Received: 20 March 2019 | Revised: 11 July 2019 | Accepted: 28 August 2019

DOI: 10.4218/etrij.2019-0140

SPECIAL ISSUE

ETRI Journal WILEY

Keyed learning: An adversarial learning framework—formalization, challenges, and anomaly detection applications

Francesco Bergadano

Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy

Correspondence
Francesco Bergadano, Dipartimento di Informatica, Università degli Studi di Torino, Italy.
Email: francesco.bergadano@di.unito.it

Funding information
Università degli Studi di Torino, Grant/Award Number: Ricerca Locale 2019

We propose a general framework for *keyed learning*, where a *secret key* is used as an additional input of an adversarial learning system. We also define models and formal challenges for an adversary who knows the learning algorithm and its input data but has no access to the key value. This adversarial learning framework is subsequently applied to a more specific context of anomaly detection, where the secret key finds additional practical uses and guides the entire learning and alarm-generating procedure.

KEYWORDS
adversarial learning, anomaly detection, keyed learning

1 | INTRODUCTION

Keyed learning is machine learning with a key. This key is a secret and is unknown to an adversary, and it is used as an additional input to the learning system. It can thus be considered a cryptographic key, or more precisely a symmetric key, but it is not used for encryption. Instead, it will influence the learning algorithm and processes in various possible ways, including the choice of features, hyperparameters, and system settings.

The name "keyed learning" is reminiscent of the concept of a "keyed hash function" [1], where a cryptographic key is used as an additional input to a hash function, and the purpose is symmetric message authentication (hence, not encryption). The terms "keyed intrusion detection" and "keyed learning" have been used before [2–8]. However, they were restricted to particular contexts and applications. In particular, in keyed intrusion detection, the secret key is used only as a "word delimiter" in intrusion payloads. In this study, we define keyed learning as a more general concept that affects any form of learning and uses any kind of secret information.

The main purpose of using a key is to make learning unpredictable, that is, we intend to prevent an adversary from simulating the learning process and obtaining a learned classifier similar to the one we will use. This is relevant in anomaly detection, as the adversary will want to devise attacks that remain undetected.

To make this process difficult, we could hide the relevant information used in the learning phase, such as the induction algorithm and its implementation, hypothesis space, or learning bias [9]. Many other important factors could be hidden from the adversary, for example, the subset of available data used for training, and the timing and context of the learning process. Based on Kerckhoffs' principle and the best "open design" practices [10], we concentrate hidden information in a secret key while making the overall learning procedure public. We define a precise and general way to derive all such hidden parameters and information from a secret key. We also provide a simple method for deriving a sufficiently long key for this purpose.

In Section 2, we develop a model (Figure 1) of keyed learning. We also propose a formal definition in (3). The model is later specialized in an implementation-oriented framework, suited for anomaly detection applications (Figure 4). In Section 3, we discuss adversarial models (*Passive Observer*, *Active Data Selector*, and *Active Data Modifier*) and adversarial challenges (*Misclassification Mining*, *Classifier Disclosure*, and *Key Recovery*). In Section 4, we describe an anomaly detection framework and promising application

areas. In the concluding section, we address interesting directions for future research.

2 | DEFINITION AND FRAMEWORK

Cyber security and machine learning often find common applications in the area of anomaly detection. This is a natural class of applications, because adaptive techniques can be used to acquire the description of a security threat automatically. When the current data fit this description, an anomaly is detected. This may allow for some useful counteraction, such as raising an alarm, blocking malicious content, or simply switching to higher protection thresholds and more verbose logs. For examples and surveys of applications, see [11–18].

However, one important issue remains to be addressed: an adversary may prevent defensive actions by working against us in a rational and pre-determined way. For example, he/she could prevent us from learning by tampering with the available data, or even lead us to learn a wrong definition of the target anomaly. In other cases, the adversary will only need to predict the output of the learning phase and use this to devise attacks that will remain undetected. Based on this observation, a relatively new field of research has emerged, known as *adversarial learning* [2–7,19–23] or *adversarial data mining* [24].

We will follow the work of Barreno et al [2] and model adversary actions under three dimensions:

- Causative vs. Exploratory. An adversary performing *causative* attacks will insert malicious data—for instance, some examples that are incorrectly classified—or he/she will alter the probability distribution of incoming examples. An *exploratory* adversary will instead only want to discover information. Most notably, he/she will want to predict the anomaly description that we will finally learn.
- Targeted vs. Indiscriminate. A *targeted* adversary will want a specific instance of an undetected or successful attack because this is required for his/her practical purposes. An *indiscriminate* adversary will be content with any kind of successful attack, proving that the available defenses have been bypassed.
- Integrity vs. Availability. An adversary interested in countering *integrity* will attempt to perform attacks that will not be recognized. When *availability* is our concern, instead, the adversary will attempt to cause the detection system to become unstable or practically unusable. For instance, alarms are caused to behave randomly or learning is made unfeasible (e.g., by performing a denial-of-service (DoS) attack that will substantially limit the number of available examples).

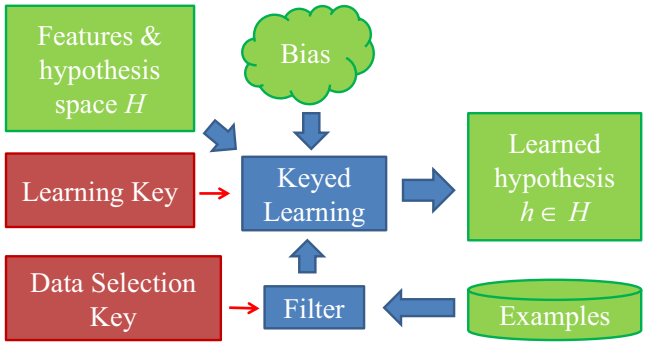

FIGURE 1 Keyed Learning

For anomaly detection applications, the context is often as follows:

- Exploratory: the adversary will want to mimic our learning procedure, or predict the classifier's output, with the aim of performing attacks that will be classified as "normal";
- Targeted: not any undetected pattern is wanted, but one that will correspond to a practically useful attack;
- Integrity: in this case, the adversary may also be interested in DoS or availability issues, but we will concentrate on integrity in this study, where the attacker only intends to find some attack vector that is wrongly classified.

We thus focus on preventing the adversary from succeeding in this exploratory and targeted context. Therefore, our main objective will be to maintain our learning procedure secret and make its results unpredictable.

One approach would be to use *security through obscurity* (STO) and avoid revealing any detail of our learning procedure by maintaining it hidden and private on our servers. STO is generally deprecated in cyber security because it can be an easy target of attacks and social engineering and it is generally clumsy and difficult to maintain (the so-called Kerckhoffs' principle). Consequently, a better approach (see, e.g., [2–7,9,25]) would be to have publicly available learning algorithms, and concentrate the secrets into a cryptographic *key*. Accordingly, the concept of "keyed intrusion detection" [3,6,23,25] has been proposed. In this case, the secret keys are used as simple "word delimiters" of intrusion payloads. In this study, we address a more general concept of keyed learning that applies to any learning methodology, and where keys can affect any component of the learning system. This idea is defined in a more precise way in the following subsections.

2.1 | A model of keyed learning

We will start by providing a "box and arrow" model of keyed learning, which will be useful for a better understanding of this concept. In the following subsection, we will provide a more formal definition.

Based on the standard practice, a machine learning module will receive numerous inputs, for example,

- data or "examples"—in this study, we concentrate on supervised learning, where such examples are associated with a correct *a priori* classification (training data)
- features that can be used and ways of combining them into the description of a target hypothesis (hypothesis space)
- "bias," that is, some preference or precedence to be applied to different possible hypotheses (a part of the available *prior knowledge*)

and it will produce as output a "learned" hypothesis. This hypothesis, when given as input future data, will provide a class label (in concept learning), or a numerical value (in regression).

In keyed learning, we have an additional input: a *key*.

This key, as illustrated in Figure 1, may be divided into at least two components:

- A *data selection key*: this is used to select only some of the available examples, and the adversary will not know which of them have been selected[*].
- A *learning key*: this will influence the learning process so that the adversary will not be able to reproduce it and predict its output.

Subsequently, when discussing applications, we will also consider other forms of keys that may be relevant.

The key is a secret bit sequence known to the learner, but is unpredictable for an adversary. It is used in a way similar to a symmetric key in cryptography, except that it is not used for encryption. The main purpose of using the key is preventing an adversary from guessing the behavior and outputs of the learning process.

The *learning key* will influence the training process in numerous possible ways, including

- select a subset of features—this is not a process of "feature selection" as understood in pattern recognition (aimed at better accuracy), but rather a blind on/off labeling of features based on the key value[†]. To avoid confusion, we will call this step a *keyed feature choice*, allowing "feature selection" to retain its traditional meaning;
- limit the hypothesis space, for example, by restricting the complexity of the learned hypotheses, or by forcing a less expressive language;
- set time/space constraints on the learning algorithm;
- choose a learning algorithm from a set of possible candidates and set its parameters, for example, the maximum depth of trees in a random forest and the number of such trees—the key can also be used as a seed in the generation of required random numbers;
- randomly select a subset of classifiers in ensemble learning;
- inject prior knowledge and select parts of it based on the key, for example, in the form of partially completed classifiers, to be specialized during the learning process [27];
- restrict the bias, for example, by giving preference to a reduced set of hypotheses, more restricted than that suggested by the original input bias;
- select the bias from a possibly large list of candidate bias descriptions.

The adversary will not be able to replicate the learning process because he/she does not know how to set such hyperparameters. There should be a combinatorial number of these choices, indicating that attempting all possible choices is no easier than attempting all possible key values.

2.2 | Definition

Starting from the model in Figure 1, we will now provide a more precise definition of keyed learning. Based on [20], we define supervised learning as the problem of finding f', where:

$$f' = \arg\min_{f \in H} \left(\sum_{<x,y> \in S} g(f(x), y) \right) \quad (1)$$

where H is a hypothesis space, S is a training set, and g is a defined loss function. We need to find a hypothesis f' that minimizes the loss function when evaluated over the training set. However, the real objective is to find such a function f' anticipating that it will also behave correctly in the future (i.e., have a minimal or practically low loss function value for new, previously unseen examples). In other words, we want f' to be predictive. If f' is too closely fitted to the learning examples, or if there are insufficient available data, prediction may be at risk, which is a situation known as "overfitting."

In a simple classification context, x is an example and y is its correct classification, given in the training set. If $a = b$, then $g(a, b) = 0$, and $g(a, b) = 1$ if $a \neq b$, that is, if the classification is incorrect. In other words, we need to minimize incorrect classifications.

[*] In online learning, where examples are produced continuously over time, the data selection key will amount to a secret probability distribution over the set of possible examples

[†] The set of features selected with the key may not be performing well and could be redundant. In particular, some features may be repeated and useless for the aim of prediction. The learning system could perform further "normal" feature selection, after the available features have been determined with the key. Redundant features could then be eliminated. Feature selection, as performed in pattern recognition and machine learning, is usually used to improve the overall performance, may be predictable, and therefore represents a possible vulnerability in adversarial learning [26].

To avoid the above described overfitting problem and use the available prior domain knowledge, a "bias" J is introduced as follows:

$$f' = \arg\min_{f \in H} \left(\sum_{<x,y> \in S} g(f(x), y) + J(f) \right). \quad (2)$$

A hypothesis f is penalized by adding $J(f)$, based on an *a priori* evaluation. Usually, J will be proportional to the syntactic complexity of f, practically avoiding complicated classifiers that are too closely matched around the examples. However, the bias may also be based on prior knowledge so that some hypotheses, even if relatively complex from a syntactic point of view, are given a high preference (a low J value), because they appear sensible based on the available domain knowledge [27].

We now define *keyed learning*:

A definition of *keyed learning*
Given a secret key k, find f' such that

$$f' = \arg\min_{f \in H_k} \left(\sum_{<x,y> \in S_k} g(f(x), y) + J_k(f) \right). \quad (3)$$

The key, as also explained in the previous subsection, will help select numerous hyperparameters, including the chosen hypothesis space $H_k \subseteq H$, and the subset S_k of the training set, which will be used by the learning algorithm. Moreover, it will change the available bias, yielding a new, modified bias J_k.

2.3 | Key generation and parameter choice

We have described how the key, a secret held by the learner only, can be used to select a subset of the available examples, to be identified with a learning set. We have also stated that the secret key can be used to influence the learning process by selecting features and other parameters. However, how can this be achieved in practice? We face two problems.

First, we start from a key k that is sufficiently long to avoid a brute force attack. For instance, we could have a 256-bit key, which is usually considered secure in symmetric cryptography. However, starting from k, we need to set a possibly large number of choices: a subset of the examples, a subset of features, their parameters, the algorithmic constraints, and characteristics. There might not be sufficient bits in k to select all the options directly, and we will then need to extend the original key and generate additional key components.

Second, we need to propose a practical, implementation-oriented scheme for the keyed feature choice phase. We now address these two problems in the following two subsections.

2.3.1 | Key extension

This is a well-known problem in cryptography and many techniques are available [28–30]. We use a simplification of TLS-PRF in this study (transport layer security pseudo random function, see RFC 5246 [28]).

The key k is fed to a pseudorandom function P to obtain an infinite and secret bit stream. The function P also takes as an input a seed value s, an arbitrary bit vector:

$$P(k, s) = \mathrm{HMAC}_k(A_1||s)||\mathrm{HMAC}_k(A_2||s)|| \ldots,$$

where || represents bit vector concatenation and

$$A_0 = s,$$
$$A_i = \mathrm{HMAC}_k(A_{i-1}).$$

We now have an infinite and unpredictable vector of secret bits $P(k, s)$, thus solving the first of the above mentioned problems. The number of possible keys has not increased, as it is still equal to $2^{\mathrm{length}(k)}$.

2.3.2 | Keyed parameter choice

Keyed parameter choice is illustrated in Figure 2, where a scheme is provided for choosing numerous parameters based on the infinite secret bit stream $P(k, s)$.

First, we need to choose a secret subset of the available features $f_1, ..., f_n$. A general solution might be to use the first n bits of $P(k, s)$. However, some features are parametric, or, in other words, they represent a set of features, one for each possible parameter value. For instance, if we want to discriminate a legitimate user of a web application from a robot, we might want to evaluate the feature *clickCount*(p, t, *min*), representing the number of links followed from page p in the application, starting from time t and for the subsequent *min* minutes.

In general, feature $f(n)$ will be chosen with $1 + len(n)$ bits taken from $P(k, s)$, where $len(n)$ is the number of bits required for the parameter n. The first bit will be used to determine whether the feature is to be used at all (1 = it will be used, 0 = it will not be used). The other *len(n)* bits are the value of

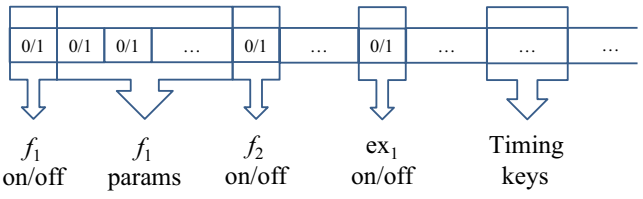

FIGURE 2 Keyed parameter choice

the parameter. This is shown in Figure 2, where the bits used to choose the feature f_1 and its parameters are visible.

After all the features and the corresponding parameters are chosen, we use the following bits in $P(k, s)$ to select a subset of the examples. If there are m examples, we use m bits in $P(k, s)$ to determine which are chosen to be a part of the training set.

The following bits in $P(k, s)$ are used to select the learning hyper-parameters such as the learning algorithm and the parameters of this algorithm (e.g., maximum complexity of the learned hypothesis). Finally, more bits can be used for the overall system and application-specific parameters, such as timing bits (to be addressed in the applications section).

3 | CHALLENGES

We will now define an adversarial model and address the relevant open problems. We will limit our analysis to exploratory adversarial machine learning, as defined in [2] and in Section 2 above.

In this context, we intend to make the learning process unpredictable for the adversary so that he/she will be unable to predict the learned hypotheses or the behavior of a learned classifier. In general, we would like the adversary to infer as little information as possible, ideally no more than he/she could infer by throwing a dice. More precisely and with reference to our definition of keyed learning in (3), he/she should be unable to find any useful information, given the general training parameters (S, H, g, J) and possibly available input/output samples for the learned function f'. This will not always be possible. To make our analysis more precise, we need to define an adversarial model (what the adversary can do) and formalize his/her goals (what kind of information he/she wants to acquire).

3.1 | Adversarial models

We will now define several adversarial models, where different contexts emerge, based on the kind of information available, and how this information may be requested and used. In all the models, with reference to the definition 3, we assume that the adversary has access to the initial hypothesis space H, loss function g, and general bias J. Not all the available examples S may be viewed by the adversary, but often only a subset $S_a \subseteq S$ may be viewed. For example, in network intrusion detection, the adversary may eavesdrop on some of the network traffic, but not all of it. We define the following adversarial models, from the weakest to the most powerful:

- *Passive Observer*

  The adversary can only read some data but cannot modify them or even influence their choice; three subcases are possible.
  - *Learning set observer with unclassified data*
    The adversary views some data x such that $\exists < x, y > \in S_a \subseteq S$, that is, a set of unclassified examples drawn randomly from the learning set. This is very little information, making the adversary's job very difficult.
  - *Learning set observer with classified data*
    The adversary views a stream of randomly chosen examples $< x, y > \in S_a \subseteq S$. In this case, the correct classifications are known, but the adversary does not know how the learned hypothesis will behave on those examples and on future data.
  - *Passive observer of data classified by the learner*
    The adversary views a stream of $< x, f'(x) >$ pairs, where x is a randomly chosen and unclassified example, and $f'(x)$ is the corresponding classification provided by the learner. An eavesdropper in network intrusion detection would fall under this case, being able to view some network data, with some alarms being triggered.
- *Active Data Selector (chosen input attack)*
  The adversary selects the number of inputs x, and is given the corresponding learner's classification $f'(x)$. This is meaningful in some anomaly detection applications, where the adversary can cause unwanted situations, and check if an alarm is triggered.
- *Active Data Modifier (data poisoning)*
  The adversary can insert *poisoned* examples in S, for example, examples with a modified classification to drive the learner away from learning the right hypothesis. This case does not apply in exploratory settings, and we will not refine it any further.

Finally, and following another dimension of adversary modeling, we may consider data and algorithmic complexity. For instance, we might require that the adversary can only view a polynomial number of examples, or assume that he/she will only be able to perform computations that are polynomial in time. If we take a more practical approach, we might require that the adversary reaches a relevant goal before some meaningful time limit.

3.2 | Adversary goals

We will now define possible adversary goals, from the simplest (A), to more difficult (B), and to the most demanding (C). The more difficult the goals, the more harm the adversary might be able to cause.

(A) Misclassification mining
Find a desired input $< x, y >$ that is misclassified by the learner, that is, such that

$$g(f'(x), y) \geq \delta \tag{4}$$

where y is the correct classification of x, and $f'(x)$ is the classification provided by the learned hypothesis.

For binary classifications (e.g., normal vs. anomalous), $\delta = 1$. This is practically relevant in anomaly detection, as the adversary will use an anomaly x as an attack vector, knowing beforehand that it will not be detected.

(B) Classifier disclosure
Find f'' similar to the learned hypothesis f'

$$\sum_{<x,y>\in T} g(f'(x), f''(x)) < \epsilon |T| \quad (5)$$

where the sum is carried over a set T of relevant examples.

Therefore, the average difference of the outputs of f' and f'' is less than an arbitrarily low value ε. This is a more demanding goal than misclassification mining, because we intend to approximate the behavior of the learned classifier, and not only find a case where the learned classifier fails. When $\varepsilon = 0$, the adversary needs to find a hypothesis f'' that behaves exactly like f' on the input set T. If T is the set of all possible examples, the adversary needs to find a hypothesis functionally equivalent to the learned hypothesis.

(C) Key recovery
Find the key k.

If the adversary has at least as much computational power as the learner, he/she might be able to replicate the learning process and obtain the same classifier that the learner would have obtained. It is apparent that the key recovery makes keyed learning useless, as its idea was based on maintaining the key secret. Two questions remain.

- Does Classifier disclosure (B) imply misclassification mining (A)?

  The answer is "yes," if at least one example $x \in T$ is misclassified by the hypothesis f'', which is known to the adversary, and if ε is sufficiently small. In this case, as the learned hypothesis f' is arbitrarily close to f'', x will also be misclassified by f'.

- Does key recovery (C) imply classifier disclosure (B)?

  In many practical applications, this might well be true and probable. However, there might be cases where it does not occur, because although the adversary uses the same key, he/she will not obtain the same classifier, or even a similar one. This may be because the learner and the classifier have different computational powers, or because the learning algorithm uses random inputs (as in the case of ensemble learning).

We will now provide an example where some of these adversarial models and challenges can be instantiated and discussed.

3.3 | Adversarial goals and models: an example

Consider the classification problem in Figure 3. This is an extremely simple example for illustrating adversarial models and challenges. We have two classes ("+" and "–"), where class "+" is identified by the classifier $f_1 < f_2$. We also suppose we have eight available examples in the learning set S, four positive and four negative, as also shown in Figure 3.

With reference to our keyed learning framework, having two non-parametric features f_1 and f_2, a two-bit key k will suffice for keyed feature choice:

00 ⇒ no feature can be used,
01 ⇒ only f_2 is used,
10 ⇒ only f_1 is used,
11 ⇒ both features are used.

If k = 11 and a sufficient number of examples are available, it is evident that learning is an easy linear discrimination problem for both the learner and the adversary. Consequently, the adversary will win the game, being able to approximate the learner's classifier, and hence predict its use (classifier disclosure as defined in the previous section). In general, when learning is feasible and data abound, we face a difficult situation in an adversarial context, because everyone converges to the correct classifier and no secrets can be maintained.

Let us now consider $k = 10$, that is, the learner can use only f_1. We must assume that the adversary does not know k. In this simple case, the adversary could conduct a brute force attack and attempt all possible values of the two bits in k. In real applications, we will have a sufficiently long key to avoid this scenario. Let us then assume here, for the sake of discussion,

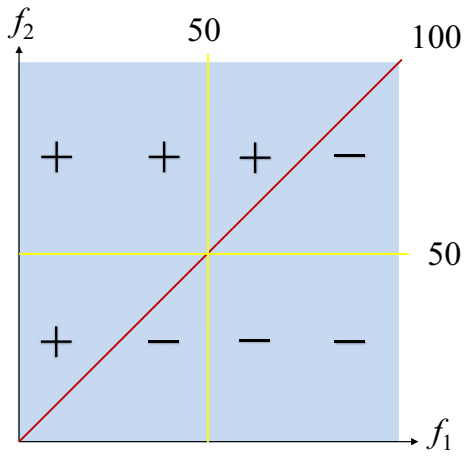

FIGURE 3 A simple classification problem with two features

that the adversary cannot attempt all possible values of k. Given the eight examples of Figure 3, and based on the employed algorithms and bias, the learner will produce a classifier such as $f_1 < 50$ (the vertical line in the figure), having 75% accuracy on the learning set. The adversary will only be able to either attempt a random value of k or decide to use all the available features if sufficient data and computational power are available ($k = 11$). We will now consider two interesting cases for adversarial models and goals, applied to this example.

3.3.1 | Passive learning set observer and classifier disclosure

The adversary does not know the key and views no more than the eight examples of Figure 3. If a random value of the key is selected, the adversary will obtain

$k = 00 \Rightarrow$ random classification: the learner's classifier is not disclosed, because it differs from the adversary's classifier in 50% of the cases.

$k = 01 \Rightarrow$ classifier $f_2 > 50$: the two classifiers differ in 50% of the cases, again the adversary fails.

$k = 10 \Rightarrow$ the adversary and learner converge to the same classifier as more examples are viewed. This indicates that classifier disclosure is achieved if the threshold ε in (5) is sufficiently small and sufficient examples are processed.

$k = 11 \Rightarrow$ the correct classifier $f_2 > f_1$ is learned by the adversary, but it differs from the learner's classifier in 25% of the cases. Consequently, the adversary fails on classifier disclosure.

Consequently, the adversary has only a 25% probability of achieving classifier disclosure (one out of the four above cases), which is the same probability of flipping two coins and selecting the right key. Notably, higher classifier power (in this case, being able to use all the features and learning the correct classification) does not help him/her achieve his/her goals.

3.3.2 | Active data selector and key recovery

Let us assume that the adversary cannot conduct a full brute force attack (attempt all possible key values and hence attempt all feature combinations), but assume that he/she can attempt one feature at a time. We consider a chosen input attack. By using the chosen example $<f_1 = 40, f_2 = 25>$ and asking the learner to classify it, the adversary will obtain a wrong classification (–) and thus exclude the use of f_2 as a feature. This will exclude the key values 01 and 11. A few more attempts will also probably exclude random classification ($k = 00$). The adversary has then detected the correct key $k = 10$ and wins the game.

One could attempt to analyze more cases, by combining the five adversarial models of Section 3.1 and the three adversary goals of Section 3.2. Different results can be obtained depending on which and how many examples are available.

3.4 | Randomization

In numerous recent works (e.g., [4,31,32]), the concept of *randomization* was used in the context of adversarial learning, with motivations similar to the present *keyed learning* framework. In particular, randomization aims at hiding information from an adversary by introducing random perturbations in the learned classifiers.

In such works, a game theoretic approach was often used and an adversary utility function is part of the definition. We provide below an alternative formalization simplified under the assumption that the goal of the adversary is classifier disclosure as defined in (5):

Bounded randomized learning

Find f' such that

$$f' = \arg\min_{f \in H} \left(\sum_{<x,y> \in S} g(f(x), y) + J(f) \right). \quad (6)$$

Output a randomized classifier $R(f')$, where

$$R(f')(x) = f'(x) + r(x, f'(x), seed) \quad (7)$$

and r is an a priori defined and efficiently computable random function.

We also require r to be *bounded*

$$\sum_{<x,y> \in S} |r(x, f'(x), seed)| < \varepsilon |S| \quad (8)$$

where ε is an appropriately chosen and constant value.

The requirement for r to be *bounded* corresponds to a performance requirement for the resulting randomized classifier $R(f')$, because a large random perturbation of f' might cause a degradation in the correctness of the classifier.

We now discuss the differences in the two approaches (keyed and randomized learning) and observe whether the reduction properties hold. We first define the concept of *monotonic* keyed learning as follows:

A keyed learning algorithm is *monotonic* with respect to a set of examples S, if, for any pair of keys $< k1, k2>$, the classifiers f_{k1} and f_{k2} learned with the two keys are similar:

$$\sum_{<x,y> \in S} |f_{k1}(x) - f_{k2}(x)| < \varepsilon |S|. \quad (9)$$

The example of keyed learning in Section 3.3 is not monotonic for $\varepsilon \geq 0.5$ as the classifiers obtained with keys 01 and

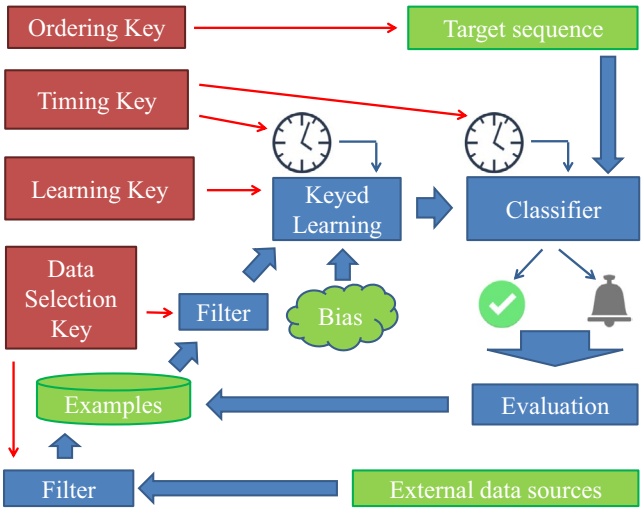

FIGURE 4 Anomaly detection framework with keyed Learning

10 differ in four out of the eight examples in the given learning set S.

In general, we do not expect monotonicity to apply for all possible key pairs, as it would be too strong a requirement. After fixing a learning key $k1$, we could instead require keyed learning to be *weakly monotonic*:

$$\sum_{<x,y>\in S} |f_{k1}(x), \mu_k(f_k(x))| < \varepsilon |S| \tag{10}$$

in this case, the effect of changing $k1$ is bounded only when averaging out over all possible alternative keys. The example of keyed learning in Section 3.3 is weakly monotonic with respect to $k1 = 01$, for $\varepsilon > (10/4)/8 = 0.313$, where 10/4 is the average classification difference between $k1$ and each possible key value, and 8 is the number of examples in the learning set S.

We may now formalize two reduction properties.

Reduction property 1: *Monotonic keyed learning can be reduced to bounded randomized learning.*

Proof: We declare a base key $k1$ and publish it so that the adversary will know it. We and the adversary can then compute a classifier f_{k1}. Based on the definition of keyed learning, we then select a random, secret key $k2$, and obtain a learned classifier f_{k2}. This can be written as a randomized classifier as follows:

$$f_{k2}(x) = R(f_{k1})(x) = f_{k1}(x) + r(x, f_{k1}(x), k2) \tag{11}$$

where $r(x, f_{k1}(x), k2) = f_{k2}(x) - f_{k1}(x)$.

As keyed learning is assumed to be monotonic, we also have

$$\sum_{<x,y>\in S} |r(x, f_{k1}(x), k2)| = \sum_{<x,y>\in S} |f_{k2}(x) - f_{k1}(x)| < \varepsilon |S| \tag{12}$$

and hence, the keyed learning of f_{k2} matches the definition of bounded randomized learning.

Reduction property 2: *Weakly monotonic keyed learning can be reduced to bounded randomized learning if the adversary can perform learning for all keys.*

Proof: The adversary can learn f_k for all possible keys k, and hence compute $\mu_k(f_k(x))$ for any input x. We denote by $\mu_k(f_k)$ the corresponding classifier function. Based on the definition of keyed learning, we then select a random, secret key $k2$, and obtain a learned classifier f_{k2}. This can be written as a randomized classifier as follows:

$$f_{k2}(x) = R(\mu_k(f_k))(x) = \mu_k(f_k(x)) + r(x, \mu_k(f_k)(x), k2) \tag{13}$$

where $r(x, \mu_k(f_k)(x), k2) = f_{k2}(x) - \mu_k(f_k(x))$.

As keyed learning is assumed to be weakly monotonic, we also have

$$\sum_{<x,y>\in S} |r(x, \mu_k(f_k)(x), k2)| = \sum_{<x,y>\in S} |f_{k2}(x) - \mu_k(f_k(x))| < \varepsilon |S| \tag{14}$$

and hence the keyed learning of f_{k2} matches the definition of bounded randomized learning.

However, in many practical contexts, keyed learning is not monotonic, as changes in the key produce unexpected and potentially profound differences in the learned classifiers. Moreover, keyed learning will pose an additional burden on the adversary, related to the combinatorial effort required to attempt all possible keys. Consequently, keyed learning and randomization are usually distinct concepts with different characteristics.

4 | APPLICATIONS

Applications of keyed learning fall within the scope of exploratory adversarial learning [2]. This context is generally appropriate for anomaly detection, which comprises several application domains, including intrusion detection [3,5,6,14,25,33,34], attack and malware analysis [7,16,35–37], defacement response [8,17,38,39], Web promotional infection detection [40], and biometric and continuous user authentication [11,18].

We will now provide a general methodology for keyed anomaly detection based on the architecture of Figure 4. Subsequently, we will list some specific application contexts that fall within this scheme.

Figure 4 is derived from our general keyed learning framework of Figure 1; however, it includes implementation-oriented components and is specialized for anomaly detection applications. In particular, we must consider the

fact that, in anomaly detection applications, data are often produced continuously over time. This applies both to the learning examples and to current data sequences that need to be classified as either *normal* or *anomalous*. Such current data sequences are located in the "Target sequence" box of Figure 4, and they might be generated by several sensors and logging agents, often running at high speeds and in parallel.

At some point, one of these target data sequences is selected and the anomaly detection system will classify it, thus generating an "ok, all normal" classification, or an alarm. This possible alarm will then be handled by a security incident and event management (SIEM) software, and ultimately and when required, by human intervention through a security operations center (SOC) facility.

We first observe that the selection of this target sequence from a possibly large set of available data should be secret. The adversary would use any such information and will avoid security attacks that will fall within the selected target sequences. In keyed anomaly detection, we will use a key component (named "ordering key" in Figure 4) to select a data sequence secretly for processing.

The time at which this action is taken will also be maintained secret. As seen in many prison-break movies, if the guards check the inmates at regular, predictable intervals, it will be a huge advantage when inmates plan an escape. Therefore, the time at which anomaly classification is undertaken should be unknown to the adversary, and we will use another key component for this purpose (named "timing key" in Figure 4).

Second, keyed learning will also be repeated over time, because its input data will change: the session key might change owing to possible compromise, the bias might change, and especially more and different examples will become available. For the same reasons described for the classifier above, the timing choice that determines when the learning process is iterated should also be maintained secret. Therefore, in Figure 4, the timing key is also used to trigger the keyed learning phase.

Finally, the examples are not viewed as an immutable block of data (as in the simplified scheme of Figure 1), but as a continuously fed database, originating from diverse data sources. These sources include external data generators and system/user feedback. In particular, the classifier outputs may be analyzed and evaluated by other system components (e.g., SIEM software) and SOC operators yielding a revised and authoritative classification. This becomes a new example, and is fed back to the system for future use.

The general keyed anomaly detection framework described here can be applied to several specific domains, as described below.

4.1 | Network intrusion detection

Target sequences are data payloads [3,7], often arriving at high speeds and from different sub-networks and logging agents [14]. We focus on a data subset at a time and extract a target sequence to be monitored. The time at which this check is performed is maintained secret and driven by the timing key. Historic intrusion examples are provided by external data sources. The example database is enriched by SOC operators, who manually check some of the target data and label them with a correct classification.

4.2 | Continuous user authentication

Target data contain sequences of user behavior information, including keystroke dynamics [11], smartphone location, and smartphone use parameters (e.g., accelerometer, the frequency and type of touchscreen use, audio) [28]. Some of these data sequences are selected for classification: if anomalous, an additional authentication factor is required, such as a password, and if correct, the example is re-labeled as normal and fed back to the example database. If the user cannot input a correct password, the example is labeled as anomalous and again fed back to the database; in this case, an external alarm is also generated.

4.3 | Defacement response

Defacement detection and response is an important application that can be considered a form of reverse CAPTCHA [8,41]. The target sequence is a set of URLs within the target website. We generate such URLs by crawling the web application. Lower weights are provided as we move further from the document root so that the home page and top-level pages of the application will be checked more frequently, because they usually have higher reputational impact. The timing key will be combined with such weights to trigger and feed the classifier.

Adversarial approaches to defacement response have been addressed in [8,9], and this is a fitting application for keyed learning. The adversary is willing to deface our website and can attempt to pass undetected by predicting the alarm system behavior. We make this more difficult by using timing and learning keys.

5 | CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

We have described a general scheme for keyed learning (Figure 1), and specialized it in the previous section for the context of anomaly detection applications (Figure 4). We have also formalized and defined the notion of keyed learning, and then used the same notation to describe alternative adversarial models and to define possible adversary goals.

The following broad areas of future research are of interest.

5.1 | Study the effect of key size on learning performance

A sufficiently large key will usually be generated as an unpredictable sequence of random bits, and it will influence learning (see Figure 1). However, this could be a "bad influence" for the sake of prediction. For example, a random choice of all available features could exclude meaningful concept descriptions, and a random choice of hyperparameters may be determined to be a bad choice. In future research, algorithms and methodologies should be devised to prevent this effect. One strategy could be as follows: (a) select a key, (b) evaluate performance through cross-validation on the available data, and (c) go back to step 1 if the performance on the test set is below a certain threshold. However, this might be a never-ending loop, or it might require an exponential number of trials.

5.2 | Analyze the effect of learnability on unpredictability

In a world where there are sufficient examples and learning is feasible, keyed learning does not work. This is because both the learner and the adversary will converge to a correct hypothesis, that is, they both reach the truth, perhaps in slightly different ways. Consequently, the adversary will predict the learner, in the sense of classifier disclosure as defined in (5). We then need to understand and formalize situations where learning is possible but not perfect, and especially where learning can produce equally valid, but functionally distinct results, depending on the chosen hyper-parameters. This can be analyzed, for example, in a probably approximately correct learnability framework (PAC-learnability).

5.3 | Define key effectiveness with respect to unpredictability

Keyed learning will prevent a simulation of the learning phase by the adversary. However, even in a case where perfect prediction may not be achieved, the adversary could use a different hypothesis space to obtain a similar classifier, as in (5). We should then devise methods and define cases where different keys yield results that are different not only formally, but also functionally, in the sense that they will produce classifications that are substantially different on new data. In particular, we should formalize the notion of a "functionally relevant feature set," a set of features producing a distinctive learned hypothesis[‡].

[‡]Again, this is not a relevant feature set as intended in pattern recognition, where the goal is only prediction. Here, we also have a prediction goal, but our learned hypotheses must also be different from others that would be obtained with a different key choice.

In addition to the above general areas of investigation, we suggest, as an interesting and specific target of future research, the following approach:

1. Define a hypothesis space or language, for example, random forests with n real-valued features, or k-DNF Boolean expressions.
2. Select an adversarial model from among those defined in Section 3.1, for example, *passive observer*, or *active data selector*.
3. Determine whether the adversary goals of Section 3.2 can be reached, for example, *misclassification mining* or *key recovery*.

Finally, keyed learning can be used practically and experimentally in many of the above-cited anomaly detection applications, such as in intrusion detection [3] and defacement response [8]. It could be integrated effectively in SIEM software and in security monitoring infrastructures that may generate alarms while avoiding adversarial counteractions.

REFERENCES

1. M. Bellare, R. Canetti, and H. Krawczyk, *Keying hash functions for message authentication*, in Proc. Auun. Int. Cryptology Conf., Santa Barbara, CA, USA, Sug. 1996, pp. 1–15.
2. M. Barreno et al., *Can machine learning be secure?* in Proc. ACM Symp. Inf., Comput. Commun. Security (AsiaCCS), ACM, Taipei, Taiwan, Mar. 2006, pp. 16–25.
3. R. S. Mrdovic and B. Drazenovic, *KIDS: a Keyed Intrusion Detection System*, in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment (DIMVA), IEEE, Bonn, Germany, July 2010, pp. 173–182.
4. B. Biggio, G. Fumera, and F. Roli, *Adversarial pattern classification using multiple classifiers and randomization*, in Proc. Joint IAPR Int. Workshop Structural, Syntactic, Statistical Pattern Recogn., Springer, Orlando, FL, USA, Dec. 2008, pp. 500–509.
5. K. Wang, J. Parekh, and S. Stolfo, *Anagram: a Content Anomaly Detector Resistant to Mimicry Attack*, in Proc. Int. Conf. Recent Adv. Intrusion Detection, Springer, Hamburg, Germany, Sept. 2005, pp. 226–248.
6. V. M. Lomte and D. Patil, *Survey on keyed IDS and key recovery attacks*, Int. J. Sci. Research **4** (2015), no. 12, 846–849.
7. R. Perdisci et al., *McPAD: a multiple classifier system for accurate payload-based anomaly detection*, Comput. Netw. **53** (2009), no. 6, 864–881.
8. F. Bergadano et al., *Defacement response via keyed learning*, in Proc. Int. Conf. Inf. Intell. Syst. Applicat., Larnaca, Cyprus, Aug. 2017, pp. 1–6.
9. G. Davanzo, E. Medvet, and A. Bartoli, *Anomaly detection techniques for a web defacement monitoring service*, Expert Syst. Applicat. **38** (2011), no. 10, 12521–12530.
10. K. Scarfone, W. Jansenamd, and M. Tracy, *Guide to General Server Security, Section 2.4*, Special Publication 800–123, NIST, Gaithersburg, MD, 2008.

11. F. Bergadano, D. Gunetti, and C. Picardi, *Identity verification through dynamic keystroke analysis*, Intell. Data Analysis J. **7** (2003), no. 5, 469–496.
12. G. Ruffo and F. Bergadano, *Enfilter: a password enforcement and filter tool based on pattern recognition techniques*, in Proc. Int. Conf. Image Analysis Process., Cagliari, Italy, Sept. 2005, pp. 75–82.
13. S. Y. Ooi, S. C. Tan, and C. W. Ping, *Anomaly Based Intrusion Detection through Temporal Classification*, in Proc. Int. Conf. Neural Inf. Process., Kuching, Malaysia, Nov. 2014, pp. 612–619.
14. J. Kim et al., *A lightweight network anomaly detection technique*, in Int. Conf. Comput., Netw. Commun.(ICNC), Santa Clara, CA, USA, Jan. 2017, pp. 1–5.
15. G. Wang, J. Yang, and R. Li, *Imbalanced SVM based anomaly detection algorithm for imbalanced training datasets*, ETRI J. **39** (2017), no. 5, 621–631.
16. P. Parrend et al., *Foundations and applications of artificial Intelligence for zero-day and multi-step attack detection*, EURASIP J. Inf. Security **4** (2018), 1–21.
17. F. Maggi et al., *Investigating web defacement campaigns at large*, in Proc. Asia Conf. Comput. Commun. Security, Incheon, Rep. of Korea, June 2018, pp. 443–456.
18. C. Shen et al., *Touch-interaction behavior for continuous user authentication on smartphones*, in Proc. IEEE Int. Conf. Biometrics, Phuket, Thailand, May 2015, pp. 157–162.
19. M. Kearns and M. Li, *Learning in the presence of malicious errors*, SIAM J. Comput. **22** (1993), no. 4, 807–837.
20. D. Lowd and C. Meek, *Adversarial Learning*, in Proc. ACM Conf. Knowledge Discovery Data Mining, ACM, Chicago, IL, UDS, Aug. 2005, pp. 641–647.
21. L. Huang et al., *Adversarial machine learning*, in Proc. ACM Workshop Security Artif. Intell., Chicago, IL, USA, Oct. 2011, pp. 43–58.
22. N. Šrndic and P. Laskov, *Practical evasion of a learning-based classifier: A case study*, in Proc. IEEE Symp.Security Privacy, San Jose, CA, USA, May 2014, pp. 197–211.
23. J. E. Tapiador et al., *Key-recovery attacks on KIDS, a keyed anomaly detection system*, IEEE Trans. Dependable Secure Comput. **12** (2015), no. 3, 312–325.
24. C. Aggarwal, J. Pei, and B. Zhang, *On privacy preservation against adversarial data mining*, in Proc. ACM SIGKDD Int. Conf. Knowled. Discovery Data Mining, ACM, Philadelphia, PA, USA, Aug. 2006, pp. 510–516.
25. R. Bendale et al., *KIDS: Keyed Anomaly Detection System*, Int. J. Adv. Eng. Res. Dev. **12** (2017), 312–325.
26. H. Xiao et al., *Is feature selection secure against training data poisoning?* in Proc. Int. Conf. Mach. Learn., Lille, France, July 2015, pp. 1689–1698.
27. F. Bergadano and A. Giordana, *A knowledge intensive Approach to concept induction*, in Proc. Fifth Int. Conf. Mach. Learn., Margan Kaufmann Publishers, Ann Arbor, MI, USA, June 1988, pp. 305–317.
28. T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, IETF, 2008, https://doi.org/10.17487/rfc5246.
29. J. Arkkom et al., MIKEY: *Multimedia Internet KEYing*, RFC3820, IETF, 2004, https://doi.org/10.6028/NIST.SP.800-108.
30. L. Chen, *Recommendation for Key Derivation Using Pseudorandom Functions*, NIST Special Publication 800–108, NIST, 2009.
31. Y. Vorobeychik and B. Li, *Optimal randomized classification in adversarial settings*, in Proc. Conf. Autonomous Agents Multiagent Syst., Paris, France, May 2014, pp. 485–492.
32. S. Rota Bulò et al., *Randomized prediction games for adversarial machine, learning*, IEEE Trans. Neural Netw. Learn. Syst. **28** (2017), no. 11, 2466–2478.
33. N. Munaiah et al., *Are Intrusion Detection Studies Evaluated Consistently? A Systematic Literature Review*, Technical Report, University of Rochester, 2016.
34. S. Anwar et al., *From intrusion detection to an intrusion response system: fundamentals, requirements, and future directions*, MDPI Algorithms J. **10** (2017), no. 39, 1–24.
35. S. Kim and B. B. Kang, *FriSM: malicious exploit kit detection via feature-based string-similarity matching*, in Proc. Int. Conf. Security Privacy Commun. Netw., Singapore, Aug. 2018, pp. 416–432.
36. Y. Bae, I. Kim, and S. O. Hwang, *An efficient detection of TCP Syn flood attacks with spoofed IP addresses*, J. Intell. Fuzzy Syst. **35** (2018), no. 6, 5983–5991.
37. Q. T. Hai and S. O. Hwang, *An efficient classification of malware behavior using deep neural network*, J. Intell. Fuzzy Syst. **35** (2018), no. 6, 5801–5814.
38. K. Borgolte, C. Kruegel, and G. Vigna, *Meerkat: Detecting website defacements through image-based object recognition*, in Proc. USENIX Security Symp., Washington, D.C., USA, Aug. 2015, pp. 595–610.
39. A. Bartoli, G. Davanzo, and E. Medvet, *A framework for large-scale detection of web site defacements*, ACM Trans. Internet Technol. **10** (2010), no. 3, 1–3.
40. X. Liao et al., *Seeking nonsense, looking for trouble: efficient promotional-infection detection through semantic inconsistency search*, in Proc. IEEE Symp. Security Privacy, San Jose, CA, USA, May 2016, pp. 707–723.
41. A. Basso and F. Bergadano, *Anti-bot strategies based on human interactive proofs*, in Handbook of Information and Communication Security, Springer, New York, 2010, pp. 273–291.

AUTHOR BIOGRAPHY

Francesco Bergadano obtained his PhD degree in computer science from the Universities of Milan and Turin. He has then worked as a visiting Professor at George Mason University (USA), an Associate Professor at the University of Catania (Italy), and now a Full Professor at the University of Turin (Italy). In the past, he has been a consultant for companies and public administrations, and a principal investigator in several national and EU-funded research projects. More recently, he was an evaluator for EU H2020 proposals. He has authored more than 120 publications in the areas of machine learning and IT security. His current research interests are in the area of cyber security, including adaptive anomaly detection, continuous and biometric user authentication, security analytics, and mobile application security.