Encrypted-Input Obfuscation of Image Classifiers

Giovanni Di Crescenzo, Lisa Bahler, Brian A. Coan, Kurt Rohloff, David B. Cousins, Yuriy Polyakov

Encrypted-Input Obfuscation of Image Classifiers

Giovanni Di Crescenzo[1], Lisa Bahler[1], Brian A. Coan[1], Kurt Rohloff[2], David B. Cousins[3], and Yuriy Polyakov[3]

[1] Peraton Labs, Basking Ridge, NJ, USA
{gdicrescenzo,lbahler,bcoan}@peratonlabs.com
[2] New Jersey Institute of Technology, Newark, NJ, USA
rohloff@njit.edu
[3] Duality Technology Inc., Newark, NJ, USA
{dcousins,ypolyakhov}@dualitytech.com

Abstract. We consider the problem of protecting image classifiers simultaneously from inspection attacks (i.e., attacks that have read access to all details in the program's code) and black-box attacks (i.e., attacks where have input/output access to the program's code). Our starting point is cryptographic program obfuscation, which guarantees some provable security against inspection attacks, in the sense that any such attack is not significantly more successful than a related black-box attack. We actually consider the recent model of encrypted-input cryptographic program obfuscation, which uses a key shared between the obfuscation deployer and the input encryptor to generate the obfuscated program. In this model we design an image classifier program and an encrypted-input obfuscator for it, showing that the classifier program is secure against both inspection and black-box attacks, under the existence of symmetric encryption schemes. We evaluate the accuracy of our classifier and show that it is significantly better than the random classifier and not much worse than more powerful classifiers (e.g., k-nearest neighbor) for which however no efficient obfuscator is known.

Keywords: Inspection Attacks · Black-box Attacks · Program Obfuscation · Image Classifiers

1 Introduction

According to web sources, the Internet of Things (IoT) market is expected to grow by $ 421.28 billions during 2021-2025, progressing at a compound annual growth rate of 33%. In many typical IoT applications, servers perform analytics over data received by multiple distributed sensors (see, e.g., [21]). Just like most web or cloud computing services, IoT analytics servers can be subject to a number of attacks. In this paper, we focus on attacks to the server programs, here categorized as inspection attacks (informally defined as attacks that try to access internal data or computation used by the server program), and black-box attacks (informally defined as attacks only use input-output access to the attacked server program, and no access to any internal data or computation).

Our starting point to propose solutions mitigating these attacks is the recent area of cryptographic program obfuscation, which promises a set of solutions with some provable security guarantee in the presence of inspection attacks, but does not address the problem of protecting programs against black-box attacks. Program obfuscation is the problem of modifying a computer program so to hide any sensitive details without changing its input/output behavior. While this problem has been known for several years in computer science, only in the last 20 years or so, researchers have considered the problem of *provable* program obfuscation, where sensitive code details are proved to remain hidden under a widely accepted intractability assumption, such as those often used in cryptography. The most studied security guarantee offered by provable program obfuscation, also called "virtual-black-box" obfuscation [4], says that for any efficient inspection attack to the program (i.e., an attack that has access to all details in the program's code) there exists an efficient black-box attack to the program (i.e., an attack that only has input-output access to the program) that is about as equally successful. Early results in the area implied the likely impossibility of constructing a single program capable of obfuscating any input arbitrary polynomial-time program into a virtual black box [4]. Most recent results show the possibility of constructing, under close to standard intractability assumptions, practically efficient obfuscators for very restricted families of functions, such as point functions and a few extensions of them (see, e.g., [6, 17, 22, 5, 11, 13]), as well as theoretically feasible obfuscators for large families of functions (e.g., compute-and-compare functions [23]).

On one hand, such provable program obfuscation solutions make inspection attacks to the program's sensitive information essentially useless, in that any inspection attack would not be significantly better than a related black-box attack. On the other hand, the security guarantee does not say anything new about black-box attacks. Recent results (see, e.g., [19, 20, 12]) show successful black-box attacks to popular programs (e.g., machine learning programs), even undermining the success of the related business model (e.g., MLaaS). Motivated by these results, recent work [10] has considered the problem of augmenting the cryptographic program obfuscation model so to achieve, in at least some class of application scenarios, program confidentiality in the presence of *both* inspection and black-box attacks. A resulting model, called *encrypted-input program obfuscation*, has been proposed as a mixed encryption/obfuscation model with the following security guarantee: for any efficient inspection attack to the program (i.e., an attack that has access to all details in the program's code) there exists an efficient algorithm (note: *not* one that is given black-box attack to the program) that is about as equally successful. Thus, a provable encrypted-input obfuscation solution makes both inspection attacks and black-box attacks to the program's sensitive data essentially useless. Moreover, in this model the parties generating the inputs (e.g., the IoT devices) are assumed to encrypt them by using a key shared with the entity obfuscating the program, which can then work by computing over the obfuscated program and the encrypted (and authenticated) input.

In this paper we continue this effort and specifically focus on posing the problem for machine learning classifiers (instead of arbitrary programs) and on finding a concrete classifier that can be secured against both black-box and inspection attacks in this model. We start by observing that any encrypted-input obfuscator for an arbitrary program can be used to design an encrypted-input obfuscator for a machine learning classifier, but then note that the resulting scheme would be very inefficient (e.g., depending linearly in the dataset) and may likely require conversions to other representations (e.g., circuits). Thus, we consider the problem for a specific task: *image matching classification*, where given a secret image, and an input image, the classifier returns 1 if they belong to the same class, or 0 otherwise. Since no efficient obfuscators are known for complex machine learning classifiers, we opt for designing our own image matching classifiers, using tools like principal component transformations and textbook statistics, for which we know how to produce an encrypted-input obfuscator for the related evaluation program. We study the true positive rate and true negative rate for this classifier and obtain that they are significantly better than the random classifier and not much worse than much more powerful classifiers (e.g., k-nearest neighbor) for which however no efficient obfuscator is known.

Table 1. For each computing model (programs or classifiers), 2 security models are considered: cryptographic obfuscation (briefly, obfuscation) and encrypted-input cryptographic obfuscation (briefly, ei-obfuscation). For each computing model and associated security model, the table lists if provable security is guaranteed in the presence of inspection security, and of black-box attack security, which paper first defines the security model, and which appendix in this paper contains the main definition.

Computing Model	Security Model	Inspection attack security	Black-box attack security	Defined in
Programs	Obfuscation	yes	no	[3], Appendix B.1
Programs	ei-Obfuscation	yes	yes	[10], Appendix B.3
Classifiers	Obfuscation	yes	no	[9], Appendix B.2
Classifiers	ei-Obfuscation	yes	yes	Appendix C

2 Definitions and Models

We present definitions of the computing models of interest (i.e., secret-based programs and matching classifiers) in Section 2.1; and an informal discussion of attack classes and resources, previous related obfuscation models, and the model for encrypted-input obfuscation of matching classifiers in Section 2.2. Formal definitions of the various obfuscation models from the literature are recalled in Appendices A, B.1, B.2 and B.3, and finally the (new) formal definition for encrypted-input obfuscation of matching classifiers is presented in Appendix C.

2.1 Computing Models: Programs and Classifiers

We consider two computation models: (secret-based) programs and (matching) classifiers. Informally, secret-based programs are programs with both a public and a secret input, and matching classifiers are a pair of programs: a training program and a matching program, with a specific syntax, including inputs of specific data and label types. We now proceed more formally.

(Secret-based) Programs. We consider *families of (secret-based) functions* as families $F = \{f_{pv,sv}\}$ of maps $f_{pv,sv} : \{0,1\}^n \to \{0,1\}$ parameterized by some public values $pv \in \{0,1\}^{m_p}$ and secret values $sv \in \{0,1\}^{m_s}$, for some *length parameter* n, and *parameter value lengths* m_p, m_s polynomial in n. We will think of public values pv as being available to all parties, including the adversary, and of secret values sv as encoding the information that has to remain secret from the adversary. In later sections of the paper, we will specifically consider obfuscation of the following classes of (secret-based) programs:

1. the *family of range-membership programs*, where the program computes if an input value belongs to a range, where we keep the two range limits secret but the length of their binary representation public. Formally, we define the family of programs $RM_{pv,sv}$, where $pv = (1^n), sv = (a, b)$, with $[a, b] \subseteq \{0,1\}^n$, and that on input $x \in \{0,1\}^n$, return 1 if $x \in [a, b]$ and 0 otherwise.
2. the *family of conjunctions of range-membership programs*, where the program computes if each input value in a sequence belongs to a (potentially different) range, where we want to keep all the range limits secret but can keep the length of their binary representation public. Formally, we define the family of programs $CRM_{pv,sv}$, where $pv = (1^n, 1^t), sv = (a_1, b_1, \ldots, a_t, b_t)$, with $[a_i, b_i] \subseteq \{0,1\}^n$, and that on input $x_1, \ldots, x_t \in \{0,1\}^n$, return 1 if $x_i \in [a_i, b_i]$ for all $i = 1, \ldots, t$, and 0 otherwise.

Matching Classifiers. By D we denote a *probability distribution*, and by dS we denote a *data space*; that is, the set $dS = \{d_1, \ldots, d_N\}$ of all possible data samples, that can be drawn according to distribution D. For instance, a sample d_i could be an image of an object (e.g., a car) and D could returns images of cars of possibly different brands.

By cS we denote a *class space*; that is, the set $cS = \{c_1, \ldots, c_q\}$ of all possible data classes. For instance, c_i could be the i-th car brand name within a known and pre-specified list.

The *class function* is a function $cF : dS \to cS$ mapping a data sample in dS to its class in cS. In the given example, function cF would map the data sample $d \in dS$ containing the image of a car to a value $c_i \in cS$ denoting this car's brand name, as from the known list.

We define a *matching classifier* (briefly, *classifier*) *for class function* cf as a pair of algorithms MC = (CTrain,CMatch) such that:

– on input a dataset $ds = (d_1, \ldots, d_n) \in dS^n$ of n data samples, labels $cl = (c_1, \ldots, c_n)$ such that $c_i = cF(d_i)$ for $i = 1, \ldots, n$, and a data sample $d_0 \in dS$, algorithm CTrain returns matching auxiliary input $maux$;

– on input data sample d and matching auxiliary input $maux$, algorithm CMatch returns a bit b. Here, the value $b = 1$ denotes that this classifier predicts that class $c = cF(d)$ of data sample d is equal to the class $c_0 = cF(d_0)$ of data sample d_0; and, naturally, the value $b = 0$ denotes that the classifier predicts that $c \neq c_0$.

Towards defining classifier MC's output accuracy metrics of interest, relatively to a dataset ds with samples independently drawn from distribution D and with labels cl, a data sample d_0, and a random execution of $\text{CTrain}(ds, cl, d_0)$ returning matching auxiliary input $maux$, we say that a data sample d can be a

– *true positive*, if $cF(d) = c_0$ and $\text{CMatch}(d, maux) = 1$ (predicting $c = c_0$);
– *true negative*, if $cF(d) \neq c_0$ and $\text{CMatch}(d, maux) = 0$ (predicting $c \neq c_0$);
– *false positive*, if $cF(d) \neq c_0$ and $\text{CMatch}(d, maux) = 1$ (predicting $c = c_0$);
– *false negative*, if $cF(d) = c_0$ and $\text{CMatch}(d, maux) = 0$ (predicting $c \neq c_0$).

Let tp (resp., tn, fp, fn) denote an estimate of the number of true positives (resp., true negatives, false positives, false negatives) for matching classifier MC. Based on these definitions, relatively to a dataset ds, a class c, and a random execution of $\text{CTrain}(ds, cl, d_0)$ returning matching auxiliary input $maux$, we can then define the following *classification metrics* for MC:

– *true negative rate* (aka specificity): $tn/(tn + fp)$;
– *true positive rate* (aka recall): $tp/(tp + fn)$.

Similarly as for the program computing model, even in the classifier computing model we will think of some inputs as being available to all parties, called *public parameter values* $pv(\text{MC})$, and of values that have to remain secret from the adversary, called *secret parameter values* $sv(\text{MC})$. Specifically, $pv(\text{MC})$ contains a description of CTrain and CMatch, a syntactic description of dataset ds, including the number n of data samples, and a syntactic description of data space dS, including the total number N of possible data samples. Moreover, $pv(\text{MC})$ contains dataset ds, labels cl, data sample d_0 and its class $c_0 = cF(d_0)$.

In later sections of the paper, we will specifically consider obfuscation of the *family of image matching classifiers*, denoted as $\text{iMC} = (iCTrain, iCMatch)$. This is defined exactly as a family of matching classifiers, with the (only semantic) difference that data samples are images.

2.2 Modeling Obfuscation of Matching Classifiers

Our goal is to produce a formal model for the encrypted-input obfuscation of image matching classifiers. Since much literature focuses on obfuscation of programs, we first discuss related formal models on program obfuscation from the literature, and then where we extend these models.

Threat model: attack classes and attacker resources. We consider attacks trying to infer whether a binary-valued property (possibly involving secret data or program description) is satisfied or not.

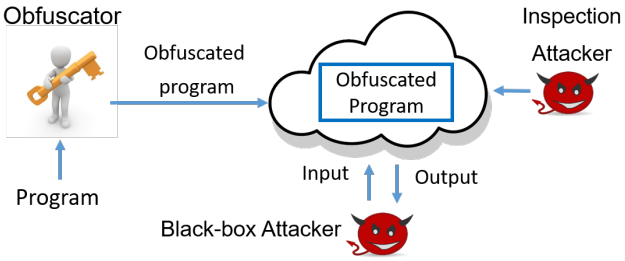

Fig. 1. Usage paradigm for a cryptographic program obfuscator.

An *inspection attack to a program* is defined as a polynomial-time attack, that given access to some or all of the program's code, tries to find out whether a property of the program is satisfied or not. An *inspection attack to a matching classifier* is defined as an inspection attack to the classifier's matching program, where the attacker is additionally given access to some or all of the output $maux$ of the classifier's training program.

A *black-box attack to a program* is defined as a polynomial-time attack, that given input-output access to the program (but not given access to any part of the program's code), tries to find out whether a property of the program is satisfied or not. A *black-box attack to a matching classifier* is defined as a black-box attack to the classifier's matching program, whose input $maux$ is set as the output of the classifier's training program.

Previous related obfuscation models. Cryptographic program obfuscation, as originally studied in [14, 4], is about security of an arbitrary program in the presence of inspection attacks. Here, a virtual black-box obfuscation property is formalized, briefly speaking, as follows: for any efficient inspection attack, there is an efficient black-box attack that is about equally successful. Note that this property does not target black-box attacks. A pictorial description of the usage paradigm for a cryptographic program obfuscator can be found in Figure 1.

Intrusion-resilient matching classifiers, as recently defined in [9], extend the above cryptographic program obfuscation model to a type of matching classifiers. The extension is necessary due to a different program syntax (specifically, matching classifiers are a pair of programs, instead of a single program), and a different set of public data and information to be kept secret (specifically, in matching classifiers the description of algorithms CTrain and CMatch may be public while the dataset ds, the class labels cl, and a data class c are desired to remain secret). The usage paradigm of a cryptographic obfuscator for a matching classifier is obtained by using, in Figure 1, a program equal to $\mathrm{CMatch}(d, maux)$, where $maux = \mathrm{CTrain}(ds, cl, d_0)$.

Encrypted-input program obfuscation, as recently defined in [10], targets a combined key-based encryption/obfuscation of an arbitrary program, satisfying the following property: for any efficient inspection attack, there is an efficient attack with neither inspection nor black-box access to the program that is equally

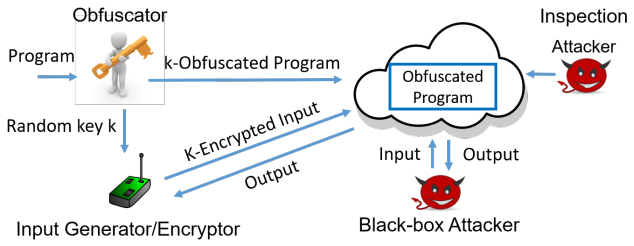

Fig. 2. Usage paradigm for an encrypted-input cryptographic program obfuscator.

successful. The model in [10] can be seen as obtained by applying two modifications to the original model in [14, 4]: (1) the obfuscation of the program is performed using a random key that is unknown to the attacker; and (2) the inputs to the program are always generated in encrypted form, using this key. Here, (1) suggests that security against inspection attacks may follow from some form of encryption of the program, and (2) suggests that security against black-box attacks may follow since an attacker, not knowing the key, may not be able to generate valid inputs for a black-box attack. The usage paradigm for an encrypted-input cryptographic program obfuscator is depicted in Figure 2.

In this paper, we generalize the encrypted-input program obfuscation model of [10] to matching classifiers. The generalization is necessary due to a different program syntax (specifically, matching classifiers are a pair of programs, instead of a single program), and a different set of public data and information to be kept secret (specifically, in matching classifiers the description of algorithms CTrain and CMatch may be public while the dataset ds, the class labels cl, the secret data sample d_0 and its class c_0 are desired to remain secret). Our definition of matching classifiers also slightly generalizes [9], as follows: in our paper the algorithm CMatch of a matching classifier takes as a secret input a data sample d_0, and possibly but not necessarily its data class c_0, while in the previous definition the secret input was just a secret class c. The usage paradigm of an encrypted-input obfuscator for a matching classifier is obtained by using, in Figure 2, a program equal to $\mathrm{CMatch}(d, maux)$, where $maux = \mathrm{CTrain}(ds, cl, d_0)$. It is interesting to compare the original virtual-black-box obfuscation requirement, as rewritten in Appendix B.1 for programs and in Appendix B.2 for classifiers, with the simulated-view obfuscation requirements in Definition 2 for programs and in Definition 3 for classifiers. In the virtual-black-box obfuscation requirements, it is required that the adversary's view can be simulated by an efficient algorithm that is given access to a black-box computing the original program or the original matching algorithm of the classifier, while in the latter the efficient algorithm is *not* given access to any such black box. We derive:

- For an obfuscator satisfying the virtual-black-box obfuscation property, an inspection attack is not significantly more successful than a related black-

box attack, but this does not rule out the existence of a black-box attack that learns the (program's or) the classifier's secret parameter values sv.
- For an encrypted-input obfuscator satisfying the simulated-view obfuscation property, if the obfuscation key remains secret, the obfuscated version of the (program or) matching classifier is of no help to inspection or black-box attacks to learn the (program's or) classifier's secret parameters vaues sv.

3 A General Result on Encrypted-Input Obfuscation

We consider a question naturally arising from our definitions of encrypted-input obfuscation of programs and matching classifiers: is it possible to construct an encrypted-input obfuscator for an arbitrary family of matching classifiers (as formally defined in Definition 3) starting from an encrypted-input obfuscator for any family of programs (as formally defined in Definition 2, recalled in Appendix B.3). We give a positive answer to this question and show the following

Theorem 1. For any family of matching classifiers MC, there exists a family of secret-based programs P such that the following holds. If there is an encrypted-input obfuscator for P then there exists an encrypted-input obfuscator for MC.

To prove Theorem 1, we show a family of secret-based programs for any family of matching classifiers. Specifically, let MC = (CTrain,CMatch) be a family of matching classifiers, as formally defined in Section 2.1. We define the family of secret-based programs $P_{pv,sv}$, where

- $pv = (1^n, 1^N, desc(\text{CTrain}), desc(\text{CMatch}), pv(\text{MC}))$,
- $sv = (ds, cl, d_0, sv(\text{MC}))$,

and such that, on input x, it returns $b = \text{CMatch}(x,\text{CTrain}(ds, cl, d_0))$. The computation correctness, low runtime overhead and simulated-view obfuscation properties of the encrypted-input obfuscator for MC directly follow from the analogue properties of the encrypted-input obfuscator for $P_{pv,sv}$.

Remarks. In [10] it was observed that Yao's garbling circuit technique [24] can be directly used to construct an encrypted-input obfuscator for any polynomial-time program, hiding all circuit gates of the input circuit equivalent to the program. When combined with Theorem 1, this implies a similar general result for any matching classifier. We caution the reader that a direct use of Theorem 1 would result in an obfuscated matching classifier of size polynomial in the dataset size (which is undesirable as in many practical applications datasets are very large). However, we believe that this result is still encouraging towards finding, in some model, general methods to efficiently and provably secure classifiers against both inspection and black-box attacks.

4 Image Matching

In this section we present our result on image matching classifiers. First, in Section 4.1, we recall background definitions of interest, including principal component transformations. Next, in Section 4.2, we formally describe our new image

matching classifier, and finally in Section 4.3 we report our experimental analysis of its accuracy properties over 3 different datasets.

4.1 Background Definitions and Tools

An *attribute* A is a function that maps a data sample $d_i \in dS$ to a numerical *attribute value* v in some *value space* vS. Let $A_1, \ldots, A_m$ denote m distinct attributes. To a data sample $d_i \in dS$, we can then associate a value tuple $v_i = (v_{i,1}, \ldots, v_{i,m})$, where $v_{i,j}$ represents $A_j(d_i)$; that is, the value returned by attribute A_j on input data sample d_i. For instance, if d_i is an image, the value $v_{i,j}$ could represent the numeric value associated with the j-th pixel (or block of pixels) in d_i. We can then define a *data $n \times m$-matrix* V as a matrix where each of the n rows is associated to a data sample d_i, each of the m columns is associated to an attribute A_j, and each entry $v_{i,j}$ contains value $A_j(d_i)$.

For a sequence of values $v_j = (v_{1,j}, \ldots, v_{n,j})$ from value set vS, we use the textbook definitions of *mean*, denoted as $\mu(v_j)$, and *standard deviation*, denoted as $\sigma(v_j)$, for all $j = 1, \ldots, m$.

An important tool used in our classifier construction is a *Principal Components transformation* (briefly, pcT). Informally speaking, pcT is defined as an orthogonal linear transformation converting a data matrix to a new coordinate system where the greatest variance by some scalar data projection lies on the 1st coordinate (also called the 1st principal component), the 2nd greatest variance on the 2nd coordinate, and so on [15]. Thus, truncating this transformation to its first coordinates is sufficient to capture a large part of the data variability of interest in many practical uses. More formally, given a data $(n \times m)$-matrix V, we define the *truncated Principal Components transformation* (briefly, tpcT) of V as the function tpcT that on input V, returns $Y = VW$, where W is an $(m \times \ell)$-matrix, dependent on V, with elements in vS, and the product VW is a matrix product, returning $(n \times \ell)$-matrix Y, where we usually think of ℓ as much shorter than m. Thus, tpcT also serves as a dimensionality reduction method. While one classical way to compute W from V consists of setting W as the matrix whose columns are the first ℓ eigenvectors of matrix $V^{\mathrm{T}}V$, many more efficient variants and generalizations have been studied (see, e.g., [15]).

4.2 Our New Image Matching Classifier

In this subsection we formally describe a new image matching classifier. First, we provide formal definitions for some useful families of classifiers: range membership, and conjunction of range memberships. Then we use these classifiers and the background tools from Section 4.1 to present our image classifier IC.

Our image matching classifier IC. Our image matching classifier, denoted as IC=(imCTrain,imCMatch), reduces the problem of image matching (i.e., matching a new test image against a secret image) to the problem of evaluating a conjunction of range memberships (i.e., testing if each of the values in an input test sequence belongs to a prespecified, secret, value range). Informally speaking,

this is performed using the following steps: (1) a truncated principal components transformation tpcT maps the dataset images to short vectors capturing a large part of the images' variability; (2) the tpcT output is processed by using basic statistics like mean and standard deviation to capture summary ranges for attribute values relative to each attribute, for each attribute and each image class; and finally (3) such ranges and the input secret image are used to define a conjunction of ranges of attribute values, one range for each attribute, against which any new test image can be later matched.

A formal description follows. (For simplicity, in this description we assume that the class c_0 of the secret data sample d_0 is known to algorithm imCTrain. Later, we show the extension to the more general cases when c_0 is not known.)

Input to algorithm imCTrain:

- an n-image dataset $ds = (d_1, \ldots, d_n)$
- class labels $cl = (c_1, \ldots, c_n)$ such that $c_i = cF(d_i)$, for $i = 1, \ldots, n$
- a secret data sample d_0 and its class $c_0 = cF(d_0)$
- a parameter 1^m denoting the number of image attributes

Instructions for algorithm imCTrain:

1. let V be the data $n \times m$-matrix associated with the n-image dataset ds
2. set $W = \text{tpcT}(V)$
3. set $Y = VW$ (i.e., Y is the product of matrices V and W)
4. let A_j denote the attribute that maps the data sample d_i in V's i-th row to the j-th data block in Y's i-th row, for $i = 1, \ldots, n$, and $j = 1, \ldots, m$
5. for all $i = 1, \ldots, n$ such that $cF(d_i) = c_0$,
   set $y_i = (y_{i,1}, \ldots, y_{i,m})$,
   where $y_{i,j} = A_j(d_i)$, for all $j = 1, \ldots, m$
6. let α be a configurable constant (e.g., $\alpha = 1.5$)
7. for all $j = 1, \ldots, m$
   compute $ct_j = \mu(\{y_{i,j} | cF(d_i) = c_0\})$
   compute $std_j = \sigma(\{y_{i,j} | cF(d_i) = c_0\})$
   set $a_j = ct_j - \alpha \cdot std_j$, and $b_j = ct_j + \alpha \cdot std_j$
8. return: $maux = (W, (a_1, b_1), \ldots, (a_m, b_m))$

Input to Algorithm imCMatch:

- a tuple $maux$ returned by imCTrain
- an image data sample d

Instructions for Algorithm imCMatch:

1. write $maux$ as $(W, (a_1, b_1), \ldots, (a_m, b_m))$
2. compute $e = dW$ (i.e., e is the product of vector d and matrix W)
3. let A'_j denote the attribute mapping data sample d to the j-th data block in e, for $j = 1, \ldots, m$
4. set $e = (e_1, \ldots, e_m)$, where $e_j = A'_j(e)$, for $j = 1, \ldots, m$

5. if $e_1 \in [a_1, b_1]$ AND $\cdots$ AND $e_m \in [a_m, b_m]$ then return 1 else return 0.

We assumed, for description simplicity, that the above algorithm imCTrain_1 takes as input the class c_0 of the secret data sample d_0, and the algorithm only needed to compute ranges from the data samples d_i such that $cF(d_i) = c_0$. In the case class c_0 is not known, the algorithm can find c_0 as follows. First, it computes ranges from all the data samples d_i such that $cF(d_i) = c$, for all $c \in cS$. Then, the algorithm matches data sample d_0 against the conjunction of range membership statements obtained from all data samples in class c, for all $c \in cS$. Finally, it sets c_0 as the class which maximizes the number of range memberships within the same conjunction.

4.3 Accuracy properties of our Image Classifier

To evaluate the accuracy properties of our image matching classifier, we performed experiments with data values from the following 3 datasets (including one often used dataset with well structured image samples, as well as two less structured datasets, one of which including real-life images with 3D objects):

- The MNIST dataset [16], containing images of handwritten digits; specifically, a training set of 60,000 images and a test set of 10,000 images. The digits have been size-centered and normalized into a fixed-size image. Each image is a 28x28 pixel array, where the value of each pixel is a positive integer in the range [0; 255]. We used 50,000 images for training ad 10,000 images for testing.
- The 'ETL Character Database' [1], a collection of images of about 1.2 million hand-written and machine-printed numerals, symbols, Latin alphabets and Japanese characters and compiled in 9 datasets (ETL-1 to ETL-9). We have used images from this dataset containing the 26 lower-case letters and the 26 upper-case letters from the English alphabet.
- The 'Statlog (Vehicle Silhouettes) Data Set' [18], where the purpose is to classify a given silhouette as one of four types of vehicle (a double decker bus, Chevrolet van, Saab 9000 and an Opel Manta 400), using a set of features extracted from the silhouette. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars. The vehicles may be viewed from one of many different angles. The original purpose was to find a method of distinguishing 3D objects within a 2D image by application of an ensemble of shape feature extractors to the 2D silhouettes of the objects.

We stress that the design possibilities for our image classifiers were heavily constrained within the small class of functions that have an efficient cryptographic obfuscator in the literature. Thus, our classifiers were limited in that they had to be selected from the family of point functions (see, e.g., [8] and [2] for efficient implementations) or wildcard matching classifier (see, e.g., [7] for an efficient implementation).

With respect to the 3 above datasets (briefly denoted as the 'digits', 'letters' and 'vehicles' datasets), in Table 2 we show the true positive rate and true negative rate of our 2 image matching classifiers, as well as 2 baseline classifiers: a random classifier (i.e., a classifier that, on input an image, returns a random class value as output); and k-nearest neighbor (for this latter classifier, however, we do not know of an efficient cryptographic program obfuscator in the literature).

Table 2. Classification metrics for our image matching classifier and 2 baseline classifiers. The true positive rate and true negative rate are defined in Section 2.1. The average rate is simply defined as the average of these two rates.

Datasets	Classifiers	True positive rate	True negative rate	Average rate
Digits	Random	10%	90%	50%
Digits	Conjunction of Range Memberships	84%	85%	84.5%
Digits	k-Nearest Neighbor	89%	90%	89.5%
Letters	Random	98%	2%	50%
Letters	Conjunction of Range Memberships	44%	79%	62%
Vehicles	Random	75%	25%	50%
Vehicles	Conjunction of Range Membership	34%	83%	59%

Main takeaways from this analysis include the following:

1. Even on the 'vehicle' dataset, containing very unstructured and close to real life images, our image classifier IC performs better than the random classifier; moreover, this improvement becomes even larger as we use more structured datasets, like the 'letters' and the 'digits' datasets;
2. On the quite well-structured 'digits' dataset, the average accuracy of our image classifier IC, based on conjunction of range memberships, is no more than 5% less accurate than the much more powerful nearest neighbor classifier (for which however we do not know how to construct an efficient cryptographic program obfuscator); in other words, we showed a classifier for which we can gain the obfuscation property at a very small accuracy loss.

5 Encrypted-Input Obfuscation of Image Matching

In this section we present our encrypted-input obfuscator for the image matching classifier IC from Section 4.2. Formally, we obtain the following

Theorem 2. Let $IM_{ipv,isv}$ be the family of image matching classifiers. If there exists a symmetric encryption scheme, then there exists an encrypted-input program obfuscator for $IM_{ipv,isv}$.

In [10] it was proved that the existence of a symmetric encryption scheme suffices to construct an encrypted-input program obfuscator for the family of conjunctions of range-membership programs. Thus, to prove Theorem 2, it suffices to prove the following

Lemma 1. Let $IM_{ipv,isv}$ be the family of image matching classifiers. and let $CRM_{cpv,csv}$ be the family of conjunctions of range-membership programs. If there exists an encrypted-input program obfuscator for $CRM_{cpv,csv}$, then there exists an encrypted-input program obfuscator for $IM_{ipv,isv}$.

The rest of this section is devoted to the proof of Lemma 1. We start with an informal description of the ideas behind our obfuscator, and then present

Obfuscator description: Denoting as IC = (imCTrain,imCMatch) the image matching classifier to be obfuscated, we have to show an obfuscator icO consisting of 4 algorithms: the key generator $kGen$, the obfuscation generator $oGen$, the input encryptor $iEnc$ and the obfuscation evaluator $oEval$.

The main idea underlying the construction of these 4 algorithms is that icO runs the classifier IC's training algorithm and obfuscates its output using the obfuscator iCO. More specifically, algorithm $oGen$ runs the training algorithm of classifier IC, and obtains the principal component transform matrix W and m ranges $(a_1, b_1), \ldots, (a_m, b_m)$. Then, the matrix W is passed to the input encryptor $iEnc$ and the ranges are passed to the obfuscation evaluator $oEval$, so that the execution of the classifier IC's matching algorithm can be suitably distributed, encrypted and obfuscated by both $iEnc$ and $oEval$, as follows. Algorithm $iEnc$ processes its input by first multiplying it with matrix W and then encrypting it using the input encryptor for obfuscator $crmO$. Algorithm $oEval$ processes its input ranges by obfuscating them using the obfuscation generator for obfuscator $crmO$.

Note that if obfuscator $crmO$ correctly computes a conjunction of range memberships then the execution of classifier IC's matching algorithm is well distributed across $iEnc$ and $oEval$. Moreover, an inspection attacker to algorithm $oEval$ can be turned into an inspection attacker to algorithm $oEval_{crm}$, which is not successful since we assume the existence of an encrypted-input obfuscator for $CRM_{cpv,csv}$. Finally, a black-box attacker to algorithm $oEval$ can be turned into a black-box attacker to algorithm $oEval_{crm}$, but we know from [10] that this attacker is not successful since it cannot produce valid encrypted inputs for $oEval_{crm}$, not having the key used for these encryptions (assuming the existence of symmetric encryption schemes). Now we proceed more formally.

Input to kGen: a security parameter 1^n

Instructions for kGen:

1. run algorithm $kGen_{rm}$ to obtain a random key k
2. Return: key k.

Input to oGen: public parameter values pv(IC)= ($desc$(imCTrain), $desc$(imCMatch)), secret parameter values sv(IC) = (ds, cl, d_0, c_0), and key k

Instructions for oGen:

1. Let $(W, (a_1, b_1), \ldots, (a_m, b_m))$ be the output returned by imCTrain on input the n-image dataset ds, the class labels $cl = (c_1, \ldots, c_n)$, the secret image data sample d_0 and its secret class c_0

2. let $t = m$, $pv = (1^n, 1^t)$ and $sv = (a_1, b_1, \ldots, a_t, b_t)$, and run algorithm $oGen_{crm}$ on input k, pv, sv, thus obtaining $gout_{crm}$, the obfuscated version of sv, and $iaux_{crm}$, the auxiliary input for $iEnc_{crm}$
3. Return: obfuscated program $gout = gout_{crm}$ and auxiliary string $iaux = (W, iaux_{crm})$.

Input to iEnc: key k, public parameter values pv(IC), auxiliary string $iaux$, input string d

Instructions for iEnc:

1. Write auxiliary string $iaux$ as $(W, iaux_{crm})$
2. compute $e = dW$ (i.e., e = vector d times matrix W)
3. let A'_j denote the attribute mapping data sample d to the j-th data block in e, for $j = 1, \ldots, m$
4. set $e = (e_1, \ldots, e_m)$, where $e_j = A'_j(e)$, for $j = 1, \ldots, m$
5. run algorithm $iEnc_{crm}$ on input $k, pv, iaux_{crm}, (e_1, \ldots, e_t)$, thus obtaining as output $iout$, an encrypted version of $(e_1, \ldots, e_t)$
6. Return: encrypted input $iout$.

Input to oEval: public parameter values pv(IC), an obfuscated program $gout$ and an encrypted input string $iout$

Instructions for oEval:

1. run algorithm $oEval_{crm}$ on input public parameter values pv(IC), obfuscated program $gout$ and encrypted input $iout$, thus obtaining $eout$, which is intended to be equal to the output of $CRM_{pv,sv}$ on input $(e_1, \ldots, e_t)$.
2. Return: $eout$.

Obfuscator properties. To complete the proof of Lemma 1, it remains to prove that obfuscator $icO = (kGen, oGen, iEnc, oEval)$ satisfies the properties listed in Definition 3: computation correctness, low runtime overhead, and simulation-based obfuscation.

The computation correctness property of obfuscator icO directly follows from the analogue property of obfuscator $crmO = (kGen_{crm}, oGen_{crm}, iEnc_{crm}, oEval_{crm})$, and by observing that obfuscator icO uses obfuscator $crmO$ to compute the same functionality as the IC classifier. Specifically, the output of algorithm $oEval$ is defined to be equal to the output $eout$ of algorithm $oEval_{crm}$ on input pv(IC), $gout$ and $iout$. By the computation correctness property of obfuscator $crmO$, the output $eout$ is equal to the output of $CRM_{pv,sv}$ on input $(e_1, \ldots, e_t)$. Finally, observe that since $e = dW$, and W is the principal component transformation of the dataset matrix V, the output of $CRM_{pv,sv}$ on input $(e_1, \ldots, e_t)$ is equal to the output of the classifier algorithm imCMatch, by inspection of algorithms imCTrain and imCMatch.

The low runtime overhead property of obfuscator icO follows by algorithm inspection, after observing that the runtime complexity of $oEval$ is essentially the same as that of algorithm $oEval_{crm}$, which, in turn, only requires $O(m \log |D|)$,

where by $|D|$ we denote the size of the public domain that is a superset of all secret input ranges $[a_i, b_i]$, for $i = 1, \ldots, m$.

The simulated-view obfuscation property directly of obfuscator icO directly follows from the simulated-view obfuscation property of obfuscator $crmO$ since $oEval$ just runs $oEval_{crm}$ and returns its output.

6 Conclusions

We extend a recent model to consider encrypted-input cryptographic program obfuscation of matching classifiers, and observe that in this model solutions (based on Yao's garbled circuits) are possible for a very general class of matching classifiers. It remains of interest to produce much more efficient constructions for such a general set of classifiers. We then consider the problem of constructing encrypted-input obfuscators for image matching classifiers and show a new classifier for image matching which both has non-trivial accuracy properties and can be obfuscated and protected against both inspection and black-box attacks. It remains of interest to study what other classifiers can be efficiently obfuscated in this model.

Acknowledgements. Part of this work was supported by the Defense Advanced Research Projects Agency (DARPA) via U.S. Army Research Office (ARO), contract number W911NF-15-C-0233. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, ARO or the U.S. Government.

References

1. National institute of advanced industrial science and technology (aist) and japan electronics and information technology industries association: The etl character database. p. http://etlcdb.db.aist.go.jp/
2. Bahler, L., DiCrescenzo, G., Polyakov, Y., Rohloff, K., Cousins, D.B.: Practical implementations of lattice-based program obfuscators for point functions. In: Int. Conference on High Performance Computing & Simulation, HPCS 2017 (2017)
3. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Proc. of CRYPTO 2001. pp. 1–18 (2001)
4. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. J. ACM **59**(2), 6 (2012)
5. Bellare, M., Stepanovs, I.: Point-function obfuscation: A framework and generic constructions. In: Proc. of TCC 2016-A2. pp. 565–594 (2016)
6. Canetti, R., Micciancio, D., Reingold, O.: Perfectly one-way probabilistic hash functions (preliminary version). In: Proc. of 30th ACM STOC. pp. 131–140 (1998)
7. Cousins, D.B., DiCrescenzo, G., Gür, K.D., King, K., Polyakov, Y., Rohloff, K., Ryan, G.W., Savas, E.: Implementing conjunction obfuscation under entropic ring LWE. IEEE Symposium on Security and Privacy (SP) pp. 68–85 (2018)

8. DiCrescenzo, G., Bahler, L., Coan, B.A., Polyakov, Y., Rohloff, K., Cousins, D.B.: Practical implementations of program obfuscators for point functions. In: Proc. of HPCS 2016. pp. 460–467 (2016)
9. DiCrescenzo, G., Bahler, L., Coan, B.A., Rohloff, K., Polyakov, Y.: Intrusion-resilient classifier approximation: From wildcard matching to range membership. In: Proc. of IEEE TrustCom/BigDataSE 2018
10. DiCrescenzo, G., Bahler, L., McIntosh, A.: Encrypted-input program obfuscation: Simultaneous security against white box and black box attacks. In: 8th IEEE Conference on Communications and Network Security, CNS 2020
11. Dodis, Y., Smith, A.D.: Correcting errors without leaking partial information. In: Proc. of 37th ACM STOC. pp. 654–663. ACM (2005)
12. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proc. of 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1322–1333 (2015)
13. Galbraith, S.D., Zobernig, L.: Obfuscated fuzzy hamming distance and conjunctions from subset product problems. In: Proc. of 17th Int. Conference on Theory of Cryptography, TCC 2019. LNCS, vol. 11891, pp. 81–110. Springer (2019)
14. Hada, S.: Zero-knowledge and code obfuscation. In: Advances in Cryptology - ASIACRYPT 2000, Proc. of 6th Int. Conference on the Theory and Application of Cryptology and Information Security. pp. 443–457 (2000)
15. Jolliffe, I.T.: Principal Component Analysis. Springer Series in Statistics, 2nd edition, Springer (1986)
16. LeCun, Y., Cortes, C., Burges, C.J.: The mnist database of handwritten digits. p. http://yann.lecun.com/exdb/mnist/
17. Lynn, B., Prabhakaran, M., Sahai, A.: Positive results and techniques for obfuscation. In: Proc. of EUROCRYPT 2004. LNCS, vol. 3027, pp. 20–39. Springer (2004)
18. Mowforth, P., Shepherd, B.: Statlog (vehicle silhouettes) data set. pp. https://archive.ics.uci.edu/ml/datasets/Statlog+%28Vehicle+Silhouettes%29
19. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: IEEE Symposium on Security and Privacy, SP 2017. pp. 3–18 (2017)
20. Song, L., Shokri, R., Mittal, P.: Membership inference attacks against adversarially robust deep learning models. In: 2019 IEEE Security and Privacy Workshops, SP Workshops. pp. 50–56 (2019)
21. Souri, H., Dhraief, A., Tlili, S., Drira, K., Belghith, A.: Smart metering privacy-preserving techniques in a nutshell. In: Proc. of 5th ANT and 4th SEIT. Procedia Computer Science, vol. 32
22. Wee, H.: On obfuscating point functions. In: Proc. of 37th ACM STOC, 2005. pp. 523–532 (2005)
23. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under LWE. In: Proc. of 58th IEEE FOCS 2017. pp. 600–611 (2017)
24. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: Proc. of 27th IEEE FOCS 1986. pp. 162–167 (1986)

A Basic notations and definitions

The expression $\{0,1\}^n$ denotes the set of n-bit strings, where n is a positive integer. If S is a set, the expression $x \leftarrow S$ denotes the probabilistic process of

uniformly and independently choosing x from S. If A is an algorithm, the expression $y \leftarrow A(x_1, x_2, \ldots)$ denotes the probabilistic process of running algorithm A on input $x_1, x_2, \ldots$ and any random coins, and obtaining y as output.

A function ϵ over the set of natural numbers $\mathbb{N}$ is *negligible* in n if for every polynomial p, there exists an n_0 such that $\epsilon(n) < 1/p(n)$, for all integers $n \geq n_0$.

Two distribution ensembles $\{D^0_\sigma : \sigma \in \mathbb{N}\}$ and $\{D^1_\sigma : \sigma \in \mathbb{N}\}$ are *computationally indistinguishable* if for any efficient algorithm A, the quantity

$$| \text{ Prob } [\ x \leftarrow D^0_\sigma : A(x) = 1\ ] - \text{Prob } [\ x \leftarrow D^1_\sigma : A(x) = 1\ ] |$$

is negligible in σ (i.e., no efficient algorithm can distinguish if a random sample came from one distribution or the other).

B Cryptographic Program Obfuscation: Previous Models

B.1 Cryptographic Program Obfuscation: Original Model

Cryptographic program obfuscation schemes are usually defined as a pair of algorithms: an obfuscation generator and an obfuscation evaluator. On input the original program, the obfuscation generator returns an obfuscated version of it, called the obfuscated program. On input the obfuscated program and a program input, the obfuscation evaluator, returns an output, which is intended to be the same as the original program's output for this program input. Program obfuscation schemes are required to satisfy the following requirements: preserving the same computation, adding low runtime overhead and offering the same security as a (virtual) black box. The latter says, informally, that any efficient adversary's output bit on input the obfuscated program can be efficiently simulated given access to a black box computing the program. Now we proceed more formally.

Definition 1. We define a *program obfuscation scheme* for family of functions F as a pair $(oGen, oEval)$ of algorithms satisfying the following syntax

1. on input parameter values pv, sv of function $f_{pv,sv} \in F$, the *obfuscation generator* algorithm $oGen$ returns an output, denoted as $gout$, which is intended to be an obfuscated version of $f_{pv,sv}$.
2. on input parameter values pv and the obfuscation $gout$, the *obfuscation evaluator* algorithm $oEval$ returns an output, which we denote as $eout$, which is intended to be the original program's output, when run on input x;

and the following requirements:

1. *Computation Correctness:* Except with very small probability, it holds that $eout = f_{pv,sv}(x)$.
2. *Low Runtime Overhead: oGen*'s runtime is only polynomially slower than the circuit computing $f_{pv,sv}$.

3. *Virtual-Black-Box Obfuscation:* Any efficient adversary's output bit on input the obfuscated program can be efficiently simulated given access to a black box computing the program. A bit more formally: for any efficient adversary Adv given $pv, gout$ as input, there exists an efficient simulator algorithm Sim given pv as input and given oracle access to a black box computing $f_{pv,sv}$, such that the probability that Adv returns 1 and the probability that Sim returns 1 only differ by a negligible amount.

B.2 Cryptographic Obfuscation of Classifiers

We review the definition of cryptographic obfuscation of classifiers, from [9].

In line with the Kerckhoff's principle of modern cryptography, this definition assumes that the description of algorithms CTrain and CMatch is known to an adversary, while the dataset ds and the matching auxiliary input $maux$ returned by CTrain are not. An *intrusion attack* to a classifier is defined as an attack capable of obtaining the matching auxiliary input $maux$ returned by CTrain on input ds and a class number c. Accordingly, classifiers are defined so that an intrusion attack does not leak any true/false property about $maux$, other than what possibly leaked by the capability of performing remote calls to algorithm CMatch$(\cdot, maux)$. In other words, the classifier CMatch$(\cdot, maux)$ would look to an adversary very much like a virtual black box, in the sense of Definition 1. Specifically, this *intrusion-resiliency* notion for classifiers says that an adversary's 1-bit output obtained when given as input the output $maux$ of algorithm CTrain can be simulated, up to a small error, by the output of an efficient algorithm Sim that does not know $maux$ but can query the algorithm CMatch as an oracle. More formally, the classifier (CTrain,CMatch) satisfies *intrusion resiliency* with respect to dataset distribution D if for all probabilistic polynomial-time algorithms A, there exists a probabilistic polynomial-time algorithm Sim with black-box access to Cmatch$(\cdot, maux)$ such that the quantity

$$| \text{ Prob } [\text{ AdvExp}_{\text{IR}}(1^k, c) = 1] - \text{Prob } [\text{ SimExp}_{\text{IR}}(1^k) = 1] |$$

is negligible in the security parameter k, where the probability experiments $\text{AdvExp}_{\text{A}}, \text{SimExp}_{\text{Sim}}$ are detailed below.

$\text{AdvExp}_{\text{IR}}(1^k)$	$\text{SimExp}_{\text{IR}}(1^k)$
1. $c \leftarrow cS$	1. $c \leftarrow cS$
2. $ds \leftarrow D(1^k)$,	2. $ds \leftarrow D(1^k)$,
3. $maux \leftarrow CTrain(ds, c)$	3. $maux \leftarrow CTrain(ds, c)$
4. $b \leftarrow A(maux)$	4. $b \leftarrow Sim^{CMatch(\cdot, maux)}$
5. return: b	5. return: b

B.3 Encrypted-Input Obfuscation: The Model

This model, introduced in [10], extends the original 2-algorithm model of program obfuscation (including an obfuscation generator an an obfuscation evaluator, as reviewed in Appendix B.1) to a 4-algorithm model that additionally

includes a key generator and an input encryptor. The key generator returns a random key to be used by the obfuscation generator to generate its obfuscation of the program and by the input encryptor to generate an encrypted input on which the obfuscation evaluator can be run. The changes to the computation correctness requirement are only of syntactic nature. The runtime requirements extend to the input generation algorithm as well. The obfuscation property is strengthened in that the adversary's output on input the obfuscated program can be efficiently simulated by an algorithm given only public information and, in particular, *without* need for black-box access to the original program.

Definition 2. Let F be a family of functions. We define an *encrypted-input program obfuscator* for F as a 4-tuple $(kGen, oGen, iEnc, oEval)$ of algorithms satisfying the following syntax:

1. on input a security parameter, the *key generator* algorithm $kGen$ returns a random key k
2. on input k, and parameter values pv, sv of function $f_{pv,sv} \in F$, the *obfuscation generator* algorithm $oGen$ returns an output, denoted as $gout$, which is intended to be an obfuscated version of $f_{pv,sv}$, and an output, denoted as $iaux$, intended to be an auxiliary input for the input encryptor algorithm.
3. on input k, parameter values pv of function $f_{pv,sv} \in F$, auxiliary input $iaux$, and input x, the *input encryptor* algorithm $iEnc$ returns an output, denoted as $iout$, intended to be an encrypted version of input x to function $f_{pv,sv}$;
4. on input parameter values pv, the obfuscation $gout$ and the encrypted input $iout$, the *obfuscation evaluator* algorithm $oEval$ returns an output, which we denote as $eout$, which is intended to be the original program's output, when run on input x;

and the following requirements:

1. *Computation Correctness:* Except with very small probability, it holds that $eout = f_{pv,sv}(x)$.
2. *Low Runtime Overhead:* the sum of $iEnc$ and $oGen$'s runtime is only polynomially slower than the circuit computing $f_{pv,sv}$.
3. *Simulated-view Obfuscation:* The obfuscated program returned by algorithm $oGen$ can be efficiently simulated given only the program's public parameters. Formally, there exists a polynomial-time algorithm Sim such that for any function $f_{pv,sv}$ from the class of function F, and any distribution hD returning secret parameter values sv with high min-entropy, the distributions D_{view} and D_{sim} are computationally indistinguishable, where

$$D_{view} = \{\ sv \leftarrow hD;\ k \leftarrow kGen(1^\sigma); gout \leftarrow oGen(k, pv, sv)\ :\ gout\ \},$$
$$D_{sim} = \{\ gout \leftarrow Sim(1^\sigma, pv)\ :\ gout\ \}.$$

C Encrypted-Input Classifier Obfuscation: Formal Model

Let F be a function and let MC = (CTrain,CMatch) a matching classifier for F, as defined in Section 2.1. The *public parameter values* of MC, denoted as pv(MC),

are available to all parties, including an attacker. The *secret parameter values* of MC, denoted as sv(MC), are only available to the obfuscation generator. In our model, we will consider the description of CTrain and CMatch as being known to the attacker, while the dataset, the secret data sample and its class as unknown to the attacker. Thus, we have that $pv = (desc(\text{CTrain}), desc(\text{CMatch}), n, N)$ and $sv = (ds, cl, d_0, c_0)$, where $c_0 = cF(d_0)$. An encrypted-input obfuscator for MC is defined as a 4-tuple of algorithms: a key generator, an obfuscation generator, an input encryptor, and an obfuscation evaluator; which need to satisfy 3 properties: computation correctness, low runtime overhead and simulated-view obfuscation.

Definition 3. Let F be a function and let MC = (CTrain,CMatch) be a matching classifier for F. We define an *encrypted-input program obfuscator for matching classifier MC* as a 4-tuple icO=(kGen,oGen,iEnc,oEval) of algorithms satisfying the following syntax

1. on input a security parameter, the *key generator* algorithm $kGen$ returns a random key k
2. on input key k, public parameter values pv(MC) and secret parameter values sv(MC) for classifier MC, the *obfuscation generator* algorithm $oGen$ returns an output, denoted as $gout$, which is intended to be an obfuscated version of MC, and an output, denoted as $iaux$, which is intended to be an encryption auxiliary input for the input encryptor algorithm $iEnc$.
3. on input key k, public parameter values pv(MC) for MC, encryption auxiliary input $iaux$, and data input d, the *input encryptor* algorithm $iEnc$ returns an output, denoted as $iout$, which is intended to be an encrypted version of input d to algorithm CMatch;
4. on input public parameter values pv(MC), the obfuscation $gout$ and the encrypted input $iout$, the *obfuscation evaluator* algorithm $oEval$ returns an output, which we denote as $eout$. The latter is intended to be equal to CMatch's output, when run on input $d, maux$, where $maux$ has been returned by algorithm CTrain on input ds, cl, d_0;

and the following requirements:

1. *Computation Correctness:* Except with very small probability, it holds that $eout = \text{CMatch}(d, maux)$, where $maux = \text{CTrain}(ds, cl, d_0)$.
2. *Low Runtime Overhead:* the sum of $iEnc$ and $oGen$'s runtime is not significantly slower than the program computing CMatch with public parameter values pv(MC) and secret parameter values sv(MC).
3. *Simulated-view Obfuscation:* The obfuscated program returned by algorithm $oGen$ can be efficiently simulated given only the program's public parameters pv(MC). Formally, there exists a polynomial-time algorithm Sim such that for any matching classifier MC for function F, and any distribution hD returning secret parameter values sv(MC) with high min-entropy, the distributions D_{view} and D_{sim} are computationally indistinguishable, where
   - $D_{view} = \{\, sv(MC) \leftarrow hD;\ k \leftarrow kGen(1^n);$
     $gout \leftarrow oGen(k, pv(MC), sv(MC)) \ :\ gout \,\}$,
   - $D_{sim} = \{\, gout \leftarrow Sim(1^n, pv(MC)) \ :\ gout \,\}$.