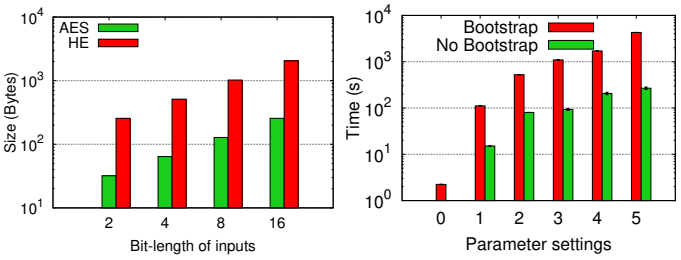
Fig. 5: A comparison of homomorphic operations

the polynomial representation of a decision tree with depth δ requires a multiplicative depth of δ in sequential or $\log_2 \delta$ when multiplying in pairs. Note, we also include the final multiplication with the leaf node, as discussed in Sec. 4.2. The multiplicative depth to evaluate a single decision tree of depth δ is set to $3\ell-2+\delta$ in the sequential testing or $\log_2 \ell + \frac{\ell}{2} + \ell + (\log_2 \delta)$ in the parallel testing. Hence, the sequential evaluation of a decision tree requires at most $\mathcal{O}(m\ell + \delta)$ homomorphic multiplications, while it costs $\mathcal{O}(m(\log_2 \ell + \ell) + \log_2 \delta)$ multiplications in parallel.

To securely count the number each of the φ class labels accumulates based on the evaluation results from τ decision trees, the cloud requires ℓ homomorphic multiplication; the computation complexity for SecCount is $\mathcal{O}(\ell + \varphi + \tau)$.

For communication complexity, we analyze the number of message exchanges between the different parties and the size of the transmitted messages. In the setup phase, when n model owners send their models to the cloud, each round transmits $2m\ell$ AES encrypted thresholds and class labels for a complete tree. Because the size of a block-cipher is significantly smaller than that of SWHE, the use of hybrid encryption offers lower communication overhead. Figure 6a shows the total size of ciphertexts sent to the cloud evaluator, where each bit is individually encrypted under the respective encryption scheme. We have adapted the implementation of homAESdec provided in HElib [22] to evaluate the performance of homomorphic AES decryption, which is performed once when the evaluator first receives the inputs. The evaluation is shown in Fig. 6b considers different HELib-specific parameter sets (indexed from 0 to 5) suggested in HELib and in [17] to support different multiplicative depths and ciphertext packing options. The evaluation compares the performance with and without Bootstrapping. Note that Bootstrapping option has lower multiplicative depth and is suggested to be used if further computations are expected to be performed on the ciphertext [22] as the case in our proposed system.

The model owner also exchanges $2n$ messages with the cloud to generate and send the decryption components. The client exchanges 2 rounds of communication with the cloud, one during the request of evaluation, where the client sends $m\ell$ AES encrypted features, and one SWHE encryption of its own AES secret key.



(a) Size of AES and HE ciphertexts (b) Homomorphic AES decryption

Fig. 6: Empirical results for hybrid encryption

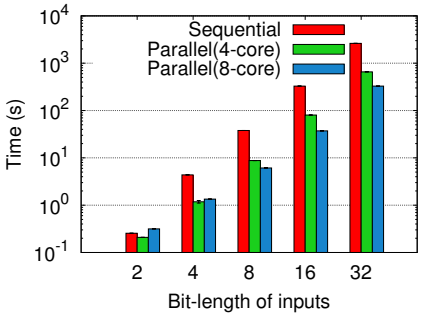


Fig. 7: Empirical results of the SecComp protocol

The other round is when the client receives the evaluation result from the cloud in the form of either 2φ ciphertexts representing each class label with its corresponding count or as one single ciphertext of the class label with the maximum count. The latter option requires an additional round transmitting $\varphi\ell$ counts and performing a 1-out-of-φ OT. A comparison of the proposed system with the state-of-the-art is presented in Table 3.

6.4 Empirical Study

We have developed a proof-of-concept prototype for performance evaluation. Specifically, we focus on the SecComp and SecCount protocols, which are the core of the proposed system. These two protocols contain many homomorphic operations which will dominate the run-time overhead of the overall system. The communication overhead of our system depends on the network factors (such as bandwidth) and can be directly estimated given those factors. Due to the unavailability of a library that implements the multi-key BGV SWHE scheme [11], we defer the multi-key version to future work and focus on other core parts of the proposed solution. The implementation is based on the HElib [22], which implements the BGV SWHE scheme [10] using Number Theory Library (NTL) and GNU Multiple Precision Arithmetic Library (GMP). We run each experiment multiple times on two systems, one with Intel Dual Core i7, 3.1GHz and 16GB RAM, and the other with Intel Xeon, 2.00GHz with 8 CPU cores. We recorded experiments' average time and standard deviation.

We set the security parameter λ of BGV scheme to be 128 bits, which corresponds to a 3072-bit asymmetric key [37]. We set the plaintext modulus $t = 2$, which means the plaintext messages are encoded as binary values as our client's inputs and threshold values are both encoded in binary. The rest of the BGV scheme parameters are set to the defaults [22]. The multiplicative depth L of is configured[4] according to what required by the evaluated circuits in our protocol, for which we derived in Sec. 6.3.

6.4.1 Sub-protocols

First, we evaluate the SecComp protocol and compare the performance of the sequential and parallel versions. We randomize the input values with various bit-lengths in each experiment. Fig. 7 shows the evaluation results in the logarithmic scale. As expected, the running time increases linearly with the length of inputs because more homomorphic multiplications will be required.

As described in Sec. 5.5, we speed up the evaluation of the protocol by pre-computing a lookup table of all leaf nodes and constructing an evaluation tree and vector to facilitate parallel computing with OpenMP library [38]. The reduction in running time is visible in Fig. 7. For a small input size (see 2-bit results), the parallel approach is not significantly faster. That is because building the evaluation tree and assigning tasks to threads using OpenMP introduce additional time that is roughly the same as performing the comparison sequentially. The ratio of speedup is approximately $1{:}\log_2 \ell$ in each experiment, which agrees with the estimated improvement since we perform multiplications in pairs using the evaluation tree.

In another experiment, we study the performance gain when we increase the parallelization from 4-core to 8-core. As shown in Fig. 7, there is a significant improvement when more processor cores are available. Comparing two 16-bit inputs requires around 328 s to complete. The running time decreases to 80 s when using a 4-core system and to 37 s when using an 8-core system. Note, the rest of the experiments are run using an 8-core system.

We also test the performance of SecCount through three different experiments. In the first experiment, we set the number of class labels to 2, mimicking a binary classification problem, and calculate their counters based on one classification result. Fig. 8a shows that the running time increases with the bit-length of the class labels and classification results. For example, the protocol counts labels and results in 31 s if their bit-length is 16, while it only needs 6 s if the length is 8 bits. Similar to SecComp, this is due to the increase of homomorphic multiplications. Hence, we parallelize the evaluation of this protocol using OpenMP. The parallel version achieves approximately a speedup by a factor of $\times 2$ when we calculate the counter of 2 class labels simultaneously.

In the second experiment (Fig. 8b), we investigate the performance for a classification problem that has multiple class labels. Specifically, we vary the number of class labels φ for each experiment and set the number of results to 5φ. We aim to test the performance of the SecCount protocol when the number of inputs (both labels and results) scales up. In this experiment, we fix the bit-length of each input to 8 bits. The third experiment, as shown in Fig. 8c, we consider a different number of classification results but fix the number of class labels to two. From the three experiments, we demonstrate the feasibility of the SecCount protocol and validate the technique to speed up the evaluation process.

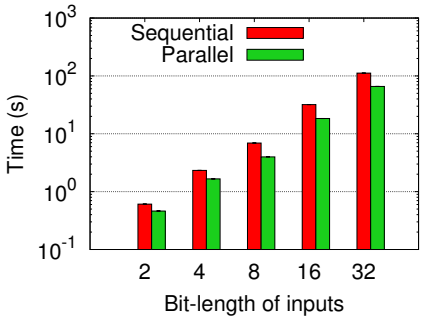
6.4.2 Decision Trees and Random Forests

Next, we investigate the performance of evaluating the entire decision tree, consisting of the secure comparison protocol at each decision node and operating on the polynomial representation of the tree. Our experiments are over *complete* trees since they represent the worst case. There are $m = 2^\delta - 1$ invocation of

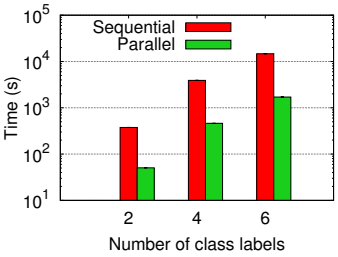
4. The number of levels in the scheme is set as $(20 \times L)$ as suggested in the latest release of the HELib library. https://github.com/shaih/HElib

TABLE 3: A comparison with the recent work on privately evaluating one decision tree: n is the number of feature, m is the number of decision nodes, ℓ is bit-length of input, and δ is the depth of decision tree

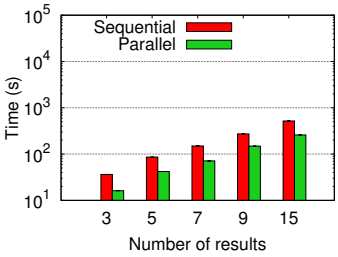
Protocol	HE scheme	Comparison protocol	Computation complexity			Rounds
			Client	Model owner	Cloud	
Bost *et al.* [5]	SWHE + Additive HE	Veugen's protocol [9]	$\mathcal{O}((n+m)\ell)$	$\mathcal{O}(m\ell)$	-	6
Wu *et al.* [6]	Additive HE	Improved DGK [25]	$\mathcal{O}((n+m)\ell+\delta)$	$\mathcal{O}(m\ell+2^\delta)$	-	6
Tai *et al.* [7]	Additive HE	Improved DGK [25]	$\mathcal{O}((n+m)\ell)$	$\mathcal{O}(m\ell)$	-	4
This work	SWHE	SecComp	$\mathcal{O}(n\ell)$	$\mathcal{O}(m\ell)$	$\mathcal{O}((m+n)\ell+\delta)$	2



(a) Varying bit-length (results=1, labels=2)

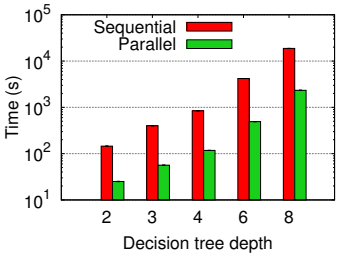


(b) Varying # of labels φ (results=5φ)

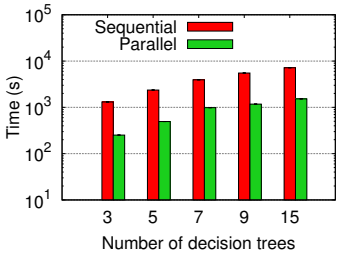


(c) Varying # of results (labels=2)

Fig. 8: Evaluation of different aspects of the SecCount protocol



(a) Evaluation of decision tree



(b) Evaluation of random forest

Fig. 9: Empirical results using synthetic data

the secure comparison protocol. We use randomized values (for features, threshold values, and class labels) of 8-bit long.

Fig. 9a shows the results. The running time grows with the increase of the decision tree depth as the number of decision nodes increases. Correspondingly, this means there will be more invocations of SecComp. The performance achieves a speedup of factor $\times 7$ when evaluating a tree in parallel. The improvement is largely due to our parallel evaluation of SecComp and processing of the polynomial representation of the tree discussed in Sec. 5.5.

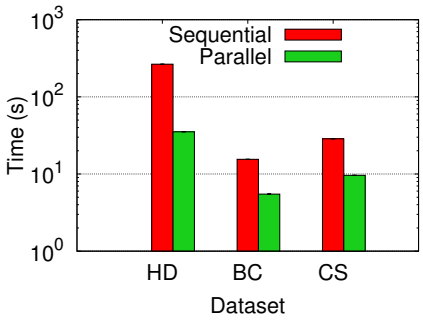
We also study the performance of evaluating a forest of trees which have a depth of $\delta = 2$. We use synthetic data with bit-length of 8 to construct all the tree in the random forest. We vary the number of trees in each experiment. Fig. 9b shows the performance. The most expensive part of evaluating the random forest is SecComp at each decision node of each tree. For *complete* trees, it takes $\tau(2^\delta - 1)$ SecComp invocations where τ is the number of trees. In this experiment, no parallelism has been exploited to support concurrent evaluation of decision trees. The parallelization is only applied to speed up the building blocks, i.e., SecComp at each decision node, the polynomial evaluation, and SecCount. There is significant speedup (in the ratio of 1:4.5) between the sequential and parallel evaluation of the random forest. The parallel evaluation of random forest with 3 decision trees requires around 250 s. In total, it is approximately one-fifth of the time 1307 s needed in the sequential evaluation.

We validate the feasibility and performance of our system using real-world datasets. Specifically, we use the Heart Disease (HD), Breast Cancer (BC), and Credit Screening (CS) datasets from the UCI Machine Learning Repository [39]. For each dataset, we split that data into training data (50%) and testing data (50%) and train a collection of decision trees using FFTrees (fast-and-frugal decision trees) [40]. Note that we verify the ground truth using the plaintext version in each of the following experiments. We achieve an accuracy of 100% in our classification results since the encryption scheme does not affect any computation.

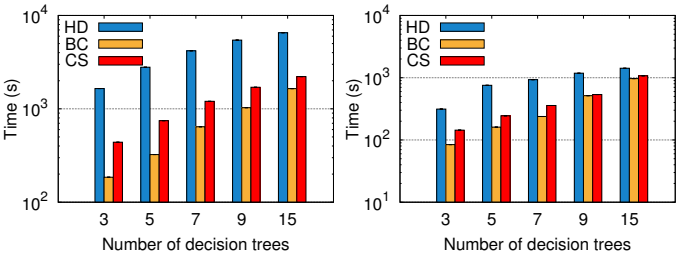
Similar to the empirical study using synthetic data, we first randomly pick one trained decision tree from each dataset. We then measure the running time of the homomorphic evaluation of the three representative trees. In these experiments, we choose the bit-length 8-bit for HD inputs and the bit-length 4-bit for the BC and CS inputs. Fig. 10a shows the results. Note that each dataset has different characteristics (i.e., the tree depth δ and the number of decision nodes m). Clearly, the running time is correlated to m and the number of SecComp invocations.

Next, we randomly pick some number of decision trees from the collection of trained trees to form a random forest of a specific size. Table 4 summarizes the characteristics of the created random forests for each dataset, which include the number of decision trees in the forest, the number of decision nodes in the entire forest, the maximum depth of each tree, and the bit-length of all inputs.

Figures 10b and 10c show the performance of the sequential and parallel evaluation as the number of trees varies. Overall, these results agree with the findings in earlier experiments. In general, the more decision nodes, the longer the running time is, primarily, due to the number of SecComp instances. Also, the bit-length of inputs in the HD dataset is 8-bit, which incurs a longer running time as shown in Fig. 7. For a random forest of size 15, the forest for HD takes around 6532 s to evaluate sequentially, and approximately 1420 s in parallel. The BC forest takes 1645 s, and the CS forest takes 2213 s to be evaluated sequentially. In the parallel setting, they run in 668 s and 1073 s respectively.



(a) Evaluation of one decision tree for 3 datasets: i) HD ($\delta = 4, m = 4$), ii) BC ($\delta = 2, m = 2$), and iii) CS ($\delta = 3, m = 3$)



(b) Varying # of trees (Sequential) (c) Varying # of trees (Parallel)

Fig. 10: Empirical results using real-world datasets

TABLE 4: The numbers of decision nodes in the random forests we created for different number of trees and datasets

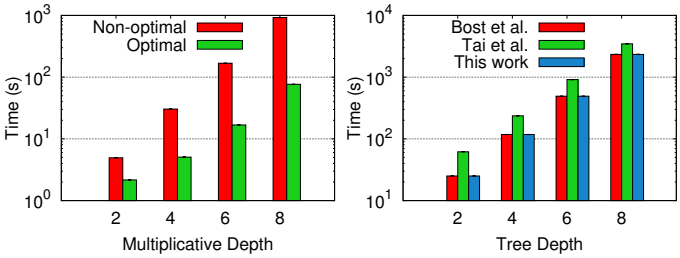
Dataset / Number of Trees τ	3	5	7	9	15
HD ($\delta = 4, \ell = 8$)	10	16	22	29	49
BC ($\delta = 2, \ell = 4$)	6	10	14	18	30
CS ($\delta = 4, \ell = 4$)	11	17	25	31	53

6.4.3 Comparison to Existing Works

We also compare the performance of our protocol with different state-of-the-art protocols for secure evaluation of decision tree [5], [7]. Most related works are focused on a two-party setting, whereas our design focuses on a collaborative setting which requires multi-key support. For a fair comparison, we compare the core techniques across the best-known works and consider a special case where the model owner holds a random forest consisting of one decision tree. The tree is of depth δ (a variable in our experiments) with $(2^\delta - 1)$ decision nodes and 2^δ leaves.

In our experiment setup, we use SWHE to encrypt model information (i.e., thresholds and class labels) and client's inputs. We use our non-interactive SecComp protocol for evaluating the boolean function at each decision node. We focus our testing on two main approaches used in literature for tree traversal, the OT approach [6], [7] and the tree polynomial evaluation [5]. Among the OT approaches, we implemented the path cost [7] because it is the most efficient among the two works. We compare it to the polynomial approach, which is used in both [5] and in our work.

In Fig. 11a, we show the performance of evaluating a decision tree as a polynomial. Recall that a tree of depth δ can be represented as a polynomial with a multiplicative depth of δ (Sec. 4.2). Evaluating δ consecutive multiplications is non-optimal, hence we evaluate them in-pairs instead to achieve optimal performance. Bost *et al.* [5] briefly showed the impact of this approach for evaluating decision trees that have a small multiplicative depth $\delta = 4$. Here, we further investigate the impact of this optimization with varying depths. As shown in Fig. 11a, the optimal approach



(a) Analysis of multiplicative depth (b) Evaluation of decision tree

Fig. 11: Empirical results of comparison with state-of-the-art

performs approximately $7\times$ better compared to the non-optimal one and the improvement increases as the decision tree depth ($\delta = 4$, $\delta = 6$, and $\delta = 8$ achieve factors of $1:6$, $1:10$, and $1:12$ speedup respectively). For example, it takes around 168 s to evaluate a polynomial of depth $\delta = 6$, while it only takes 16 s to evaluate the same polynomial in-pairs.

We also compare the performance of evaluating a decision tree using the optimized polynomial approach in Bost *et al.* [5] and our protocol, and the path cost approach in Tai *et al.* [7]. As can be seen in Fig. 11b, the evaluation with the polynomial approach outperforms the path cost approach of Tai *et al.* [7] across all different depths. For example, it takes around 236 s to evaluate a tree of depth $\delta = 4$ using the path cost approach and 117 s using our approach. Moreover, the path cost approach [7] sends back 2^δ pairs of encrypted data for each decision tree to the client. In contrast, our work sends back in the "worst case" φ pairs of encrypted data. Note that the number of labels is less than the number of all leaf nodes in the random forest (i.e., $\varphi \ll 2^\delta$).

7 Conclusion and Future Work

We proposed a semi-honest protocol that supports privacy-preserving evaluation of random forest in an outsourcing setting. In particular, we integrated hybrid homomorphic encryption and multi-key encryption to reduce the communication overhead in homomorphic computations on data encrypted using different keys. We also developed two sub-protocols. The first is a secure counting protocol that goes beyond the state-of-the-art approaches to compute the result of the random forest. Our secure comparison protocol achieves a lower round complexity compared to existing work. We discussed optimization techniques based on the execution path to speed up the evaluations of the comparison protocol and the polynomial associated with the decision tree.

With these new features and improvements, we also developed a proof-of-concept prototype for asserting the feasibility of our protocol through experiments. We demonstrated that collaborative evaluation of multiple models in an outsourcing setting is feasible.

One potential improvement of our proposed solution is to allow operation on floating-point numbers instead of integers. Computing homomorphically on floating-point numbers can be handled using the CKKS scheme [41], which is implemented in most of the recent HE libraries, such as HELib [22] and SEAL [42]. Other future works include extending our protocol for security against malicious adversaries and exploring further techniques to speed up the homomorphic operations.

ACKNOWLEDGMENT

The authors would like to thank Krish Sunil Rohra for his valuable contributions in protocol implementation and evaluation.

References

[1] T. Hofmann and J. Basilico, "Collaborative machine learning," in, M. Hemmje, C. Niederée, and T. Risse, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 173–182.

[2] T. Fox-Brewster, *120 million american households exposed in 'massive' consumerview database leak*, https://www.forbes.com/sites/thomasbrewster/2017/12/19/120m-american-households-exposed-in-massive-consumerview-database-leak, Dec. 2017.

[3] D. Linthicum, *Safer but not immune: Cloud lessons from the equifax breach*, https://www.infoworld.com/article/3225479/cloud-computing/safer-but-not-immune-cloud-lessons-from-the-equifax-breach.html, Sep. 2017.

[4] U.S. Department of Health & Human Services, *Notice to the secretary of HHS breach of unsecured protected health information*, https://ocrportal.hhs.gov/ocr/breach/breach_report.jsf.

[5] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Network and Distributed System Security Symposium (NDSS)*, 2015.

[6] D. J. Wu, T. Feng, M. Naehrig, and K. Lauter, "Privately evaluating decision trees and random forests," *Proceedings on Privacy Enhancing Technologies*, vol. 4, pp. 1–21, 2016.

[7] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. S. M. Chow, "Privacy-preserving decision trees evaluation via linear functions," in *European Symposium on Research in Computer Security*, 2017, pp. 494–512.

[8] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Australasian Conference on Information Security and Privacy*, Springer, 2007, pp. 416–430.

[9] T. Veugen, "Comparing encrypted data," Delft University of Technology and TNO Information and Communication Technology, Tech. Rep., 2011.

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science Conference (ITCS)*, ACM, 2012, pp. 309–325.

[11] L. Chen, Z. Zhang, and X. Wang, "Batched multi-hop multi-key fhe from Ring-LWE with compact ciphertext extension," in *Theory of Cryptography Conference*, Springer, 2017, pp. 597–627.

[12] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM*, vol. 60, no. 6, p. 43, 2013.

[13] B. Applebaum, D. Cash, C. Peikert, and A. Sahai, "Fast cryptographic primitives and circular-secure encryption based on hard learning problems," in *Advances in Cryptology – CRYPTO*, Springer, 2009, pp. 595–618.

[14] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Advances in Cryptology – EUROCRYPT*, Springer, 2012, pp. 483–501.

[15] M. Clear and C. McGoldrick, "Multi-identity and multi-key leveled FHE from learning with errors," in *Advances in Cryptology – CRYPTO*, Springer, 2015, pp. 630–656.

[16] C. Peikert and S. Shiehian, "Multi-key FHE from LWE, revisited," in *Theory of Cryptography Conference*, Springer, 2016, pp. 217–238.

[17] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*, Springer, 2013, pp. 75–92.

[18] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology – EUROCRYPT*, 2009, pp. 24–43.

[19] J. Basilakis and B. Javadi, *Efficient parallel binary operations on homomorphic encrypted real numbers*, Cryptology ePrint Archive, Report 2018/201, https://eprint.iacr.org/2018/201, 2018.

[20] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes FV and YASHE," in *AfricaCrypt*, Springer, 2014, pp. 318–335.

[21] M. Naehrig, K. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" In *ACM workshop on Cloud Computing Security Workshop*, 2011, pp. 113–124.

[22] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology – CRYPTO*, Springer, 2014, pp. 554–571.

[23] M. O. Rabin, *How to exchange secrets with oblivious transfer*, IACR Cryptology ePrint Archive 2005/187, 2005.

[24] *CDC Diseases and Conditions*, https://www.cdc.gov/diseasesconditions/index.html, Accessed: 2018-02-15.

[25] T. Veugen, "Improving the DGK comparison protocol," in *International Workshop on Information Forensics and Security (WIFS)*, IEEE, 2012, pp. 49–54.

[26] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *Privacy Enhancing Technologies Symposium*, Springer, 2009, pp. 235–253.

[27] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[28] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology – EUROCRYPT*, Springer, 1999, pp. 223–238.

[29] M. Joye and F. Salehi, "Private yet efficient decision tree evaluation," in *IFIP Annual Conference on Data and Applications Security and Privacy*, Springer, 2018, pp. 243–259.

[30] A. Tueno, F. Kerschbaum, and S. Katzenbeisser, "Private evaluation of decision trees using sublinear cost," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 1, pp. 266–286, 2019.

[31] F. Emekçi, O. D. Sahin, D. Agrawal, and A. El Abbadi, "Privacy preserving decision tree learning over multiple parties," *Data & Knowledge Engineering*, vol. 63, no. 2, pp. 348–361, 2007.

[32] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of biomedical informatics*, vol. 50, pp. 234–243, 2014.

[33] S. Hu, Q. Wang, J. Wang, S. S. M. Chow, and Q. Zou, "Securing fast learning! Ridge regression over encrypted big data," in *TrustCom*, 2016, pp. 19–26.

[34] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *IEEE Sympposium on Security and Privacy*, 2017, pp. 19–38.

[35] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Computer and Communications Security*, 2017, pp. 619–631.

[36] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *USENIX Security*, 2016, pp. 619–636.

[37] M. Backes, P. Berrang, M. Bieg, R. Eils, C. Herrmann, M. Humbert, and I. Lehmann, "Identifying personal DNA methylation profiles by genotype inference," in *IEEE Symposium on Security and Privacy*, 2017, pp. 957–976.

[38] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[39] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[40] N. D. Phillips, H. Neth, J. K. Woike, and W. Gaissmaier, "FFTrees: A toolbox to create, visualize, and evaluate fast-and-frugal decision trees," *Judgment and Decision Making*, vol. 12, no. 4, pp. 344–368, 2017.

[41] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT*, Springer, 2017, pp. 409–437.

[42] *Simple Encrypted Arithmetic Library (release 3.1.0)*, https://github.com/Microsoft/SEAL, Microsoft Research, Redmond, WA., Dec. 2018.

APPENDIX A
THE BGV SCHEME

Our work uses the Brakerski-Gentry-Vaikuntanathan (BGV) SWHE scheme [1] which allows an arbitrary number of additions but *limited* consecutive multiplications. The scheme consists of the following algorithms ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{EvalAdd}, \mathsf{EvalMult}$):

- $\mathsf{BGV.KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm to generate public and private keys. Sample a small element $s \xleftarrow{\$} \chi_k$ such that secret key $\mathsf{sk} = s$, and sample a small noise element $e \xleftarrow{\$} \chi_e$, where χ_k and χ_e are two Gaussian distributions over R_q. Also uniformly sample $A \xleftarrow{\$} R_q$. The public key is $\mathsf{pk} = (A, As + te)$.
- $\mathsf{BGV.Enc}(\mathsf{pk}, \mu) \rightarrow c$: A probabilistic algorithm inputs a plaintext message $\mu \in R_t$ and a public key $\mathsf{pk} = (A, As + te)$. Uniformly sample a random number $\gamma \xleftarrow{\$} R_t$ and encrypt the message μ as $c = (c_0, c_1) \in R_q^2$ where $c_0 = \gamma A$ and $c_1 = \gamma(As + te) + \mu$. For clarity, we omit the γ associated with the noise γte and write $c_1 = \gamma As + te + \mu$. Note, we represent ciphertext with public key elements which is an encryption of the secret key s with some noise e. In the BGV scheme, security relies on the initial noise e and the random element γ [1]. New γ should be chosen for every encryption to ensure semantic security.
- $\mathsf{BGV.Dec}(c, \mathsf{sk}) \rightarrow \mu$: A deterministic algorithm inputs a ciphertext $c = (c_0, c_1) \in R_q^2$ encrypted under pk and the corresponding secret key sk. Decrypy by computing $\tilde{\mu} = c_1 - c_0 s$. The decryption of a ciphertext in the BGV scheme is correct if and only if $(\tilde{\mu} \bmod t = \mu)$.
- $\mathsf{BGV.EvalAdd}(c, c') \rightarrow c_{\mathsf{add}}$: A deterministic algorithm adds two ciphertexts $c = (c_0, c_1) = (\gamma A, \gamma As + \mu + te)$, $c' = (c'_0, c'_1) = (\gamma' A, \gamma' As + \mu' + te')$ and outputs $c_{\mathsf{add}} = ((c_0 + c'_0), (c_1 + c'_1)) \in R_q^2$, where, $(c_0 + c'_0) = (\gamma + \gamma')A$ and $(c_1 + c'_1) = (\gamma + \gamma')As + t(e + e') + (\mu + \mu')$. Decryption is still correct because $(\gamma + \gamma')A$ can be canceled with the secret key s.
- $\mathsf{BGV.EvalMult}(c, c') \rightarrow c_{\mathsf{mult}}$: A deterministic algorithm inputs two ciphertexts $c, c' \in R_q^2$ and outputs the result c_{mult}. Their initial homomorphic multiplication yields

$$\begin{aligned}\tilde{c}_{\mathsf{mult}} &= c \cdot c' = (c_0, c_1) \cdot (c'_0, c'_1) \\ &= (c_0 \cdot c'_0, c_0 \cdot c'_1 + c'_0 \cdot c_1, c_1 \cdot c'_1) \\ &= (c_0, c_1, c_2) \in R_q^3\end{aligned}$$

The additional component c_2 includes a quadratic element s^2 resulted from the multiplication $(c_0 \cdot s + t \cdot e + \mu)(c'_0 \cdot s + t \cdot e' + \mu')$. Not only the error noise grows quadratically, but it also increases the number of ciphertext elements. One thus needs to use *key switching* [1] after each homomorphic multiplication to reduce the dimension and yield a correct ciphertext that is decryptable by the secret key sk. The process requires generating a special set of evaluation keys $\{\mathsf{ek}\}$ which contains the s^2 element. We denote the output of the process by $c_{\mathsf{mult}} = \mathsf{KeySwitch}(\tilde{c}_{\mathsf{mult}}, \{\mathsf{ek}\})$, which encrypts the product of two ciphertexts under the secret key sk. For details about the noise reduction technique for BGV, we refer interested readers to [1].

APPENDIX B
CRYPTOGRAPHIC DEFINITIONS

B.1 Key switching

This technique is applied after homomorphic operations to transform a ciphertext from one under the key s to one under a different key s'. It is also called the *relinearization* step [1] that reduces the dimension after each homomorphic multiplication and yield a normal ciphertext that is decryptable by the secret key s'. This transformation is accomplished with the aid of auxiliary information provided as evaluation key ek which encrypts s under s'. To perform key switching, two essential functions are needed:

- $\mathsf{EvalKeyGen}(s, s')$: Given two keys $s \in R_q^k, s' \in R_q^2$, let $\beta = \lfloor \log q \rfloor$ and compute the powers-of-2 of the old secret key $\tilde{s} = \mathsf{Powersof2}(s) = (2^0 s, 2s, \ldots, 2^\beta s) \in R_q^{k\beta}$. Sample $k\beta$ RLWE instances $(a_i, a_i s' + te'_i)$ and output $\mathsf{ek} = \{(a_i, a_i s' + te'_i + \tilde{s}[i]) \in R_q^2\}_{i=1,\ldots,k\beta}$
- $\mathsf{KeySwitch}(\mathsf{ek}, c)$ Given a ciphertext $c \in R_q^k$ under s and the evaluation key ek, decompose the ciphertext to its binary such that $\tilde{c} = \mathsf{BitDecom}(c) = (u_0, \ldots, u_{\lfloor \log q \rfloor})$ where $c = \sum_{i=0}^{\lfloor \log q \rfloor} (u_i 2^i)$ and output the new ciphertext as $c' = \sum_{i=0}^{k} (\tilde{c}[i]\mathsf{ek}[i]) \in R_q^2$ which is encrypted under the new key s'.

B.2 Smudging Noise

In the threshold decryption protocol, each model owner uses their own secret share to generate a decryption component. If the error terms in these components were small, it may reveal information about the secret shares due to their different distributions. To

make sure that no secret shares can be learned, we need to add larger errors following the Smudging Lemma presented by Asharov *et al.* [2], which states that adding large noise smudges out the small values in the ciphertext.

Lemma B.1 (Smudging Noise [2]). Let $\mathcal{B}_1 = \mathcal{B}_1(\lambda)$, and $\mathcal{B}_2 = \mathcal{B}_2(\lambda)$ be positive integers and let $e_0 \in [-\mathcal{B}_1, \mathcal{B}_1]$ be a fixed integer. Let $e_1 \leftarrow [-\mathcal{B}_2, \mathcal{B}_2]$ be chosen uniformly at random. Then the distribution of e_1 is statistically indistinguishable from that of $e_1 + e_0$ as long as $\mathcal{B}_1/\mathcal{B}_2 = \epsilon$, where $\epsilon = \epsilon(\lambda)$ is a negligible function.

In our work, threshold decryption is held by model owners to produce a decryption component that helps the client decrypting the evaluation result. Each model owner uses their own secret share to produce this component; hence, we apply the smudging lemma to add a large noise to the component such that it prevents leaking information about the secret share.

APPENDIX C
THRESHOLD HOMOMORPHIC ENCRYPTION

One approach to support computation over data encrypted under multiple-key is to use the threshold extension [2] of the BGV scheme. The scheme is *key homomorphic*, which means adding a set of different individual keys produces a valid "joint key". The individual keys then become "partial keys" of the newly created joint key.

Definition C.1 (Threshold Homomorphic Encryption (THE)). Let $P = \{P_1, \ldots, , P_n\}$ be a set of parties. A threshold homomorphic encryption scheme is a tuple of PPT algorithms $\mathsf{THE} = (\mathsf{Setup}, \mathsf{JointkeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$.

- $\mathsf{THE.Setup}(1^\lambda, 1^L) \rightarrow ((\mathsf{pk}_1, \mathsf{sk}_1) \ldots, (\mathsf{pk}_n, \mathsf{sk}_n))$: Given a security parameter λ and a bound on circuit depth L, the setup algorithm outputs a set of n key pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{THE.JointkeyGen}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) \rightarrow \mathsf{pk}^*, \mathsf{ek}^*$: Given the input of n public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, the interactive algorithm outputs a joint public key pk^* and the evaluation key ek^*.
- $\mathsf{THE.Enc}(\mathsf{pk}^*, \mu) \rightarrow c$: Given a joint public key pk^* and a message μ, the encryption algorithm outputs a ciphertext c.
- $\mathsf{THE.Eval}(\mathsf{pk}^*, c, c') \rightarrow c_{\mathsf{eval}}$: Given a joint public key pk^* and two ciphertexts c, c', the evaluation algorithm outputs the evaluated ciphertext c_{eval}.
- $\mathsf{THE.Dec}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n, c) \rightarrow \mu$: Given a set of secret shares $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and a ciphertext c, the interactive decryption algorithm performs decryption and outputs the message μ

Formally, given a set of n public keys $\mathsf{pk}_i = (A, As_i + te_i)$ where $i \in \{1, \ldots, n\}$ and the element $A \in R_q$ is assumed to be shared, the joint public key is $\mathsf{pk}^* = (A, A\sum s_i + t\sum e_i)$. Likewise, given a key pair $(\mathsf{pk}^*, \mathsf{sk}^*)$, the corresponding n partial keys can be produced by dividing the joint secret key into n secret shares $\{s_1, \ldots, s_n\}$, such that $s_i = s^* - \sum_{j=1}^{n} s_j; i \neq j$.

During decryption, each party contributes its partial key s_i by computing $c_0 s_i + te_i$. Then, all parties collaboratively produce a component that contains the sum of all partial secret keys, that is $(c_0 s^* + te^*) = (c_0 \sum s_i + t \sum e_i)$. The message can be decrypted correctly when we compute $c_1 - (c_0 s^* + te^*)$.

Multiplying two ciphertexts results in a ciphertext of dimension increased quadratically. One needs a key switching technique (Appendix B.1) which takes in an evaluation key ek^* to transform the resulting ciphertext. The evaluation key encrypts the powers-of-two of $(s^*)^2$, i.e., $\{2^0(s^*)^2, 2(s^*)^2, \ldots, 2^\beta(s^*)^2\}$ where $\beta = \lfloor \log q \rfloor$.

For more details on the generation of the evaluation key, we refer readers to the setup protocol [2]. Below we quickly review some details. In the multi-key setting, the value $(s^*)^2 = (s^*[k] \cdot s^*[u])$, where $k, u \in \{0, \ldots, |s^*| - 1\}$, has to be homomorphically generated based on the secret shares $\{s_1, \ldots, s_n\}$. Hence, the n parties run a two-round protocol to generate the evaluation key in a threshold manner. In the first round, each party i shares the encryptions $\{2^\ell \cdot s_i[k]\}_{\ell \in \{0,\ldots,\lfloor \log q, i \in \{1,\ldots,n\}, k \in \{0,\ldots,|s^*|-1\}\rfloor\}}$ under s^*. Then in the second round, each party i computes and shares the encryptions $\{2^\ell \cdot s_i[k] \cdot s_j[u]\}_{\ell \in \{0,\ldots,\lfloor q \rfloor, i,j \in \{1,\ldots,n\}, k,u \in \{0,\ldots,|s^*|-1\}\}}$. Combining all encryptions yields the evaluation key $ek^* = \{ek^*_\ell\} = \{2^\ell \cdot s^*[k] \cdot s^*[u]\}$ encrypted w.r.t. s^*.

APPENDIX D
MULTI-KEY BGV SWHE SCHEME

Chen, Zhang, and Wang [3] extended the BGV scheme to a multi-key homomorphic encryption scheme, where each participant has a unique index $i \in \{1, \ldots, n\}$ and holds a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. This new scheme extends a ciphertext linearly in the number of participants' keys, which is predetermined at setup.

Definition D.1 (Multi-Key BGV (MKBGV)). The multi-key BGV scheme is a tuple of PPT algorithms $\mathsf{MKBGV} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Ext}, \mathsf{EvalKeyGen}, \mathsf{Eval}, \mathsf{Dec})$.

- $\mathsf{MKBGV.Setup}(1^\lambda, 1^L, 1^K) \rightarrow \mathsf{pp}$: Given a security parameter λ, a bound on circuit depth L, and a bound on the number of keys, the setup algorithm outputs the public parameters pp.
- $\mathsf{MKBGV.KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{ek}')$: Given the public parameters pp, the key generation algorithm outputs a public key pk, a private key sk, and a, evaluation helper element ek'.
- $\mathsf{MKBGV.Enc}(\mathsf{pk}, \mu) \rightarrow c$: Given a public key pk and a message μ, the encryption algorithm outputs a ciphertext c.
- $\mathsf{MKBGV.Ext}(\{\mathsf{pk}_1, \cdot, \mathsf{pk}_n\}, c) \rightarrow \bar{c}$: Given a set of public keys $\mathsf{pk}_1, \cdot, \mathsf{pk}_n$, and a ciphertext c, the output is the extended ciphertext $\bar{c}$ under the concatenated public key $\bar{\mathsf{pk}}$.
- $\mathsf{MKBGV.Eval}(\bar{pk}, c) \rightarrow c_{\mathsf{eval}}$: Given a concatenated public key $\bar{\mathsf{pk}}$ and two ciphertexts $\bar{c}, \bar{c}'$, the evaluation algorithm outputs the evaluated ciphertext $\bar{c}_{\mathsf{eval}}$.
- $\mathsf{MKBGV.Dec}(\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}, \bar{c}) \rightarrow \mu$: Given a set of concatenated secret shares $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_n\}$ and an extended ciphertext $\bar{c}$, the interactive decryption algorithm performs decryption and outputs the message μ

A message μ is encrypted under a public key pk_i following the standard BGV encryption (Appendix A), outputting a ciphertext $[\mu]_i = c_i = (c_{i,0}, c_{i,1}) \in R_q^2$. Each ciphertext is associated with an ordered set I that stores the indices of participants, indicating the ciphertext is encrypted under their keys. For example, a fresh ciphertext encrypted under pk_1 is associated with $I = \{1\}$. Extending this ciphertext to a set of other keys $\{\mathsf{pk}_2, \ldots, \mathsf{pk}_n\}$ yields a concatenated n sub-vectors $\bar{c} = (c_1 | c_2 | \ldots | c_n) \in R_q^{2n}$ such that c_i equals to the ciphertext of participant i if $i \in I$, otherwise, $c_i = 0$. The associated set is updated to include the new indices, such that $I = \{1, 2, \ldots, n\}$. We denote $[\cdot]$ as a standard BGV ciphertext and $[[\cdot]]$ as an extended BGV ciphertext.

Decryption of an extended ciphertext $\bar{c}$ that is encrypted under $\bar{\mathsf{pk}} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ requires the concatenated secret keys $\bar{s} =$

$\{s_1|\ldots|s_n\}$. All n participants collaborate to decrypt the message as $\langle \bar{c}, \bar{s} \rangle = \sum_{i=1}^{n} \langle c_i, s_i \rangle = te + \mu \approx \mu \mod t$.

Similar to the standard BGV scheme, the multi-key scheme requires an evaluation key to perform key switching after homomorphic computations. The evaluation key is generated based on the different keys that encrypt the given extended ciphertext. Since the set of keys can be changed during computations, the evaluation key cannot be pre-generated. Instead for secret key s_i, an evaluation helper element ek'_i is generated at key setup. This helper element encrypts the powers-of-two of the secret key s_i and the randomness values using a ring-variant of GSW encryption [3], [4]. Before performing homomorphic computation on the extended ciphertext $\bar{c}$, which is encrypted under $\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$, the corresponding evaluation helper elements $\{\mathsf{ek}'_1, \ldots, \mathsf{ek}'_n\}$ are used to generate the evaluation key ek. Each helper element ek'_i is extended to the other keys $\{s_j\}_{j \in \{1,\ldots,n\}, j \neq i}$. Then, the extended values are multiplied to produce the evaluation key that encrypts the concatenated secret key $(\bar{s})^2$.

References

[1] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Innovations in Theoretical Computer Science Conference (ITCS)*, ACM, 2012, pp. 309–325.

[2] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Advances in Cryptology – EUROCRYPT*, Springer, 2012, pp. 483–501.

[3] L. Chen, Z. Zhang, and X. Wang, "Batched multi-hop multi-key fhe from Ring-LWE with compact ciphertext extension," in *Theory of Cryptography Conference*, Springer, 2017, pp. 597–627.

[4] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology–CRYPTO 2013*, Springer, 2013, pp. 75–92.