SEEK: model extraction attack against hybrid secure inference protocols

Si Chen*[1] and Junfeng Fan†[2]

[1,2]Open Security Research, Shenzhen, China

Abstract

Security concerns about a machine learning model used in a prediction-as-a-service include the privacy of the model, the query and the result. Secure inference solutions based on homomorphic encryption (HE) and/or multiparty computation (MPC) have been developed to protect all the sensitive information. One of the most efficient type of solution utilizes HE for linear layers, and MPC for non-linear layers. However, for such hybrid protocols with semi-honest security, an adversary can malleate the intermediate features in the inference process, and extract model information more effectively than methods against inference service in plaintext. In this paper, we propose SEEK, a general extraction method for hybrid secure inference services outputing only class labels. This method can extract each layer of the target model independently, and is not affected by the depth of the model. For ResNet-18, SEEK can extract a parameter with less than 50 queries on average, with average error less than 0.03%.

1 Introduction

For a machine learning model used in a prediction-as-a-service (PaaS) setting, the model provider usually is concerned about the privacy of the deployed model. Revealing the model information will enable a user to develop his own model. In addition, the model information can be reverse-engineered to reveal its training data [1, 2], or enable an attacker to fabricate adversarial samples [3]. On the other hand, users of PaaS may have privacy concerns about the input data, and do not want to upload the input in plaintext to a service hosted by the model provider. Thus neither the server side nor the client side is a satisfactory place to perform the model inference computation.

To solve this dilemma, secure inference protocols are proposed, which enables the client to query a model deployed in a remote server, while preventing the client and server to learn any additional information. Secure inference solutions are based on homomorphic encryption (HE), multiparty computation (MPC), or both families of techniques. Solutions based on homomorphic encryption suffer from limitation of the practical FHE schemes. The levelled, and relatively efficient FHE schemes, including BFV, BGV, and CKKS, support fixed number of multiplications without bootstrapping. By replacing the activation functions with polynomial functions, the levelled FHE schemes can compute both the linear layers and non-linear layers, but cannot support multiplication depth needed by a deep neural network with more than 3 or 4 layers [4, 5]. On the other hand, the most efficient secure inference protocol based on MPC either use garbled circuit and generally incur higher communication cost [6, 7, 8], or require three non-colluding parties [9, 10, 11], which is an additional requirement not readily satisfied in practice.

*si.chen@osr-tech.com

†fan@osr-tech.com

To perform secure inference for deep neural networks, while utilizing the efficiency of levelled FHE schemes, hybrid solutions based on FHE and MPC emerged [12, 13, 14, 15, 16]. The linear part of a neural network, which contains the majority of computation cost, is processed by a FHE scheme, while the non-linear part is process by a MPC scheme. Between linear layers and non-linear layers, a pair of protocols are performed to transfer the internal features between encrypted form and secret-shared form.

However, these hybrid secure inference solutions assume semi-honest participants. Such assumption is not guaranteed in real scenarios. We observe that by considering malicious behavior, the client can secretly shift the internal features during inference, and observe its effect on the final output of the model. With this additional opportunity of changing the intermediate data, in this paper we propose a general model extraction method called SEEK (Safe-Error Extraction attacK), with which the client can extract the model parameters, more effectively than the extraction attack on models without secure inference.

2 Related Works

A model extraction attack method attepmts to retrieve information about a remotely-deployed model, and consequently copy the model parameters, mimic the model's functionality, or infer information about its training data. For a classification model, the target inference service may return class labels, top-k probabilities, logits (or equivalently, all class probabilities), or even some intermediate features and/or gradients, among which class labels contain minimal information, leading to the most secure setup.

Most existing model extraction attacks [17, 18, 19, 20] target traditional model inference service in plaintext, while [21] and this work target encrypted model inference service. Extraction methods also differ in their objectives. We follow the taxonomy made in [19], which categorized the extraction objectives into the following types:

- Exact Extraction: extract all parameters of the target model. This objective is not possible for plaintext inference service, due the model's inherent symmetries. We will show it can be efficiently achieved for encrypted model inference service.
- Functionally Equivalent Extraction: construct a model such that its output is identical with that of the target model. The extracted model has the same structure as the target model, and the same paramters up to a symmetry transformation. This is the highest possible objective against a plaintext inference service.
- Fidelity Extraction: For some input data distribution $\mathcal{D}$ and some goal similarity function $S(\cdot,\cdot)$, Fidelity Extraction aims to construct a model $\hat{O}$, such that $\Pr_{x\sim\mathcal{D}}[S(\hat{O}(x), O(x))]$ is maximized. Typically, Fidelity Extraction only guarantee the outputs from the constructed model and the target model are similar enough on some test dataset.
- Task Accuracy Extraction: construct a model to match or exceed the accuracy of the target model.

Learning-based methods access the target model to generate a training dataset, with which a substitute model is trained. Typically, learning-based methods do not attempt to extract individual parameters, resulting in Fidelity Extraction as the objective, and is generally query-efficient. In [18], in order to find the decision boundaries between classes efficiently, an iterative training algorithm is proposed, which uses the substitute model to create samples close to decision boundaries. In

Method	Extraction method	Extraction target	Model output	Highest model depth	# model calls per parameter
Tramèr *et al.* [17]	Learning	Functional Equivalence	logits	2	$0.5 \sim 5$
Tramèr *et al.* [17]	Direct Recovery	Functional Equivalence	labels	1	$20 \sim 50$
Tramèr *et al.* [17]	Learning	Fidelity	labels	3	~ 100
Papernot *et al.* [18]	Learning	Fidelity	labels	unlimited	< 1
Jagielski *et al.* [19]	Learning	Fidelity	labels	unlimited	$\ll 1$
Jagielski *et al.* [19]	Direct Recovery	Functional Equivalence	logits	2	~ 10
Carlini *et al.* [20]	Direct Recovery	Functional Equivalence	logits	4	~ 200
MUSE [21]	MPC Malleation	Exact Extraction	logits	10	$1/n_c$
SEEK	MPC Malleation	Exact Extraction	labels	unlimited	~ 50

Table 1: Feature of the extraction methods against neural network and logistic regression models.

[19], the authors leveraged several recent optimizations in training, including unlabelled training, distillation, rotation loss, and MixMatch, to be able to train a substitute model with much fewer queries than the size of the original training set.

Direct recovery methods, which aims Functionally Equivalent Extraction, treat the target model as a function explicitly expressed by the parameters, and attempt to solve for the parameters given model query inputs and outputs. In [19], for neural networks which use ReLU activation and return logits, an extraction algorithm is devised by solving the parameters in the model, which is a piecewise-linear function. In [20], the authors utilized methodologies from cryptoanalysis, and carefully improved the differential extraction method in [19], by treating more efficiently the issues arising from larger depths and numerical errors.

In [21], the target inference service is performed with the hybrid MPC-HE scheme, and is assumed to return logits. The extraction method shifts the features so that the inference becomes a linear system, whose paramters can be solved with enough query inputs and outputs.

Features of the related works are summarized in table 1. In comparison, the proposed method SEEK considers the most restrictive setup in which only class labels are returned. Additionally, by utilizing the "safe-error attack" method [22, 23], SEEK does not suffer from the numerical error induced by very deep networks, and can apply to models with arbitrary number of layers.

It is well-known that the MPC protocol malleation attacks as in [21] and this work can be mitigated by using a protocol with malicious security. Recently, a line of work with client-malicious model are proposed [21, 24, 25]. These protocols are designed based on authenticated shares, and are closing the gap of computational and communicational efficiency with respect to the protocols with semi-honest security.

3 Secure inference setup

We consider a general deep convolutional neural network (CNN), trained for a classification task. Layers in CNN can be categorized into linear layers and non-linear layers. Linear layers include convolution layers, fully-connected (FC) layers, as well as normalization layers, average-pooling layers. Addition and concatenation layers can be viewed as linear layers as well. Consecutive linear layers can be merged together to form a single linear layer. A linear layer indexed with ℓ in general can be expressed as

$$y_\ell = w_\ell \cdot x_\ell + b_\ell, \tag{1}$$

where x_ℓ is the input feature map, y_ℓ is the output feature map, w_ℓ is the weight parameter, and b_ℓ is the bias parameter. In this formalism, for a convolution layer, the weight parameters are sparse

due to localized kernel, and the values are shared across spatial locations.

Non-linear layers include activation layers, which perform some element-wise nonlinear function, as well as max-pooling, softmax, and argmax layers. In this paper, for activation layers, we use the most common ReLU activation. For our purpose, two adjacent non-linear layers will be viewed as a single non-linear layer. Typically for a CNN, a non-linear layer indexed with ℓ is composed of an activation layer

$$z_\ell = \mathrm{ReLU}(y_\ell), \tag{2}$$

or composed of a max-pooling layer followed by an activation layer,

$$z_\ell = \mathrm{ReLU}(\mathrm{maxpool}(y_\ell)), \tag{3}$$

or, for the last layer, composed of an argmax layer,

$$z_\ell = \mathrm{argmax}(y_\ell), \tag{4}$$

where y_ℓ is the input feature map, and z_ℓ is the output feature map. We do not restrict the network structure to be linear, and structures such as skip connection and Inception are allowed.

With hybrid secure inference solutions, the client encrypts its input x_0 into $[\![x_0]\!]$, and sends $[\![x_0]\!]$ to the server. For each linear layer as in equation (1), the server computes

$$[\![y_\ell]\!] = w_\ell \cdot [\![x_\ell]\!] + b_\ell.$$

To perform a non-linear layer as in equation (2), (3), and (4), the server generates a random mask r^y_ℓ, computes $[\![y_\ell]\!] - r^y_\ell = [\![y_\ell - r^y_\ell]\!]$, and send this encrypted value to the client. The client decrypts to get $y_\ell - r^y_\ell$. Now the two parties hold secret shares of the intermediate value y_ℓ. The server and the client invoke a two-party MPC protocol corresponding to the non-linear layer. As the result, the client holds $z_\ell - r^z_\ell$, and the server holds r^z_ℓ. To transform z_ℓ back to encrypted form, the client encrypts $z_\ell - r^z_\ell$ and sends $[\![z_\ell - r^z_\ell]\!]$ to the server, who can compute $[\![z_\ell]\!]$ and proceed to the next layer. After all layers are processed, the client and the server run another MPC protocol to compute equation (4), and the client reconstructs the shares to get c.

Security of the hybrid protocol guarantees the privacy of input data, intermediate features, final result, as well as the model parameters, if the two parties follow the semi-honest model. However, we observe that the client can add arbitrary shifts to the secret shares in this protocol, and semantic security of the protocol ensures the server cannot detect the shift. Instead of using $y_\ell - r^y_\ell$ as input of the MPC calculation, the client can change it to $y_\ell - r^y_\ell + \delta y_\ell$, effectively changing the underlying value from y_ℓ to $y_\ell + \delta y_\ell$. Similarly, the client can change $z_\ell - r^z_\ell$ to $z_\ell - r^z_\ell + \delta z_\ell$, effectively changing the underlying value from z_ℓ to $z_\ell + \delta z_\ell$.

Thus the client is capable of shifting all inputs and outputs of the non-linear layers by arbitrary values, although the client is ignorant of the values of the features. We consider how the client can exploit these additional inputs, to efficiently extract the model parameters. From the viewpoint of a malicious client, the model service can be formulated as

$$\begin{aligned} c &= C_{\{w\}}(x_0, \{\delta y_\ell, \delta z_\ell : \ell \in N\}) \\ &= \mathrm{argmax}(F_{\{w\}}(x_0, \{\delta y_\ell, \delta z_\ell : \ell \in N\})), \end{aligned}$$

where $\{w\}$ denotes all model parameters, C is the functionality of the classification model, which outputs the predicted class index, N is the set of non-linear layers, and F is the output of the last linear layer.

To ease the notation, we use "named arguments" to denote the set of inputs as $(x_0, \{\delta y_\ell, \delta z_\ell : \ell \in N\}) = V(\widetilde{x_0} = x_0, \ldots, \widetilde{\delta y_\ell} = \delta y_\ell, \ldots, \widetilde{\delta z_{\ell'}} = \delta z_{\ell'}, \ldots)$. If an input is not present in the list of arguments of v, it means the input is set to zeros. For example, $V(\widetilde{\delta y_\ell} = \delta y_\ell) = V(\widetilde{x_0} = 0, \widetilde{\delta y_\ell} = \delta y_\ell, \widetilde{\delta y_{\ell'}} = 0, \widetilde{\delta z_{\ell''}} = 0)$, for all $\ell' \neq \ell$ and all ℓ''. Two sets of inputs can be added with the natural element-wise addition.

Because the output of the model is a discrete value, in order to extract model parameters, the adversary needs to find the boundary between classes, where for

$$y_{\ell_{\text{last}}} = F_{\{w\}}(v),$$

it satisfies

$$y_{\ell_{\text{last}},c_1} = y_{\ell_{\text{last}},c_2} \tag{5}$$

for two different classes c_1, c_2, and

$$y_{\ell_{\text{last}},c_1} > y_{\ell_{\text{last}},c'} \tag{6}$$

for all other c'. In the following, we call a set of input satisfying the above relations a critical point, and denote the corresponding input variables with a $*$ subscript.

Starting from a set of inputs and changing feature values on a layer, a critical point can always be found. Algorithm 1 shows a routine for finding a critical point using bisection, in which all input variables are fixed except δy_ℓ.

4 Extraction of intermediate features

In this section, we present the concrete method of SEEK. The adversary is able to shift all the inputs and outputs of the activation layers, and observe the effect on the model output. One way to extract the parameters is to find the space of critical points formed by shifting the intermediate features. However, because the landscape of model output as a function of the shifts can be very complicated, this method becomes intractable when the target layer is far away from the output layer. Instead, starting from a critical point, the proposed method will add a particular set of shifts, such that if the corresponding feature satisfies certain condition, the added shifts would cancel itself and do not affect any other features. We can test the criticality of the shifted input, and determine the value of the target feature. In this way, we keep the effect of the shifts to a minimal level, making this extraction method numerically stable and the errors in the extracted parameters independent of each other. This extraction strategy is in concept similar with the safe-error attack [22, 23] as a type of fault injection attack to security systems.

4.1 Extraction of standalone ReLU layer inputs

In this subsection, we present the method to extract an input feature value of a standalone ReLU activation as in equation (2).

Consider a critical point v^*. A target feature $y_{\ell,i}$, which is the input of a standalone ReLU activation, takes the value $y^*_{\ell,i}$ from the set of input v^*. If $y^*_{\ell,i} < 0$, shifting it by a small positive or any negative $\delta y_{\ell,i}$ will not affect the model output, because $\text{ReLU}(y^*_{\ell,i} + \delta y_{\ell,i}) = \text{ReLU}(y^*_{\ell,i}) = 0$. In this case we add a positive shift to $y_{\ell,i}$. If $\delta y_{\ell,i}$ is large enough such that $y^*_{\ell,i} + \delta y_{\ell,i} > 0$, which implies $\text{ReLU}(y^*_{\ell,i} + \delta y_{\ell,i}) \neq \text{ReLU}(y^*_{\ell,i})$, the input is no longer a critical point. We can test the criticality of the input while varying $\delta y_{\ell,i}$. At the boundary between critical points and non-critical points, $y^*_{\ell,i} = -\delta y_{\ell,i}$.

Algorithm 1: search_critical – Find a critical point by shifting y_ℓ from a given set of inputs.

Input : A fixed set of inputs v^0, variable input layer index ℓ, norm d, and error threshold ϵ

Output: δy_ℓ^*

do
  Randomly sample δy_ℓ^1 and δy_ℓ^2 with norm d;
  $c^1 \leftarrow C\left(v^0 + V(\widetilde{\delta y_\ell} = \delta y_\ell^1)\right)$, $c^2 \leftarrow C\left(v^0 + V(\widetilde{\delta y_\ell} = \delta y_\ell^2)\right)$;
while $c^1 = c^2$;
while $|\delta y_\ell^2 - \delta y_\ell^1| > \epsilon$ **do**
  $\delta y_\ell^3 \leftarrow (\delta y_\ell^1 + \delta y_\ell^2)/2$, and normalize δy_ℓ^3 with norm d;
  $c^3 \leftarrow C\left(v^0 + V(\widetilde{\delta y_\ell} = \delta y_\ell^3)\right)$;
  if $c^3 = c^1$ **then**
    $\delta y_\ell^1 \leftarrow \delta y_\ell^3$;
  else
    $\delta y_\ell^2 \leftarrow \delta y_\ell^3$, $c^2 \leftarrow c^3$;
  end
end
return $\delta y_\ell^* \leftarrow \delta y_\ell^2$;

On the other hand, if $y_{\ell,i}^* > 0$, subtracting a small positive or any negative $\delta y_{\ell,i}$ from $y_{\ell,i}$, and at the same time adding the same shift $\delta y_{\ell,i}$ to $z_{\ell,i}$, will not affect the model output, because $\mathrm{ReLU}(y_{\ell,i}^* - \delta y_{\ell,i}) + \delta y_{\ell,i} = \mathrm{ReLU}(y_{\ell,i}^*) = y_{\ell,i}^*$. If $\delta y_{\ell,i}$ is large enough such that $y_{\ell,i}^* - \delta y_{\ell,i} < 0$, the input is no longer a critical point. We can test the criticality of the input while varying $\delta y_{\ell,i}$. At the boundary between critical points and non-critical points, $y_{\ell,i}^* = \delta y_{\ell,i}$.

To test the criticality of a set of inputs v, we use the properties equation (5) and (6). Consider we start from a critical point v^* at the boundary between class c_1 and c_2, and add some shifts δv to y_ℓ^* and z_ℓ^*. If the added shifts do not change any feature values other than y_ℓ and z_ℓ, then the set of inputs $v = v^* + \delta v$ is also a critical point between class c_1 and c_2. In this case, v satisfy

$$C\left(v + V(\widetilde{\delta y_{\mathrm{last},c_1}} = \epsilon)\right) = c_1,$$

and

$$C\left(v + V(\widetilde{\delta y_{\mathrm{last},c_2}} = \epsilon)\right) = c_2,$$

where ϵ is a small positive value. If the added shifts affect other feature values, then the above equations are not satisfied with overwhelming probability.

The algorithm for extracting a input feature of a standalone ReLU is shown in algorithm 2.

4.2 Extraction of maxpool-ReLU layer inputs

In this subsection, we present the method to extract an input feature value of a maxpool layer followed by a ReLU layer, as in equation (3).

The method is similar with the one in previous subsection. Because the maxpool layer maps multiple features into one feature, when adjusting one input feature and one output feature, we need to find a way to suppress the effect of the other related input features.

Algorithm 2: extract_feature – Extraction of an intermediate feature value at a critical point

Input : Input critical point v^*, target activation layer index ℓ, and target feature index i
Output: $y^*_{\ell,i}$

if $v = v^* + V(\widetilde{\delta y_{\ell,i}} = -1)$ *is critical* **then**
  // $y^*_{\ell,i} \leq 0$
  For points of the form $v = v^* + V(\widetilde{\delta y_{\ell,i}} = \eta)$, where $\eta \in [0, \infty)$, search for the boundary $\eta = \bar{\eta}$ between critical points and non-critical points;
  return $y^*_{\ell,i} \leftarrow -\bar{\eta}$;
else
  // $y^*_{\ell,i} > 0$
  For points of the form $v = v^* + V(\widetilde{\delta y_{\ell,i}} = -\eta, \widetilde{\delta z_{\ell,i}} = \eta)$, where $\eta \in [0, \infty)$, search for the boundary $\eta = \bar{\eta}$ between critical points and non-critical points;
  return $y^*_{\ell,i} \leftarrow \bar{\eta}$;
end

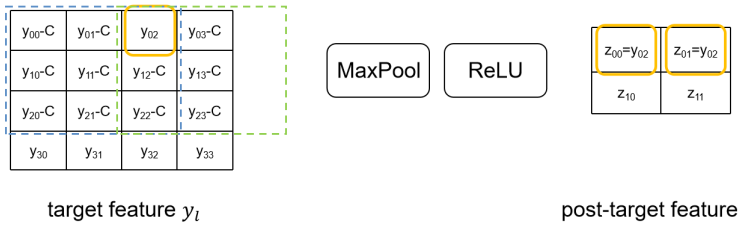

Figure 1: An example of extraction method for a maxpool-ReLU layer input feature. Features in the orange boxes are the target feature $y_{\ell,i}$ and its related features $Z_{\ell,i}$ in the post-target layer, respectively. The dashed rectangles are ranges of maxpool kernels. C is a large positive constant, added in order to suppress the effect from other features on y_ℓ to $Z_{\ell,i}$.

Assume the feature value to be extracted is $y_{\ell,i}$, and the set of output features affected by shifting $y_{\ell,i}$ is $Z_{\ell,i}$. We can add a large negative shift to all features in y_ℓ, except $y_{\ell,i}$. As a result, z_ℓ will be zero everywhere except features in $Z_{\ell,i}$, which takes the value of $y_{\ell,i}$. Now we can shift the value of $y_{\ell,i}$ and values in $Z_{\ell,i}$, observe the effect on the criticality, and consequently extract $y_{\ell,i}$. The extraction process is similar with algorithm 2, except now the feature $z_{\ell,i}$ is replaced by a set of features which should be shifted together. See figure 1 for an illustration of this method.

4.3 Extraction of linear layer parameters

The methods presented in the previous two subsections can extract all the intermediate features of a critical point. Then for each linear layer as in equation (1), with the input features and output features known, the formula is a set of linear equations for w_ℓ and b_ℓ. We can repeat this process and collect enough equations to solve all the model parameters.

To further simplify the extraction process, we note that we can add a large negative shift to the input of a ReLU activation, and ensure its output to be zero. We can also add arbitrary shifts to the zeroed outputs. Thus we have a means to accurately control the output values of ReLU activations. In equation (1), by setting x_ℓ to be identically zero and extracting y_ℓ, the value of b_ℓ

can be read off,

$$b_{\ell,j} = y_{\ell,j},$$

where j is an output feature index. By setting all but one feature value of x_ℓ zero and extracting y_ℓ, the weight parameters can be derived as,

$$w_{\ell,j,i_0} = \frac{y_{\ell,j} - b_{\ell,j}}{x_{\ell,i_0}},$$

where i_0 is index of non-zero x_ℓ value.

We observe that algorithm 2 can work on multiple target feature indices, if all the target features at these indices have the same value. In practice, running algorithm 2 on more indices improves the accuracy, because the influence of a change of their value is more significant to the model output. For convolutional layers, we can use its structure to create multiple target features with the same value. For the bias, by setting x_ℓ to be identically zero, all values on $y_{\ell,c_{\text{out}}}$ are equal to $b_{\ell,c_{\text{out}}}$, where c_{out} is an output channle index. For the weight, instead of setting one feature value on x_ℓ nonzero, for an input channel index c_{in}, we can set $x_{\ell,c_{\text{in}}}$ to be periodically nonzero, so that the target kernel value is repeated on y_ℓ.

The above extraction process is illustrated in figure 2. Algorithm 3 shows the complete algorithm to extract the parameters in a convolution layer. For clarity, we assume that the stride of the convolution to be 1. The extraction algorithm for a fully-connected layer is similar, and is omitted for brevity.

4.4 Extraction of last linear layer parameters

The extraction method described in the previous subsection applies to all the linear layers, except the last fully-connected layer before the argmax layer. Without a ReLU layer after the last fully-connected layer, the features y_{last} cannot be extracted with the extract_feature routine. Instead, the following method can be applied. Assume the numbers of input and output features of the last fully-connected layer are n_0 and n_1, respectively. To extract b_{last}, we can add shifts to the layer before the last layer, so that $x_{\text{last}} = 0$. Then we search for critical points by varying δy_{last}, which gives the relation about b_{last},

$$b_{\text{last},c_1} + \delta y^*_{\text{last},c_1} = b_{\text{last},c_2} + \delta y^*_{\text{last},c_2}.$$

$n_1 - 1$ such equations give the values of b_{last} up to an additive constant. Similarly for w_{last}, we can manipulate the layer before the last layer, so that x_{last} is zero except at feature i_0. Then we search for critical points by varying δy_{last}, which gives the relation about w_{last},

$$\begin{aligned} w_{\text{last},c_1,i_0} x_{\text{last},i_0} + b_{\text{last},c_1} + \delta y^*_{\text{last},c_1} &= w_{\text{last},c_2,i_0} x_{\text{last},i_0} + b_{\text{last},c_2} + \delta y^*_{\text{last},c_2}, \\ w_{\text{last},c_1,i_0} - w_{\text{last},c_2,i_0} &= \frac{(b_{\text{last},c_2} - b_{\text{last},c_1}) + \delta y^*_{\text{last},c_2} - \delta y^*_{\text{last},c_1}}{x_{\text{last},i_0}}. \end{aligned}$$

$(n_1 - 1)n_0$ such equations gives the values of w_{last}, up to n_0 additive constants. In fact, because only the class label is observed, this is all the degrees of freedom of w_{last} that can be determined.

5 Experiment

We test the proposed SEEK method on ResNet-18 [26], implemented in the latest PyTorch [27] release. The model contains 11.7M parameters, and is trained for ImageNet classification task.

Algorithm 3: Extraction of parameters in a convolution layer

Input : Target convolution layer index ℓ, numbers of output and input channels n_{out} and n_{in}, convolution kernal size (k_h, k_w), input feature size (f_h, f_w)

Output: Target convolution layer parameters b_ℓ and w_ℓ

Get the layer index ℓ_0 whose output is the input of layer ℓ, i.e., $z_{\ell_0} = x_\ell$;

Get the index of last non-linear layer ℓ_{last};

Add a large negative shift $-d$ to all features in y_{ℓ_0};

$\delta z^*_{\ell_{\text{last}}} \leftarrow \mathsf{search\_critical}(V(\widetilde{x_0} = x_0, \widetilde{\delta y_{\ell_0}} = -d), \ell_{\text{last}})$;

$v^* \leftarrow V(\widetilde{x_0} = x_0, \widetilde{\delta y_{\ell_0}} = -d, \widetilde{\delta z_{\ell_{\text{last}}}} = \delta z^*_{\ell_{\text{last}}})$;

for $c_{out} \leftarrow 0$ **to** $n_{out} - 1$ **do**

  $\beta \leftarrow \{(c_{\text{out}}, i, j) : 0 \le i < f_h, 0 \le j < f_w\}$;

  $b_{\ell, c_{\text{out}}} \leftarrow \mathsf{extract\_feature}(v^*, \ell, \beta)$;

end

$k'_h \leftarrow (k_h - 1)/2, k'_w \leftarrow (k_w - 1)/2$;

$\Delta \leftarrow (n_{\text{in}} \cdot k_h \cdot k_w / 4)^{1/2}$;

for $c_{in} \leftarrow 0$ **to** $n_{in} - 1$ **do**

  Create a feature map α of size (f_h, f_w) whose values are $\alpha_{i,j} = \Delta \cdot \hat{\delta}((i - k'_h)\%k_h) \cdot \hat{\delta}((j - k'_w)\%k_w))$, where $\hat{\delta}(\cdot)$ is the discrete delta function;

  $\delta z^*_{\ell_{\text{last}}} \leftarrow \mathsf{search\_critical}(V(\widetilde{x_0} = x_0, \widetilde{\delta y_{\ell_0}} = -d, \widetilde{\delta x_{\ell, c_{\text{in}}}} = \alpha, \ell_{\text{last}})$;

  $v^* \leftarrow V(\widetilde{x_0} = x_0, \widetilde{\delta y_{\ell_0}} = -d, \widetilde{\delta x_{\ell, c_{\text{in}}}} = \alpha, \widetilde{\delta z_{\ell_{\text{last}}}} = \delta z^*_{\ell_{\text{last}}})$;

  for $c_{out} \leftarrow 0$ **to** $n_{out} - 1$ **do**

    for $i \leftarrow 0$ **to** $k_h - 1$ **do**

      for $j \leftarrow 0$ **to** $k_w - 1$ **do**

        $\beta \leftarrow \{(c_{\text{out}}, i', j') : 0 \le i' < f_h, (i' - k_h + 1 + i)\%k_h = 0, 0 \le j' < f_w, (j' - k_w + 1 + j)\%k_w = 0\}$;

        $y_{\ell, c_{\text{out}}, i, j} \leftarrow \mathsf{extract\_feature}(v^*, \ell, \beta)$;

        $w_{\ell, c_{\text{out}}, c_{\text{in}}, i, j} \leftarrow (y_{\ell, c_{\text{out}}, i, j} - b_{\ell, c_{\text{out}}})/\Delta$;

      end

    end

  end

end

return b_ℓ *and* w_ℓ;

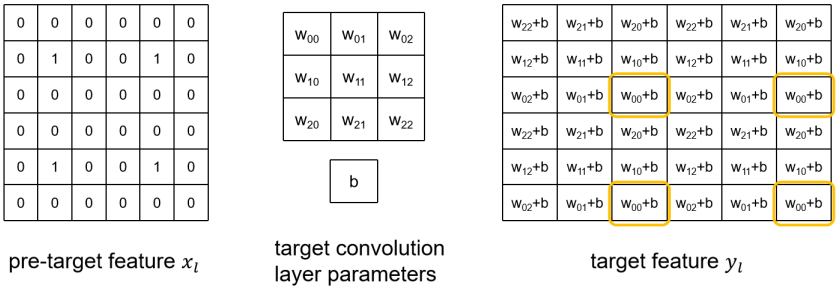

Figure 2: An example of the convolutin layer extraction method, as shown in algorithm 3. For simplicity, only one channel for each layer is shown. By setting the pre-target feature x_ℓ to be nonzero with a period of kernel size, the target feature layer y_ℓ is also periodic, and the values in the orange boxes can be extracted together for better accuracy, which reveal the values of target convolution layer parameters.

In ResNet-18, some of the linear layers have a single preceding layer, while the layers immediately after the addition layers have two preceding layers. In addition, some skip connections are identity connections, and some are down-sampling connections, which have their own convolution weights. In all of these cases, we can use the methods in the previous section to extract the linear layers' parameters. Figure 3 shows several extraction paths for different cases in ResNet.

We implemented the extraction algorithm, and experimentally tested its performance. Figure 4 shows the average number of model calls required for extracting each parameter, as well as the average relative error, for different layers in ResNet-18. For each parameter, the average number of model calls is 45.8. The average relative error of bias is 6.68×10^{-6}, and the average relative error of weight is 4.35×10^{-5}.

As figure 4 shows, the average error of weight tends to be larger as the layer is closer to the model output, except for the last FC layer. The reason for this phenomenon is, if the shift of the target feature is larger than its value (see algorithm 2), the output of ReLU function $z = z^* + \delta z$ will be different from its original value z^* and in turn changes the final logits. However, for a layer closer to the model output, the relationship between δz and the final logits y_{last,c_1} and y_{last,c_2} becomes simpler. In some rare cases, δz affects y_{last,c_1} and y_{last,c_2} approximately in the same way in a small neighborhood. In this small neighborhood of δz value, algorithm 2 cannot distinct the shift by criticality test, and resulting in a larger error. This issue can be mitigated by repeating the extraction multiple times with different initial critical points.

6 Conclusion and discussion

In this work, we proposed SEEK, a model extraction attack method against HE-MPC hybrid inference service with semi-honest security, with the most stringent assumption that the model outputs class labels only. Our method makes use of the piecewise-linear property of the ReLU activation, and the principle of safe-error attack, thus achieving an extraction process that can accurately extract each layer's parameters. As the method tests whether a shift to the internal feature affects the criticality of the whole input, it is not affected by the depth of the model, which can incur numerical issues for other extraction methods. Furthermore, because the extraction of parameters in a layer is not dependent on the extraction result of any other layer, a distributed extraction attack is straightforward.

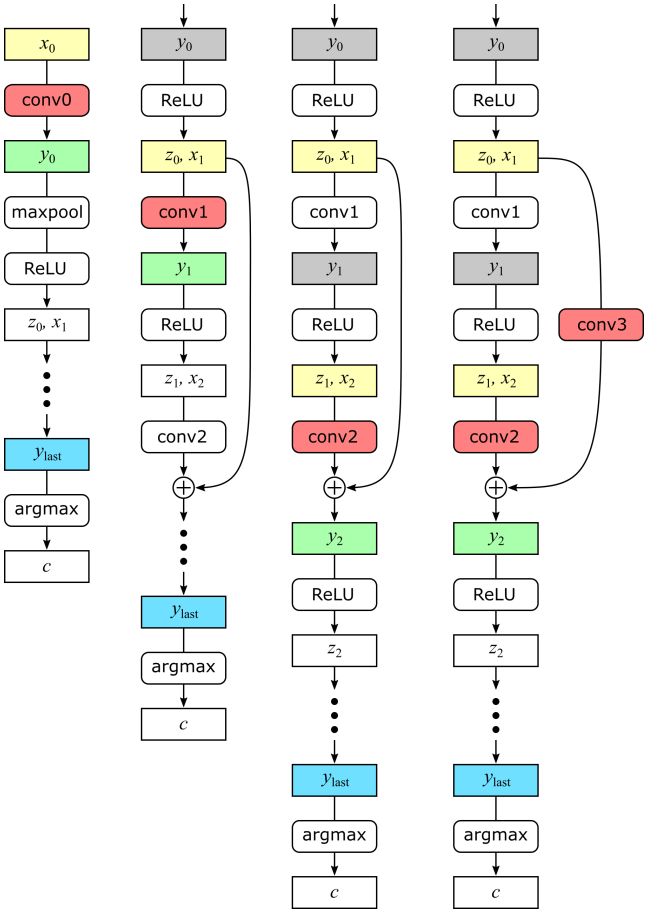

Figure 3: Examples of extraction paths for ResNet. A large negative value is added to the grey layers, so that the feature values in the pre-target layers (yellow) can be adjusted to some convenient values. A bisection search is performed on the last feature layers (blue) to find a critical point. The feature values of the post-target layers (green) are extracted, based on properties of the non-linear succeeding non-linear layers. Then the parameters of the target layers (red) are extracted.

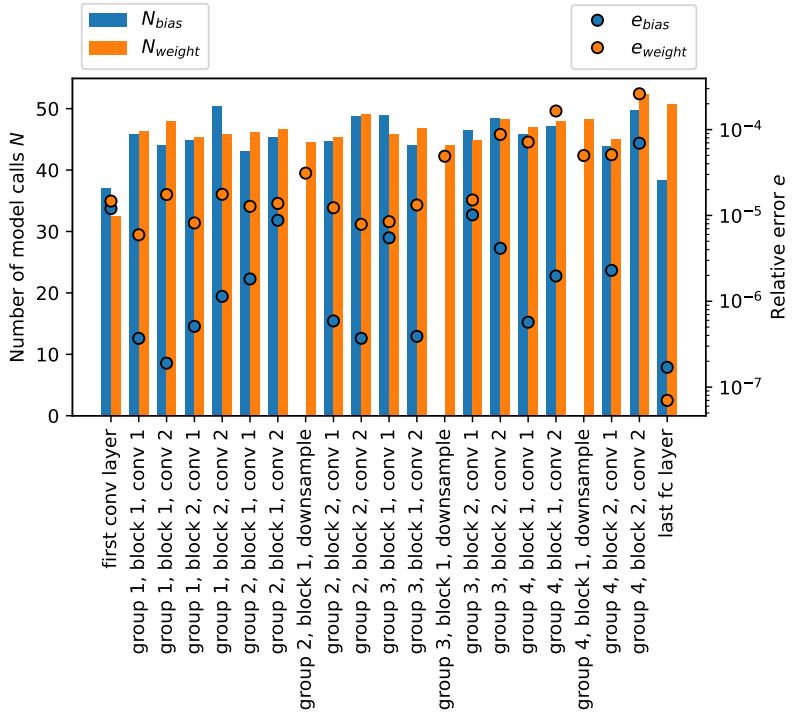

Figure 4: Result of the proposed extraction method on ResNet-18. $N_{\text{bias}}(N_{\text{weight}})$ is the average number of model calls for extracting a bias(weight) parameter. $e_{\text{bias}}(e_{\text{weight}})$ is the average relative error of the extracted bias(weight) parameter.

SEEK can be generalized to other secure inference protocols with semi-honest security. In particular, if the ReLU activation function is replaced by other piecewise-linear functions, such as ReLU6 or leaky ReLU, our method can be applied in essentially the same manner. If the activation function is linear only in part of the input range, such as the swish activation, we can also manipulate the input so that it falls in the region of linear activation. For secure inference of decision tree models, the general method of safe-error attack is applicable, because the discrete nature of decision tree inference makes it possible to change individual intemediate feature and observe the effect on the final output. We leave the security analysis of the case of decision tree models for future work.

As demonstrated by the proposed extraction method, the capability of changing all the intermediate features with arbitrary shifts is quite powerful, and it is non-trivial to prevent such attack. Shuffling the features in a layer before the MPC protocol only increases the difficulty of this attack by a constant factor. The model inference protocols with client-malicious security [21, 24, 25], albeit with significant communicational and computational cost, provide systematic countermeasure against our attack. Secure inference based on fully homomorphic encryption with bootstrapping [28, 29], or garbled circuits [6, 7, 8], lead to another direction of mitigation, in which the inference is processed in constant communication rounds, so an adversary do not have the oppotunity to malleate intermediate model features.

References

[1] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1322–1333.

[2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 3–18.

[3] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 2005, pp. 641–647.

[4] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[5] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *International Conference on Machine Learning*. PMLR, 2019, pp. 812–821.

[6] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th annual design automation conference*, 2018, pp. 1–6.

[7] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, "Garbled neural networks are practical," *Cryptology ePrint Archive*, 2019.

[8] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "{XONN}:{XNOR-based} oblivious deep neural network inference," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1501–1518.

[9] P. Mohassel and P. Rindal, "Aby3: A mixed protocol framework for machine learning," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 35–52.

[10] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training." *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[11] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 336–353.

[12] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1651–1669.

[13] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[14] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2505–2522.

[15] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow2: Practical 2-party secure inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 325–342.

[16] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference." *IACR Cryptol. ePrint Arch.*, vol. 2022, p. 207, 2022.

[17] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction {APIs}," in *25th USENIX security symposium (USENIX Security 16)*, 2016, pp. 601–618.

[18] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[19] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1345–1362.

[20] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in *Annual International Cryptology Conference*. Springer, 2020, pp. 189–218.

[21] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2201–2218.

[22] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Transactions on computers*, vol. 49, no. 9, pp. 967–970, 2000.

[23] M. Joye and S.-M. Yen, "The montgomery powering ladder," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2002, pp. 291–302.

[24] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "Simc: Ml inference secure against malicious clients at semi-honest cost," *Cryptology ePrint Archive*, 2021.

[25] G. Xu, X. Han, T. Zhang, H. Li, and R. H. Deng, "Simc 2.0: Improved secure ml inference against malicious clients," *arXiv preprint arXiv:2207.04637*, 2022.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[28] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," in *International Symposium on Cyber Security Cryptography and Machine Learning*. Springer, 2021, pp. 1–19.

[29] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.